國立臺北科技大學

2020 Spring 資工系物件導向程式實習

期末報告

冰火人

第 36 組

108590021 黃品宥

108590025 劉立傑

目錄

一、 簡介

1. 動機

這學期的物件導向程式設計實習是上學期物件導向程式設計課程的實作，運用了 GameFramework 的框架實作一款遊戲，想到了以前遊玩過的冰火姊弟覺得有許多不同功能的物件可以挑戰，就試著製作看看。

2. 分工

黃品宥：冰火湖類別、通關門類別、拉桿升降類別，素材修剪，

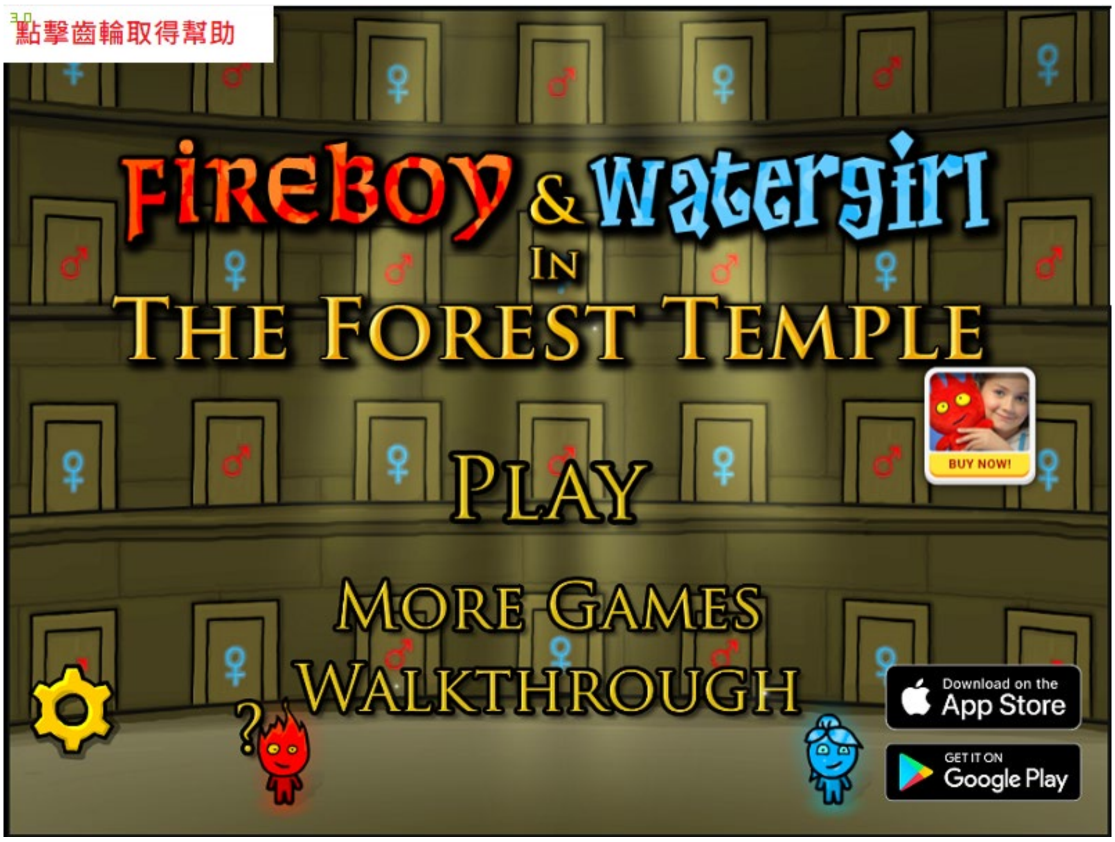
劉立傑：冰火姊弟類別、重力世界、地圖、可推動方塊類別，遊戲選單，遊戲說明

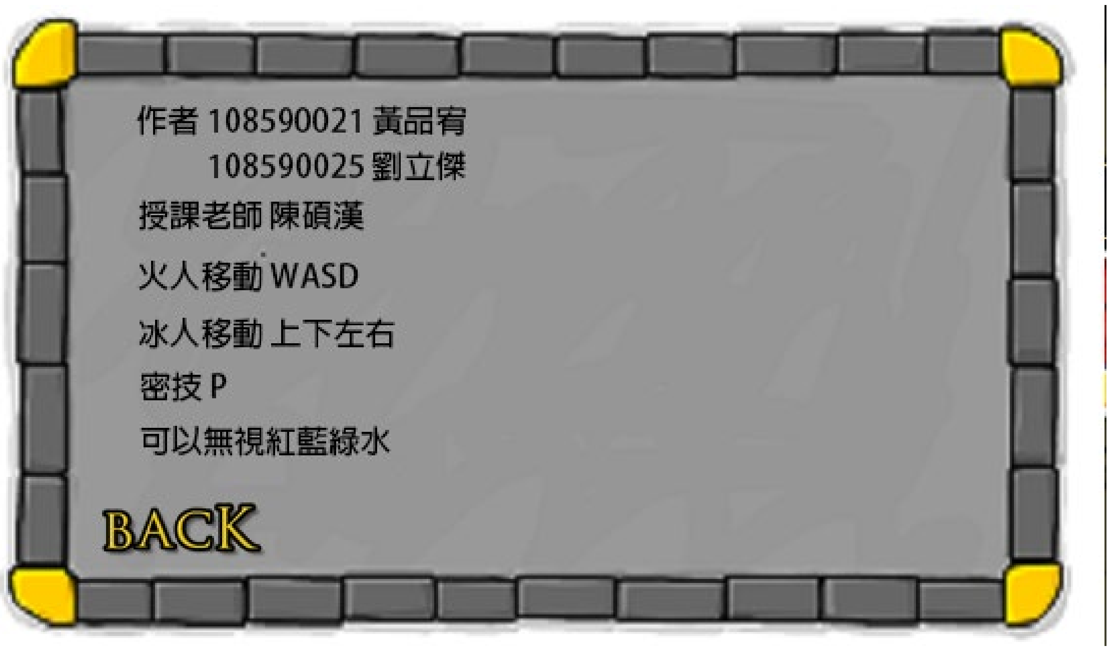
二、 遊戲介紹

1. 遊戲說明

這個是一個雙人遊戲，以 WSAD 操控冰人，上下左右方向鍵操控火人，經過各種關卡，以及吃冰火鑽石，最後跑到通關門來通關，火人可以吃紅色鑽石，可以碰紅色水，不能吃藍色鑽石，不能碰藍色水，冰人反之，冰人與火人都不能碰綠色水，密技方面是按下 P 可以無視紅藍綠水直接通過。

2. 遊戲圖形




作者 108590021 黃品宥

108590025 劉立傑

授課老師 陳碩漢

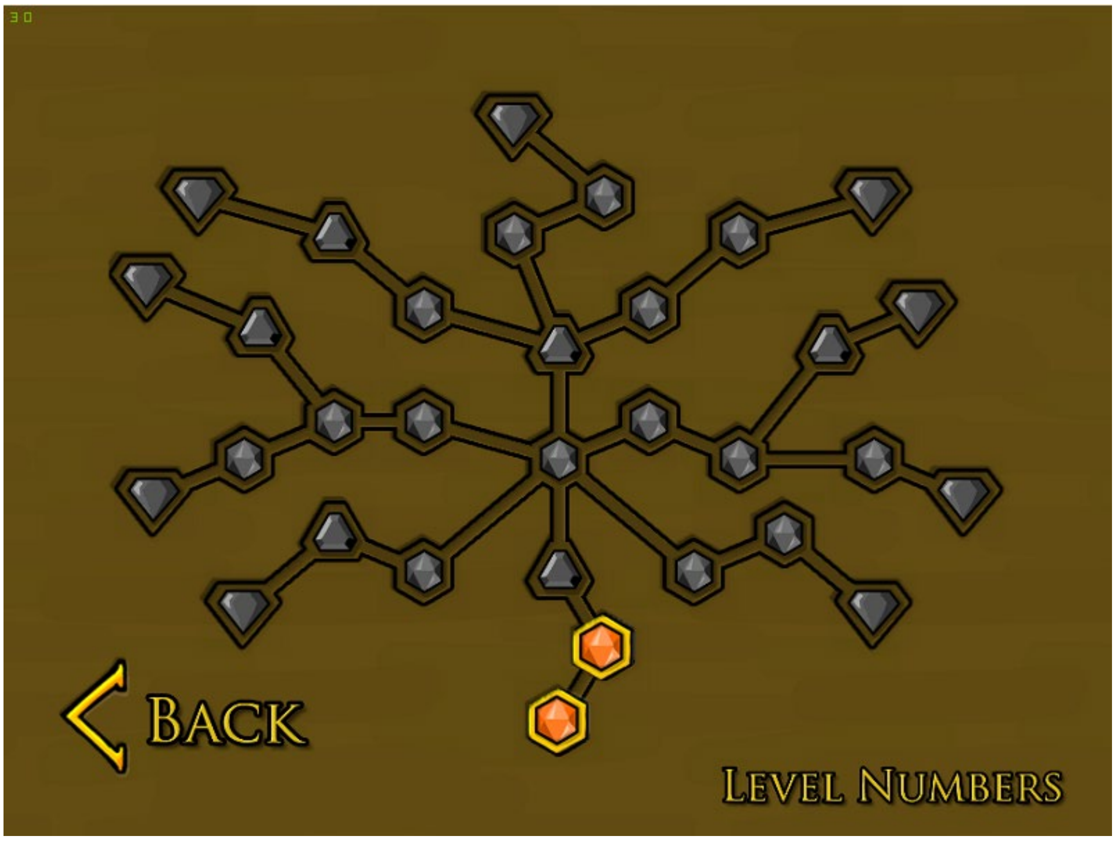
火人移動 WASD

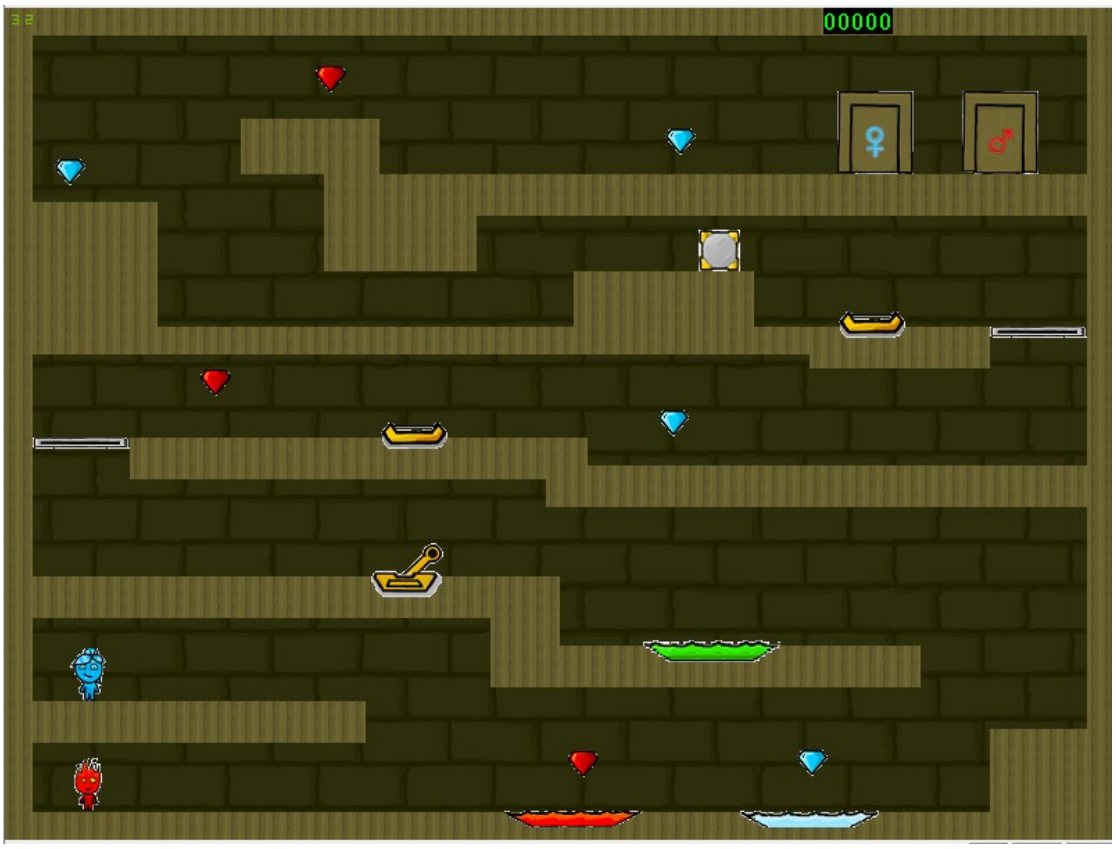
冰人移動 上下左右

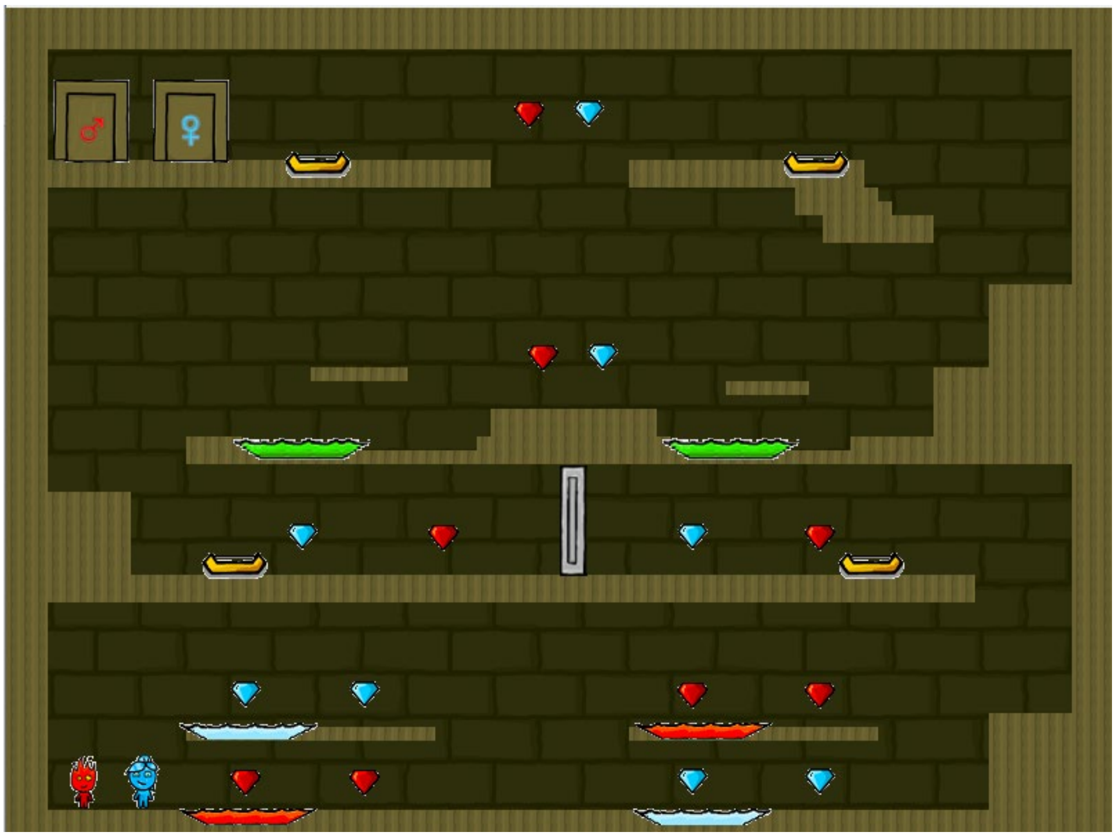
密技 P

可以無視紅藍綠水

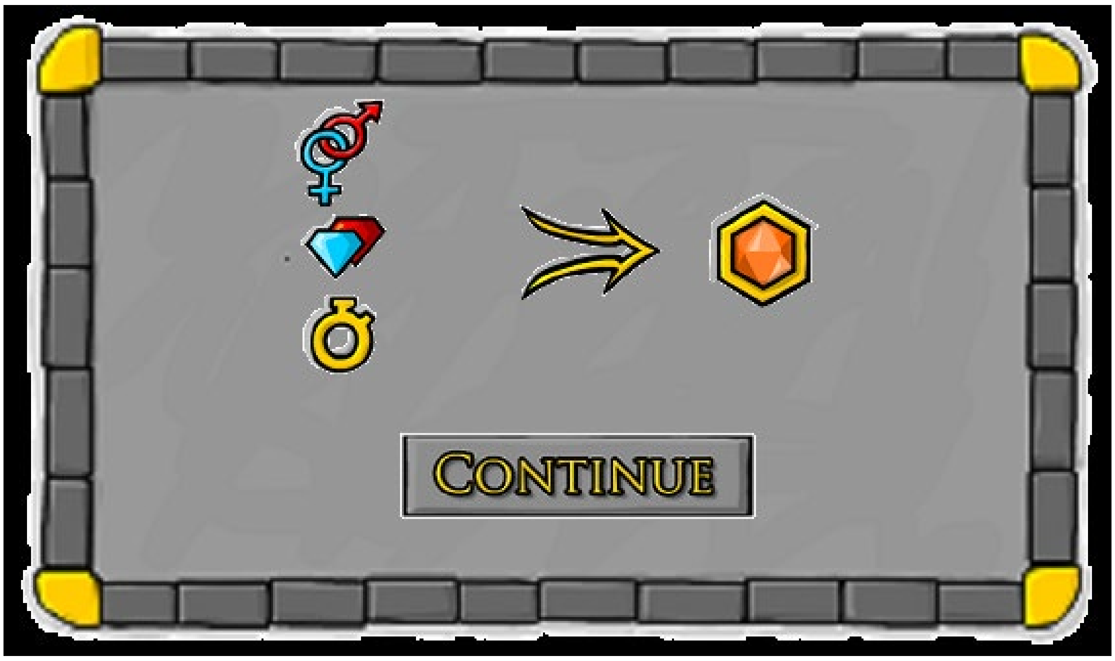
BACK


30
Back
Level Numbers


32
00000



GAME OVER
MENU
RETRY



3.　遊戲音效

遊戲背景音效 LevelMusic.mp3

三、　程式設計

1.　程式架構

我們的程式分為 6 個 GameState:

GameStateInit	遊戲大廳
GameStateMenu	遊戲關卡選單
GameStateRun	第一關
GameStateRun2	第二關
GameStateWin	遊戲贏
CGameStateOver	遊戲輸

(1) GameStateInit

在 GameStateInit 中點擊齒輪會載入遊戲說明圖片，按下 Back，說明圖片

會消失，點擊 Play 會跳轉到 GameStateMenu。

(2) GameStateMenu

在 GameStateMenu 中點擊最下面的關卡會進入 GameStateRun(第一關)，往上點擊一個關卡會進入 GameStateRun2(第二關)，點擊 Back 會返回 GameStateInit。

(3) GameStateRun

在 GameStateRun 中會生成冰火人與許多物件,物件之間的碰種會導致物件屬性的改變與 GameState 的跳轉。

火人碰到火鑽時，火鑽的 is_alive 會變成 false，火鑽就會消失，CInteger 的 hits_left 就會+1

火人碰到藍水與綠水時，CInteger 的 hits_lake 就會-1，hits_lake 預設值為 1，只要小於等於 0 就會跳轉到 GameStateOver。

箱子碰到火人時，就會傳一個 bool 到火人的物件，讓火人無法前進，當火人跳上箱子時，也會傳一個 bool 給火人，讓火人地板設置在箱子上。

冰人亦然。

當火人與冰人同時碰撞到各自通關門時，就會跳轉到 GameStateWin。

(4) GameStateRun2

GameStateRun2 的物件碰撞情形與第一關相同。

(5) GameStateWin

GameStateWin的畫面會根據是否吃到指定數量的鑽石來打勾。

按下Continue後回到GameStateMenu(遊戲選單)

(6) CGameStateOver

按下Retry可以重是當前關卡，按下Menu後回到GameStateMenu(遊戲選單)

2. 程式類別

類別名稱	.h 檔行數	.cpp 檔行數	說明
box	47	226	可推動箱子類別
Button	35	173	可採按鈕類別
GreenLake	25	58	綠水類別
IceDiamond	25	64	冰鑽類別
IceDoor	25	68	冰通關門類別
IceLake	23	53	藍水類別
IcePlayer	69	426	冰人類別
Mood	31	113	拉桿類別
RedDiamond	25	65	火鑽類別
RedDoor	25	70	火通關門類別
RedLake	23	53	紅水類別
RedPlayer	70	439	火人類別
CGameMap	25	86	地圖類別
mygame	239	1199	遊戲各個 GameState 類別
總和	687	3093	

3. 程式技術

本程式使用 GameFramework 的架構,與範例程式的調整、延伸,使用 fstream 與 stringstream 讀取外部地圖檔。

四、 結語

1. 問題與解決方法

(1)多層重力問題，因為地圖有分好幾層，每層的 Floor 都不一樣，導致有穿模問題。

解決:原先使用 if-else 作為，但是第二關無法使用，所以改寫為用 txt 檔作為地圖陣列，動態判定地板位置。

(2)動態物件問題，對於箱子，浮板等動態物件，導致 FLOOR 以及牆壁有所改變。

解決:新增 flag 以增加腳色 OnMove()時的判定。

(3)上升拉桿地板優先度問題，因為我們的地圖是靜態檔案。對於動態改變的地板，會有已經更改地板，但是讀取地圖後又被重製的問題。

解決:增加是否在拉桿上的布林值，再將其加入是否讀取地圖設置地板的判斷。

2. 時間表

週次	組員-劉立傑	組員-黃品宥	說明
1	0	0	Introducing the course
2	3	0	GitTutorial Prepare Resource
3	0	3	Framework pritace
4	3	5	Framework pritace
5	5	5	Prepare Resource

6	3	8	Prepare Resource
7	10	5	GameStateInit
8	10	5	Player and Diamond Class
9	16	8	Player Gravity and velocity
10	3	5	Player eat Diamond
11	3	10	2Lake
12	15	10	天花板 牆壁 地板 分數
13	10	10	DeBug
14	3	12	Gravity and Floor
15	10	12	GameStateOver Debug
16	5	12	Box Gravity
17	40	30	GameStateWin 重寫 MAP
18	60	60	GameStateMenu Bug Fix Button Class Mood Class CGameMap Class Level2
總時數	199	200	

3. 貢獻比例

黃品宥 50%

劉立傑 50%

4. 自我檢核表

週次	項目	完成否	無法完成原因
1	解決 Memory leak	■已完成 □未完成	
2	自定遊戲 Icon	■已完成 □未完成	
3	有 About 畫面	■已完成 □未完成	
4	初始畫面說明按鍵及滑鼠	■已完成 □未完成	
5	之用法與密技	■已完成 □未完成	
6	上傳 setup/apk/source 檔	■已完成 □未完成	
7	setup 檔可正確執行	■已完成 □未完成	
8	報告字型、點數、對齊、行	■已完成 □未完成	
9	距、頁碼等格式正確	■已完成 □未完成	
10	全螢幕啟動－改列加分項目	□已完成 ■未完成	

5. 收穫

劉立傑:

這次的專案，讓我對於物件導向的瞭解更加深刻，對於 framework 的開發，也又所了解，物件與物件之間的動作，狀態如何傳輸，以及對於物件的 OnMove 與 OnShow 的分離更加的熟悉。

黃品宥:

經過了這次的專案,我了解到物件導向程式到底是要怎麼去應用的， 從一開始根本不知道怎麼去使用老師給我們的範例程式,到後來能隨心所欲地使用，更了解到時間管理的能力的重要性，因為一開始沒有很努力的去寫程式，加上後面的邏輯越來越複雜導致進度落差很大，最後還是靠努力的加班加點免強補了回來,這次可說我的實習課程是收穫滿滿學習到了許多知識與教訓。

6. 心得

劉立傑:

這次的 OOPL，我發現我有時間分配的問題，原本以為可以在學期期間完成，但是有許多問題，再加上對自己的寬容，導致要在期末趕工，做出來的成品也有一些問題，第二次 Demo 時，為了趕出進度，重力世界使用 if-else 硬幹出來，效果有出來了，但是再做第二關時，就要砍掉重寫，對於這個情況，我覺得我需要做更好得時間規劃，對於進度的期限應該更加嚴苛，對於小組的應該更有效的溝通，這學期的課程，我學到了更多關於 OOP 的應用與團隊的開發，對我很又幫助。

黃品宥：

在整個製作專案的過程中一開始是信心滿滿的,還問助教說是不是只要把專案完成後就可以不用來上課了，那真是初生之犢不畏虎的想法， 一開始也沒太在意所以花的時間不太多,但到後來才知道了製作遊戲的困難程度,遠遠超過了我們的想像，牽一髮而動全身只要有一個不能解決的 BUG 就有可能要把之前的努力都打掉,換一個想法去寫,像是我們一開始的地圖是用 if-else 寫出來的,但後來發現以這種模式下去寫根本沒辦法去做其他關卡的功能， 且每創出新的一關就要花費很多很多的時間，到後來真的不行了就把所有的地圖改掉,但卻不單單只是改動地圖,因為地圖上牽扯的東西很多所以大多數的物件都要做微調或大修,導致後來出現了許多判斷上的不精準,一度很崩潰有想要放棄的念頭,不過這畢竟是團體賽絕不能輕言放棄,在最後的關頭,熬了無數個夜終於把遊戲給做出來,在做出來的那一瞬間真的感動萬分。

7.　對於本次課程的建議

附錄

```
<mygame.h>
#include "RedDiamond.h"
#include "IceDiamond.h"
#include "IcePlayer.h"
#include "RedPlayer.h"
#include "Button.h"
#include "RedDoor.h"
#include "IceDoor.h"
#include "RedLake.h"
#include "IceLake.h"
#include "Mood.h"
#include "box.h"
#include "Greenlake.h"
#include <fstream>
extern bool current_rank;

namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// Constants
	/////////////////////////////////////////////////////////////////////////////

	enum AUDIO_ID {				// 定義各種音效的編號
		AUDIO_DING,				// 0
		AUDIO_LAKE,				// 1
		AUDIO_NTUT				// 2
	};

	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 為遊戲的遊戲開頭畫面物件
	/////////////////////////////////////////////////////////////////////////////

	class CGameStateInit : public CGameState {
	public:
		CGameStateInit(CGame *g);
		void OnInit();  								// 遊戲的初值及圖形設定
		void OnBeginState();							// 設定每次重玩所需的變數
		void OnKeyUp(UINT, UINT, UINT); 				// 處理鍵盤 Up 的動作
		void OnLButtonDown(UINT nFlags, CPoint point);  // 處理滑鼠的動作
	protected:
		void OnShow();									// 顯示這個狀態的遊戲畫面
	private:
		CMovingBitmap logo,intro,intro2;
		bool intro_bool;
	};

	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 為遊戲第一關的狀態(Game Run 1)
	/////////////////////////////////////////////////////////////////////////////

	class CGameStateRun : public CGameState {
	public:
		CGameStateRun(CGame *g);
		~CGameStateRun();
		CGameMap gamemap;
		void OnBeginState();							// 設定每次重玩所需的變數
		void OnInit();  								// 遊戲的初值及圖形設定
		void OnKeyDown(UINT, UINT, UINT);
		void OnKeyUp(UINT, UINT, UINT);
	protected:
		void OnMove();									// 移動遊戲元素
		void OnShow();							// 顯示這個狀態的遊戲畫面
	private:
```

```
	bool JUMP, UP;

	const int		NUMRED;			// 火鑽的總數
	const int		NUMICE;			// 冰鑽的總數
	const int		LAKERED;		// 紅水的總數
	const int		LAKEICE;		// 藍水的總數
	const int		LAKEGREEN;		// 綠水的總數
	const int		NUMMOD;			// 拉桿的總數
	const int		NUMBUT;			// 按鈕的總數
	CMovingBitmap	background;		// 背景圖

	RedDiamond		*diamond1;
	IceDiamond		*diamond2;
	CInteger		hits_left;
	CInteger		hits_lake;
	CInteger		hits_door;
	RedPlayer		player1;
	IcePlayer		player2;
	RedLake			*Lake1;
	IceLake			*Lake2;
	Greenlake		*Lake3;
	RedDoor			reddoor;
	IceDoor			icedoor;
	Mood			*mood;
	box				box;
	Button			*button;
};

/////////////////////////////////////////////////////////////////////////////
// 這個 class 為遊戲第二關的狀態(Game Run 2)
/////////////////////////////////////////////////////////////////////////////
class CGameStateRun2 : public CGameState {
public:
	CGameStateRun2(CGame *g);
	~CGameStateRun2();
	CGameMap gamemap;
	void OnBeginState();						// 設定每次重玩所需的變數
	void OnInit();								// 遊戲的初值及圖形設定
	void OnKeyDown(UINT, UINT, UINT);
	void OnKeyUp(UINT, UINT, UINT);
protected:
	void OnMove();								/ 移動遊戲元素
	void OnShow();								// 顯示這個狀態的遊戲畫面
private:
	bool JUMP, UP;

	const int		NUMRED;						// 火鑽的總數
	const int		NUMICE;						// 冰鑽的總數
	const int		LAKERED;					// 紅水的總數
	const int		LAKEICE;					// 藍水的總數
	const int		LAKEGREEN;					// 綠水的總數
	const int		NUMMOD;						// 拉桿的總數
	const int		NUMBUT;						// 按鈕的總數
	const int		NUMBUT1;
	CMovingBitmap	background;					// 背景圖

	RedDiamond		*diamond1;
	IceDiamond		*diamond2;
	CInteger		hits_left;
	CInteger		hits_lake;
	CInteger		hits_door;
	RedPlayer		player1;
	IcePlayer		player2;
	RedLake			*Lake1;
	IceLake			*Lake2;
	Greenlake		*Lake3;
	RedDoor			reddoor;
	IceDoor			icedoor;
	Mood			*mood;
```

```
        Button                  *button;
        Button                  *button1;
    };
    /////////////////////////////////////////////////////////////////////////////

    /////////////////////////////////////////////////////////////////////////////
    // 這個 class 為遊戲的結束狀態(Game Over)
    /////////////////////////////////////////////////////////////////////////////

    class CGameStateOver : public CGameState {
    public:
        CGameStateOver(CGame *g);
        void OnLButtonDown(UINT nFlags, CPoint point);
        void OnBeginState();                                    // 設定每次重玩所需的變數
        void OnInit();
    protected:
        void OnMove();                                          // 移動遊戲元素
        void OnShow();                                          // 顯示這個狀態的遊戲畫面
    private:
        int counter;
        CMovingBitmap gameover;
    };
    /////////////////////////////////////////////////////////////////////////////
    // 這個 class 為遊戲贏的狀態(Game Win)
    /////////////////////////////////////////////////////////////////////////////
    class CGameStateWin : public CGameState {
    public:
        CGameStateWin(CGame *g);
        void OnLButtonDown(UINT nFlags, CPoint point);
        void OnBeginState();                                    // 設定每次重玩所需的變數
        void OnInit();
    protected:
        void OnMove();                                          // 移動遊戲元素
        void OnShow();                                          // 顯示這個狀態的遊戲畫面
    private:
        int counter;
        CMovingBitmap gamewinspace;
        CMovingBitmap alarm;
        CMovingBitmap boygirl;
        CMovingBitmap dim;
        CMovingBitmap good;
        CMovingBitmap bad;
        CMovingBitmap next;
        CMovingBitmap gold;
        CMovingBitmap conti;
        CMovingBitmap fin1;
        CMovingBitmap fin2;
        CMovingBitmap fin3;
        CMovingBitmap fin4;
    };
    /////////////////////////////////////////////////////////////////////////////
    // 這個 class 為遊戲的選關狀態(Game Menu)
    /////////////////////////////////////////////////////////////////////////////
    class CGameStateMenu : public CGameState {
    public:
        CGameStateMenu(CGame *g);
        void OnLButtonDown(UINT nFlags, CPoint point);
        void OnBeginState();                                    // 設定每次重玩所需的變數
        void OnInit();
    protected:
        void OnMove();                                          // 移動遊戲元素
        void OnShow();                                          // 顯示這個狀態的遊戲畫面
    private:
        int counter;
        CMovingBitmap bg;
    };
}
```

<mygame.cpp>

```
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "mygame.h"
#include <iostream>         // std::cout
#include <fstream>
#include <string.h>
#include <conio.h>
#include <time.h>
#include <atomic>
#include <chrono>
#include <thread>
#include <stdio.h>
#include <sstream>
namespace game_framework {
static int stage = 0;
static int findim = 0;
//NUMRED,NUMICE, LAKERED, LAKEICE, LAKEGREEN, NUMMOD, NUMBUT
//設定各個關卡各種物件數量
static int count[2][7] = { {3,4,1,1,1,2,3},
                                        {8,8,2,2,2,1,2}};
static bool corr;
//第一關各種物件座標
static int      stage1_diamond1[3][2] = { {405,535},{140,260},{223,41} },
                        stage1_diamond2[4][2] = { {570,535},{470,290},{475,87},{35,109} },
                        stage1_Lake1_position[1][2] = { {360,579} },
                        stage1_Lake2_position[1][2] = { {530,579} },
                        stage1_Lake3_position[1][2] = { {460,455} },
                        stage1_mood_position[2][2] = { {260,380},{20,310} },
                        stage1_button_position[3][2] = { {270,295},{710,230} ,{640,215} };
//第二關各種物件座標
static int      stage2_diamond1[8][2]           =           {           {161,550},{247,550},{484,487}           ,{576,487},
{304,373},{576,373},{376,243} ,{366,67} },
                        stage2_diamond2[8][2]           =           {           {484,550},{576,550},{161,487}           ,{247,487},
{202,373},{484,373},{419,243} ,{409,67} },
                        stage2_Lake1_position[2][2] = { {125,579} ,{454,517} },
                        stage2_Lake2_position[2][2] = { {453,579}, {125,517} },
                        stage2_Lake3_position[2][2] = { {164,310},{474,310} },
                        stage2_mood_position[2][2] = { {260,380} ,{300,300}    },
                        stage2_button_position[3][2] = { {140,390},{400,330},{600,390}},
                        stage2_button_position1[3][2] = { {200,100},{450,110},{560,100} };
/////////////////////////////////////////////////////////////////////////////
// 這個 class 為遊戲的遊戲開頭畫面物件
/////////////////////////////////////////////////////////////////////////////
CGameStateInit::CGameStateInit(CGame *g)
: CGameState(g)
{
        intro_bool = false;
}

void CGameStateInit::OnInit()
{
        ShowInitProgress(0);
        logo.LoadBitmap(IDB_GAME_MENU);
        intro.LoadBitmap(IDB_INTRO);
        intro2.LoadBitmap(INTRO);
}
void CGameStateInit::OnBeginState()
{
}

void CGameStateInit::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        const char KEY_ESC = 27;
        if (nChar == KEY_ESC)
```

```
		PostMessage(AfxGetMainWnd()->m_hWnd, WM_CLOSE, 0, 0);  //  關閉遊戲
}

void CGameStateInit::OnLButtonDown(UINT nFlags, CPoint point)
{
	//點擊 Play 的位置後跳轉到 GameStateMenu
	if ((point.x > 303 && point.x < 485) && (point.y > 311 && point.y < 375))
	{
		GotoGameState(GAME_STATE_MENU);
	}
	//點擊齒輪後顯示 Help
	if ((point.x > 10 && point.x < 82) && (point.y > 475 && point.y < 555))
	{
		intro_bool = true;
	}
	//點擊 Back 後回到主畫面
	if ((point.x > 165 && point.x < 257) && (point.y > 382 && point.y < 421))
	{
		intro_bool = false;
		GotoGameState(GAME_STATE_INIT);
	}
}

void CGameStateInit::OnShow()
{
	logo.SetTopLeft(0, 0);
	logo.ShowBitmap();
	intro2.SetTopLeft(0, 0);
	intro2.ShowBitmap();
	intro.SetTopLeft(118, 135);
	if (intro_bool)
	{
		intro.ShowBitmap();
	}
}
/////////////////////////////////////////////////////////////////////////////
//  這個 class 為遊戲的結束狀態(Game Over)
/////////////////////////////////////////////////////////////////////////////
CGameStateOver::CGameStateOver(CGame *g)
: CGameState(g)
{
}
void CGameStateOver::OnLButtonDown(UINT nFlags, CPoint point)
{
	if ((point.x > 254 && point.x < 367) && (point.y > 337 && point.y < 376))
	{
		GotoGameState(GAME_STATE_MENU);
	}
	if ((point.x > 444 && point.x < 556) && (point.y > 337 && point.y < 376))
	{
		if (stage == 0)
		{
			GotoGameState(GAME_STATE_RUN);
		}

		if (stage == 1)
		{
			GotoGameState(GAME_STATE_RUN2);
		}
	}
}
void CGameStateOver::OnMove()
{

}
void CGameStateOver::OnBeginState()
{

}
```

```
void CGameStateOver::OnInit()
{
        ShowInitProgress(66);
        gameover.LoadBitmap(GAMEOVER, RGB(255, 255, 255));
        ShowInitProgress(100);
}
void CGameStateOver::OnShow()
{
        gameover.SetTopLeft(118, 138);
        gameover.ShowBitmap();
}
/////////////////////////////////////////////////////////////////////////////
// 這個 class 為遊戲的選關狀態(Game Menu)
/////////////////////////////////////////////////////////////////////////////
CGameStateMenu::CGameStateMenu(CGame *g)
        : CGameState(g)
{
}
void CGameStateMenu::OnLButtonDown(UINT nFlags, CPoint point)
{
        if ((point.x > 39 && point.x < 210) && (point.y > 492 && point.y < 588))
        {
                GotoGameState(GAME_STATE_INIT);
        }
        if ((point.x > 378 && point.x < 422) && (point.y > 492 && point.y < 542))
        {
                stage = 0;
                GotoGameState(GAME_STATE_RUN);
        }
        if ((point.x > 409 && point.x < 455) && (point.y > 437 && point.y < 487))
        {
                stage = 1;
                GotoGameState(GAME_STATE_RUN2);
        }
}
void CGameStateMenu::OnMove()
{

}
void CGameStateMenu::OnBeginState()
{

}
void CGameStateMenu::OnInit()
{
        ShowInitProgress(66);
        bg.LoadBitmap(IDB_MENU, RGB(255, 255, 255));
        ShowInitProgress(100);
}
void CGameStateMenu::OnShow()
{
        bg.SetTopLeft(0, 0);
        bg.ShowBitmap();
}
/////////////////////////////////////////////////////////////////////////////
// 這個 class 為遊戲的贏的狀態(Game Win)
/////////////////////////////////////////////////////////////////////////////
CGameStateWin::CGameStateWin(CGame *g)
        : CGameState(g)
{
}
void CGameStateWin::OnLButtonDown(UINT nFlags, CPoint point)
{
        if ((point.x > 320 && point.x < 510) && (point.y > 350 && point.y < 404))
        {
                GotoGameState(GAME_STATE_MENU);
        }
}
void CGameStateWin::OnMove()
```

```
{
	if (stage == 0) {
		if (findim == 7) {
			corr = true;
		}
		else {
			corr = false;
		}
	}
	else {
		if (findim == 16) {
			bool corr = true;
		}
		else {
			bool corr = false;
		}
	}
}
void CGameStateWin::OnBeginState()
{

}
void CGameStateWin::OnInit()
{
	ShowInitProgress(66);				// 接個前一個狀態的進度，此處進度視為 66%
	fin1.LoadBitmap(GOOD, RGB(255, 255, 255));
	fin3.LoadBitmap(BAD, RGB(255, 255, 255));
	fin2.LoadBitmap(GOOD, RGB(255, 255, 255));
	fin4.LoadBitmap(BAD, RGB(255, 255, 255));
	gamewinspace.LoadBitmap(Win, RGB(255, 255, 255));
	alarm.LoadBitmap(Alarm, RGB(255, 255, 255));
	boygirl.LoadBitmap(BOYGIRL, RGB(255, 255, 255));
	dim.LoadBitmap(DIM, RGB(255, 255, 255));
	good.LoadBitmap(GOOD, RGB(255, 255, 255));
	bad.LoadBitmap(BAD, RGB(255, 255, 255));
	conti.LoadBitmap(CONTI, RGB(255, 255, 255));
	next.LoadBitmap(NEXT, RGB(255, 255, 255));
	gold.LoadBitmap(GOLD, RGB(255, 255, 255));
	ShowInitProgress(100);
}
void CGameStateWin::OnShow()
{
	gamewinspace.SetTopLeft(130, 138);
	gamewinspace.ShowBitmap();
	alarm.SetTopLeft(270, 280);
	alarm.ShowBitmap();
	boygirl.SetTopLeft(270, 180);
	boygirl.ShowBitmap();
	dim.SetTopLeft(270, 240);
	dim.ShowBitmap();
	next.SetTopLeft(380, 231);
	next.ShowBitmap();
	gold.SetTopLeft(480, 231);
	gold.ShowBitmap();
	conti.SetTopLeft(320, 350);
	conti.ShowBitmap();
	fin1.SetTopLeft(320, 180);
	fin1.ShowBitmap();
	fin3.SetTopLeft(320, 280);
	fin3.ShowBitmap();
	if (corr) {
		fin2.SetTopLeft(320, 230);
		fin2.ShowBitmap();
	}
	else {
		fin4.SetTopLeft(320, 230);
		fin4.ShowBitmap();
	}
}
```

```
/////////////////////////////////////////////////////////////////////////////
// 第一個關卡的 GameState
/////////////////////////////////////////////////////////////////////////////
CGameStateRun::CGameStateRun(CGame *g)
: CGameState(g), NUMRED(3),NUMICE(4), LAKERED(1), LAKEICE(1), LAKEGREEN(1), NUMMOD(2), NUMBUT(3)
{
	diamond1 = new RedDiamond[NUMRED];
	diamond2 = new IceDiamond[NUMICE];
	Lake1 = new RedLake[LAKERED];
	Lake2 = new IceLake[LAKEICE];
	Lake3 = new Greenlake[LAKEGREEN];
	mood = new Mood[NUMMOD];
	button = new Button[NUMBUT];
}

CGameStateRun::~CGameStateRun()
{
	if (diamond1 != NULL)
		delete[] diamond1;
	if (diamond2 != NULL)
		delete[] diamond2;
	if (Lake1 != NULL)
		delete[] Lake1;
	if (Lake2 != NULL)
		delete[] Lake2;
	if (Lake3 != NULL)
		delete[] Lake3;
	if (mood != NULL)
		delete[] mood;
	if (button != NULL)
		delete[] button;
}

void CGameStateRun::OnBeginState()
{
	findim = 0;
	const int HITS_LEFT = 0;
	const int HITS_LAKE = 1;
	const int HITS_DOOR = 1;
	const int HITS_LEFT_X = 590;
	const int HITS_LEFT_Y = 0;
	//載入地圖
	gamemap.ReadFile(stage+1);
	//設置各種物件位置地圖
	for (int i = 0; i < NUMRED; i++) {
		diamond1[i].SetXY(stage1_diamond1[i][0], stage1_diamond1[i][1]);
		diamond1[i].SetIsAlive(true);
	}
	for (int i = 0; i < NUMICE; i++) {
		diamond2[i].SetXY(stage1_diamond2[i][0], stage1_diamond2[i][1]);
		diamond2[i].SetIsAlive(true);
	}
	for (int i = 0; i < LAKERED; i++) {
		Lake1[i].SetXY(stage1_Lake1_position[i][0], stage1_Lake1_position[i][1]);
	}
	for (int i = 0; i < LAKEICE; i++) {
		Lake2[i].SetXY(stage1_Lake2_position[i][0], stage1_Lake2_position[i][1]);
	}
	for (int i = 0; i < LAKEICE; i++) {
		Lake3[i].SetXY(stage1_Lake3_position[i][0], stage1_Lake3_position[i][1]);
	}
	for (int i = 0; i < NUMMOD; i++) {
		mood[i].SetXY(stage1_mood_position[i][0], stage1_mood_position[i][1]);
		mood[i].SetIsAlive(true);
	}
	for (int i = 0; i < NUMBUT; i++) {
		button[i].SetXY(stage1_button_position[i][0], stage1_button_position[i][1]);
		button[i].SetIsAlive(true);
	}
```

```
		player1.Initialize(stage+1);
		player2.Initialize(stage+1);
		player1.SetXY(42, 542);
		player2.SetXY(42, 462);
		player2.SetXY(42, 462);
		reddoor.SetIsAlive(true);
		reddoor.SetXY(690, 60);
		icedoor.SetIsAlive(true);
		icedoor.SetXY(600, 60);
		box.init();
		box.SetXY(500, 160);
		background.SetTopLeft(0,0);
		hits_left.SetInteger(HITS_LEFT);
		hits_lake.SetInteger(HITS_LAKE);
		hits_door.SetInteger(HITS_DOOR);
		hits_left.SetTopLeft(HITS_LEFT_X,HITS_LEFT_Y);
		CAudio::Instance()->Play(AUDIO_LAKE, true);
		CAudio::Instance()->Play(AUDIO_DING, false);
		CAudio::Instance()->Play(AUDIO_NTUT, true);
	}
	void CGameStateRun::OnMove()
	{
		int i;
		for (i = 0; i < NUMRED; i++)
			diamond1[i].OnMove();
		for (i = 0; i < NUMICE; i++)
		{
			diamond2[i].OnMove();
		}
		mood[0].OnMove();
		mood[1].OnMove1();
		button[0].OnMove();
		button[1].OnMove1();
		player1.MoodY(button[1].GetY());
		player2.MoodY(button[1].GetY());
		player1.OnMove();
		player2.OnMove();
		player1.OnMove1();
		player2.OnMove1();
		button[2].OnMove();
		reddoor.OnMove();
		icedoor.OnMove();
		box.OnMove();
		for (i = 0; i < NUMRED; i++) {
			if (diamond1[i].IsAlive() && diamond1[i].HitPlayer(&player1)) {
				diamond1[i].SetIsAlive(false);
				CAudio::Instance()->Play(AUDIO_DING);
				hits_left.Add(1);
				findim = findim + 1;
			}
		}
		for (i = 0; i < NUMICE; i++)
		{
			if (diamond2[i].IsAlive() && diamond2[i].HitPlayer(&player2)) {
				diamond2[i].SetIsAlive(false);
				CAudio::Instance()->Play(AUDIO_DING);
				hits_left.Add(1);
				findim = findim + 1;
			}
		}
		for (i = 0; i < LAKERED; i++) {
			if ( Lake1[i].HitPlayer(&player2) && Lake1[i].hack == false) {
				CAudio::Instance()->Play(AUDIO_DING);
				hits_lake.Add(-1);

				if (hits_lake.GetInteger() <= 0) {
					CAudio::Instance()->Stop(AUDIO_LAKE);
					CAudio::Instance()->Stop(AUDIO_NTUT);
					GotoGameState(GAME_STATE_OVER);
				}
			}
		}
		for (i = 0; i < LAKEICE; i++)
		{
			if ( Lake2[i].HitPlayer(&player1)&& Lake2[i].hack==false) {
				CAudio::Instance()->Play(AUDIO_DING);
				hits_lake.Add(-1);
				if (hits_lake.GetInteger() <= 0) {
					CAudio::Instance()->Stop(AUDIO_LAKE);
					CAudio::Instance()->Stop(AUDIO_NTUT);
					GotoGameState(GAME_STATE_OVER);
				}
			}
		}
		for (i = 0; i < LAKEGREEN; i++)
		{
			if ((Lake3[i].HitPlayer(&player1)|| Lake3[i].HitPlayer(&player2)) && Lake3[i].hack == false) {
				CAudio::Instance()->Play(AUDIO_DING);
				hits_lake.Add(-1);
				if (hits_lake.GetInteger() <= 0) {
					CAudio::Instance()->Stop(AUDIO_LAKE);
					CAudio::Instance()->Stop(AUDIO_NTUT);
					GotoGameState(GAME_STATE_OVER);
				}
			}
		}
		if (box.HitEraser(&player1)||box.HitEraser(&player2)) {
			box.SetMovingLeft(true);
		}

		player1.setFront(player1.frontBox(box.GetX1(), box.GetY1()));
		player2.setFront(player2.frontBox(box.GetX1(), box.GetY1()));
		player1.setOnBox(player1.onBox(box.GetX1(), box.GetY1()));
		player2.setOnBox(player2.onBox(box.GetX1(), box.GetY1()));

		if (reddoor.IsAlive() && (reddoor.HitPlayer(&player1))) {
			reddoor.SetIsAlive(false);
		}
		if (!(reddoor.IsAlive()) && !(reddoor.HitPlayer(&player1))) {
			reddoor.SetIsAlive(true);
		}
		if (icedoor.IsAlive() && icedoor.HitPlayer(&player2)) {
			icedoor.SetIsAlive(false);
		}
		if (!(icedoor.IsAlive()) && !(icedoor.HitPlayer(&player2))) {
			icedoor.SetIsAlive(true);
		}
		if ((mood[0].IsAlive()) && (mood[0].HitPlayer(&player2))) {
			mood[0].SetIsAlive(false);
			mood[1].SetIsAlive(false);
			player1.SetMood(true);
			player2.SetMood(true);
		}
		if ((mood[0].IsAlive()) && (mood[0].HitPlayer(&player1))) {
			mood[0].SetIsAlive(false);
			mood[1].SetIsAlive(false);
			player1.SetMood(true);
			player2.SetMood(true);
		}
		if (button[0].IsAlive() && ((button[0].HitPlayer(&player2)) || button[0].HitPlayer(&player1))) {
			button[0].SetIsAlive(false);
			button[1].SetIsAlive(false);
			player1.SetButton(true);
			player2.SetButton(true);
		}
		if (button[2].IsAlive() && ((button[2].HitPlayer(&player2)) || button[2].HitPlayer(&player1))) {
			button[2].SetIsAlive(false);
			button[1].SetIsAlive(false);
			player1.SetButton(true);
			player2.SetButton(true);
		}
		if (!((button[0].IsAlive()) && (button[2].IsAlive())) &&\
			!(button[0].HitPlayer(&player1)) && !(button[0].HitPlayer(&player2)) &&\
			!(button[2].HitPlayer(&player1)) && !(button[2].HitPlayer(&player2)))
		{
			button[1].SetIsAlive(true);
			player1.SetButton(false);
			player2.SetButton(false);
		}
		if (!(button[0].IsAlive()) && !(button[0].HitPlayer(&player1)) && !(button[0].HitPlayer(&player2))) {
			button[0].SetIsAlive(true);
		}
		if (!(button[2].IsAlive()) && !(button[2].HitPlayer(&player1)) && !(button[2].HitPlayer(&player2))) {
			button[2].SetIsAlive(true);
		}

		if (player1.butin())
		{
			if (player1.GetY2() <= button[1].GetY()-1)
			{
				player1.isOnButton = true;
				int tmp_y = button[1].GetY();
				player1.SetFloor(tmp_y-2);
				player1.OnMove1();
			}
		}
		else
		{
			player1.isOnButton = false;
		}
		if (player2.butin())
		{
			if (player2.GetY2() <= button[1].GetY() - 1)
			{
				player2.isOnButton = true;
				int tmp_y = button[1].GetY();
				player2.SetFloor(tmp_y - 2);
				player2.OnMove1();
			}
		}
		else
		{
			player2.isOnButton = false;
		}
		if (!(icedoor.IsAlive()) && !(reddoor.IsAlive())) {
			GotoGameState(GAME_STATE_WIN);
		}
	}

	void CGameStateRun::OnInit()
	{
		ShowInitProgress(33);
		int i;
		for (i = 0; i < NUMRED; i++)
			diamond1[i].LoadBitmap();
		for (i = 0; i < NUMICE; i++) {
			diamond2[i].LoadBitmap();
		}
		for (i = 0; i < LAKERED; i++)
			Lake1[i].LoadBitmap();
		for (i = 0; i < LAKEICE; i++) {
			Lake2[i].LoadBitmap();
		}
		for (i = 0; i < LAKEGREEN; i++) {
			Lake3[i].LoadBitmap();
		}
		gamemap.LoadBitmap();
	player1.LoadBitmap();
	player2.LoadBitmap();
	reddoor.LoadBitmap();
	icedoor.LoadBitmap();
	for (i = 0; i < NUMMOD; i++) {
		mood[i].LoadBitmap();
	}
	for (i = 0; i < NUMBUT; i++) {
		button[i].LoadBitmap();
	}
	box.LoadBitmap();
	background.LoadBitmap(IDB_MAP1);

	ShowInitProgress(50);
	Sleep(300);
	hits_left.LoadBitmap();

	CAudio::Instance()->Load(AUDIO_DING,    ".\\sounds\\ding.wav");
	CAudio::Instance()->Load(AUDIO_LAKE,    ".\\sounds\\lake.mp3");
	CAudio::Instance()->Load(AUDIO_NTUT,    ".\\sounds\\LevelMusic.mp3");

}


void CGameStateRun::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	const char KEY_LEFT    = 0x25;
	const char KEY_UP      = 0x26;
	const char KEY_RIGHT = 0x27;
	const char KEY_DOWN    = 0x28;
	const char KEY_A = 'A';
	const char KEY_W = 'W';
	const char KEY_D = 'D';
	const char KEY_S = 'S';
	const char KEY_P = 'P';

	if (nChar == KEY_LEFT)
	{
		player1.SetMovingLeft(true);
	}
	if (nChar == KEY_RIGHT) {

		player1.SetMovingRight(true);
	}
	if (nChar == KEY_UP)
	{
		player1.SetMovingUp(true);
	}
	if (nChar == KEY_DOWN)
	{
		player1.SetMovingDown(true);
	}
	if (nChar == KEY_A)
	{
		player2.SetMovingLeft(true);
	}
	if (nChar == KEY_D) {

		player2.SetMovingRight(true);
	}
	if (nChar == KEY_W)
	{

		player2.SetMovingUp(true);
	}
	if (nChar == KEY_S)
	{
		player2.SetMovingDown(true);
	}
```

```
        if (nChar == KEY_P)
        {
                for (int i = 0; i < LAKERED; i++) {
                        Lake1[i].hack = true;
                }
                for (int i = 0; i < LAKEICE; i++) {
                        Lake2[i].hack= true;
                }
                for (int i = 0; i < LAKEGREEN; i++) {
                        Lake3[i].hack = true;
                }
                hits_left.Add(10000);
        }
}

void CGameStateRun::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
        const char KEY_LEFT    = 0x25; // keyboard 左箭頭
        const char KEY_UP        = 0x26; // keyboard 上箭頭
        const char KEY_RIGHT = 0x27; // keyboard 右箭頭
        const char KEY_DOWN    = 0x28; // keyboard 下箭頭
        const char KEY_A = 'A';
        const char KEY_W = 'W';
        const char KEY_D = 'D';
        const char KEY_S = 'S';
        if (nChar == KEY_LEFT)
        {
                player1.SetMovingLeft(false);
        }
        if (nChar == KEY_RIGHT) {

                player1.SetMovingRight(false);
        }
        if (nChar == KEY_UP)
        {
                player1.SetMovingUp(false);
        }
        if (nChar == KEY_DOWN)
        {
                player1.SetMovingDown(false);
        }
        if (nChar == KEY_A)
        {
                player2.SetMovingLeft(false);

        }
        if (nChar == KEY_D)
        {
                player2.SetMovingRight(false);
        }
        if (nChar == KEY_W)
        {
                player2.SetMovingUp(false);
        }
        if (nChar == KEY_S)
        {
                player2.SetMovingDown(false);
        }
}

void CGameStateRun::OnShow()
{
        background.ShowBitmap();
        gamemap.OnShow();
        hits_left.ShowBitmap();
        for (int i=0; i < NUMRED; i++)
        {
                diamond1[i].OnShow();
        }
```

```
	for (int i = 0; i < NUMICE; i++)
	{
		diamond2[i].OnShow();
	}
	reddoor.OnShow();
	icedoor.OnShow();
	mood[0].OnShow();
	mood[1].OnShow1();
	button[0].OnShow();
	button[1].OnShow1();
	button[2].OnShow();
	box.OnShow();
	for (int i = 0; i < LAKERED; i++)
	{
		Lake1[i].OnShow();
	}
	for (int i = 0; i < LAKEICE; i++)
	{
		Lake2[i].OnShow();
	}
	for (int i = 0; i < LAKEGREEN; i++)
	{
		Lake3[i].OnShow();
	}
	player1.OnShow();
	player2.OnShow();
}
/////////////////////////////////////////////////////////////////////////////
// 第二個關卡的 GameState
/////////////////////////////////////////////////////////////////////////////
CGameStateRun2::CGameStateRun2(CGame *g)
	: CGameState(g), NUMRED(8), NUMICE(8), LAKERED(2), LAKEICE(2), LAKEGREEN(2), NUMMOD(0),
NUMBUT(3), NUMBUT1(3)
{
	diamond1 = new RedDiamond[NUMRED];
	diamond2 = new IceDiamond[NUMICE];
	Lake1 = new RedLake[LAKERED];
	Lake2 = new IceLake[LAKEICE];
	Lake3 = new Greenlake[LAKEGREEN];
	mood = new Mood[NUMMOD];
	button = new Button[NUMBUT];
	button1 = new Button[NUMBUT1];
}

CGameStateRun2::~CGameStateRun2()
{
	if (diamond1 != NULL)
		delete[] diamond1;
	if (diamond2 != NULL)
		delete[] diamond2;
	if (Lake1 != NULL)
		delete[] Lake1;
	if (Lake2 != NULL)
		delete[] Lake2;
	if (Lake3 != NULL)
		delete[] Lake3;
	if (mood != NULL)
		delete[] mood;
	if (button != NULL)
		delete[] button;
	if (button1 != NULL)
		delete[] button1;
}

void CGameStateRun2::OnBeginState()
{
	const int HITS_LAKE = 1;
	const int HITS_DOOR = 1;
	const int HITS_LEFT_X = 590;
```

```
		const int HITS_LEFT_Y = 0;
		findim = 0;
		gamemap.ReadFile(stage+1);
		for (int i = 0; i < NUMRED; i++) {
			diamond1[i].SetXY(stage2_diamond1[i][0], stage2_diamond1[i][1]);
			diamond1[i].SetIsAlive(true);
		}
		for (int i = 0; i < NUMICE; i++) {
			diamond2[i].SetXY(stage2_diamond2[i][0], stage2_diamond2[i][1]);
			diamond2[i].SetIsAlive(true);
		}
		for (int i = 0; i < LAKERED; i++) {
			Lake1[i].SetXY(stage2_Lake1_position[i][0], stage2_Lake1_position[i][1]);
		}
		for (int i = 0; i < LAKEICE; i++) {
			Lake2[i].SetXY(stage2_Lake2_position[i][0], stage2_Lake2_position[i][1]);
		}
		for (int i = 0; i < LAKEICE; i++) {
			Lake3[i].SetXY(stage2_Lake3_position[i][0], stage2_Lake3_position[i][1]);
		}
		for (int i = 0; i < NUMMOD; i++) {
			mood[i].SetXY(stage2_mood_position[i][0], stage2_mood_position[i][1]);
			mood[i].SetIsAlive(true);
		}
		for (int i = 0; i < NUMBUT; i++) {
			button[i].SetXY(stage2_button_position[i][0], stage2_button_position[i][1]);
			button[i].SetIsAlive(true);
		}
		for (int i = 0; i < NUMBUT1; i++) {
			button1[i].SetXY(stage2_button_position1[i][0], stage2_button_position1[i][1]);
			button1[i].SetIsAlive(true);
		}
		player1.Initialize(stage+1);
		player2.Initialize(stage+1);
		player1.SetXY(38, 542);
		player2.SetXY(80, 542);
		reddoor.SetIsAlive(true);
		reddoor.SetXY(34, 52);
		icedoor.SetIsAlive(true);
		icedoor.SetXY(106, 52);

		background.SetTopLeft(0, 0);

		hits_lake.SetInteger(HITS_LAKE);
		hits_door.SetInteger(HITS_DOOR);
		hits_left.SetTopLeft(HITS_LEFT_X, HITS_LEFT_Y);
		CAudio::Instance()->Play(AUDIO_LAKE, true);
		CAudio::Instance()->Play(AUDIO_DING, false);
		CAudio::Instance()->Play(AUDIO_NTUT, true);
	}
	void CGameStateRun2::OnMove()
	{
		int i;
		for (i = 0; i < NUMRED; i++)
			diamond1[i].OnMove();
		for (i = 0; i < NUMICE; i++)
		{
			diamond2[i].OnMove();
		}
		mood[0].OnMove();
		mood[1].OnMove1();
		button[0].OnMove();
		button[1].OnMove2();
		button[2].OnMove();
		button1[0].OnMove();
		button1[1].OnMove3();
		button1[2].OnMove();
		player1.MoodY(button[1].GetY());
		player2.MoodY(button[1].GetY());
```

```
player1.OnMove();
player2.OnMove();
reddoor.OnMove();
icedoor.OnMove();

for (i = 0; i < NUMRED; i++) {
        if (diamond1[i].IsAlive() && diamond1[i].HitPlayer(&player1)) {
                diamond1[i].SetIsAlive(false);
                CAudio::Instance()->Play(AUDIO_DING);
                hits_left.Add(1);
                findim = findim + 1;
        }
}
for (i = 0; i < NUMICE; i++)
{
        if (diamond2[i].IsAlive() && diamond2[i].HitPlayer(&player2)) {
                diamond2[i].SetIsAlive(false);
                CAudio::Instance()->Play(AUDIO_DING);
                hits_left.Add(1);
                findim = findim + 1;
        }
}
for (i = 0; i < LAKERED; i++) {
        if (Lake1[i].HitPlayer(&player2) && Lake1[i].hack == false) {
                CAudio::Instance()->Play(AUDIO_DING);
                hits_lake.Add(-1);
                if (hits_lake.GetInteger() <= 0) {
                        CAudio::Instance()->Stop(AUDIO_LAKE);
                        CAudio::Instance()->Stop(AUDIO_NTUT);
                        GotoGameState(GAME_STATE_OVER);
                }
        }
}
for (i = 0; i < LAKEICE; i++)
{
        if (Lake2[i].HitPlayer(&player1) && Lake2[i].hack == false) {
                CAudio::Instance()->Play(AUDIO_DING);
                hits_lake.Add(-1);
                if (hits_lake.GetInteger() <= 0) {
                        CAudio::Instance()->Stop(AUDIO_LAKE);
                        CAudio::Instance()->Stop(AUDIO_NTUT);
                        GotoGameState(GAME_STATE_OVER);
                }
        }
}
for (i = 0; i < LAKEGREEN; i++)
{
        if ((Lake3[i].HitPlayer(&player1) || Lake3[i].HitPlayer(&player2)) && Lake3[i].hack == false) {
                CAudio::Instance()->Play(AUDIO_DING);
                hits_lake.Add(-1);
                if (hits_lake.GetInteger() <= 0) {
                        CAudio::Instance()->Stop(AUDIO_LAKE);
                        CAudio::Instance()->Stop(AUDIO_NTUT);
                        GotoGameState(GAME_STATE_OVER);
                }
        }
}

if (reddoor.IsAlive() && (reddoor.HitPlayer(&player1))) {
        reddoor.SetIsAlive(false);
}
if (!(reddoor.IsAlive()) && !(reddoor.HitPlayer(&player1))) {
        reddoor.SetIsAlive(true);
}
if (icedoor.IsAlive() && icedoor.HitPlayer(&player2)) {
        icedoor.SetIsAlive(false);
}
if (!(icedoor.IsAlive()) && !(icedoor.HitPlayer(&player2))) {
        icedoor.SetIsAlive(true);
}
if ((mood[0].IsAlive()) && (mood[0].HitPlayer(&player2))) {
        mood[0].SetIsAlive(false);
        mood[1].SetIsAlive(false);
        player1.SetMood(true);
        player2.SetMood(true);
}
if ((mood[0].IsAlive()) && (mood[0].HitPlayer(&player1))) {
        mood[0].SetIsAlive(false);
        mood[1].SetIsAlive(false);
        player1.SetMood(true);
        player2.SetMood(true);
}

if (button[0].IsAlive() && ((button[0].HitPlayer(&player2)) || button[0].HitPlayer(&player1))) {
        button[0].SetIsAlive(false);
        button[1].SetIsAlive(false);
        player1.SetButton3(true);
        player2.SetButton3(true);
}
if (button[2].IsAlive() && ((button[2].HitPlayer(&player2)) || button[2].HitPlayer(&player1))) {
        button[2].SetIsAlive(false);
        button[1].SetIsAlive(false);
        player1.SetButton3(true);
        player2.SetButton3(true);
}
if (!((button[0].IsAlive()) && (button[2].IsAlive())) &&\
        !(button[0].HitPlayer(&player1)) && !(button[0].HitPlayer(&player2)) &&\
        !(button[2].HitPlayer(&player1)) && !(button[2].HitPlayer(&player2))) {
        button[1].SetIsAlive(true);
        player1.SetButton3(false);
        player2.SetButton3(false);
}
if (!(button[0].IsAlive()) && !(button[0].HitPlayer(&player1)) && !(button[0].HitPlayer(&player2))) {
        button[0].SetIsAlive(true);
}
if (!(button[2].IsAlive()) && !(button[2].HitPlayer(&player1)) && !(button[2].HitPlayer(&player2))) {
        button[2].SetIsAlive(true);
}

if (button1[0].IsAlive() && ((button1[0].HitPlayer(&player2)) || button1[0].HitPlayer(&player1))) {
        button1[0].SetIsAlive(false);
        button1[1].SetIsAlive(false);
        player1.SetButton2(true);
        player2.SetButton2(true);
}
if (button1[2].IsAlive() && ((button1[2].HitPlayer(&player2)) || button1[2].HitPlayer(&player1))) {
        button1[2].SetIsAlive(false);
        button1[1].SetIsAlive(false);
        player1.SetButton2(true);
        player2.SetButton2(true);
}
if (!((button1[0].IsAlive()) && (button1[2].IsAlive())) &&\
        !(button1[0].HitPlayer(&player1)) && !(button1[0].HitPlayer(&player2)) &&\
        !(button1[2].HitPlayer(&player1)) && !(button1[2].HitPlayer(&player2))) {
        button1[1].SetIsAlive(true);
        player1.SetButton2(false);
        player2.SetButton2(false);
}
if (!(button1[0].IsAlive()) && !(button1[0].HitPlayer(&player1)) && !(button1[0].HitPlayer(&player2))) {
        button1[0].SetIsAlive(true);
}
if (!(button1[2].IsAlive()) && !(button1[2].HitPlayer(&player1)) && !(button1[2].HitPlayer(&player2))) {
        button1[2].SetIsAlive(true);
}

if (!(icedoor.IsAlive()) && !(reddoor.IsAlive())) {
        GotoGameState(GAME_STATE_WIN);
}
```

```
}

void CGameStateRun2::OnInit()
{

	ShowInitProgress(33);
	int i;
	for (i = 0; i < NUMRED; i++)
		diamond1[i].LoadBitmap();
	for (i = 0; i < NUMICE; i++) {
		diamond2[i].LoadBitmap();
	}
	for (i = 0; i < LAKERED; i++)
		Lake1[i].LoadBitmap();
	for (i = 0; i < LAKEICE; i++) {
		Lake2[i].LoadBitmap();
	}
	for (i = 0; i < LAKEGREEN; i++) {
		Lake3[i].LoadBitmap();
	}
	gamemap.LoadBitmap();
	player1.LoadBitmap();
	player2.LoadBitmap();
	reddoor.LoadBitmap();
	icedoor.LoadBitmap();
	for (i = 0; i < NUMMOD; i++) {
		mood[i].LoadBitmap();
	}
	for (i = 0; i < NUMBUT; i++) {
		button[i].LoadBitmap();
	}
	for (i = 0; i < NUMBUT1; i++) {
		button1[i].LoadBitmap();
	}
	background.LoadBitmap(IDB_MAP1);

	ShowInitProgress(50);
	Sleep(300);
}


void CGameStateRun2::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	const char KEY_LEFT = 0x25; // keyboard 左箭頭
	const char KEY_UP = 0x26; // keyboard 上箭頭
	const char KEY_RIGHT = 0x27; // keyboard 右箭頭
	const char KEY_DOWN = 0x28; // keyboard 下箭頭
	const char KEY_A = 'A';
	const char KEY_W = 'W';
	const char KEY_D = 'D';
	const char KEY_S = 'S';
	const char KEY_P = 'P';

	if (nChar == KEY_LEFT)
	{
		player1.SetMovingLeft(true);
	}
	if (nChar == KEY_RIGHT)
	{
		player1.SetMovingRight(true);
	}
	if (nChar == KEY_UP)
	{
		player1.SetMovingUp(true);
	}
	if (nChar == KEY_DOWN)
	{
		player1.SetMovingDown(true);
	}
	if (nChar == KEY_A)
	{
		player2.SetMovingLeft(true);
	}
	if (nChar == KEY_D) {

		player2.SetMovingRight(true);
	}
	if (nChar == KEY_W)
	{
		player2.SetMovingUp(true);
	}
	if (nChar == KEY_S)
	{
		player2.SetMovingDown(true);
	}
	if (nChar == KEY_P)
	{
		for (int i = 0; i < LAKERED; i++) {
			Lake1[i].hack = true;
		}
		for (int i = 0; i < LAKEICE; i++) {
			Lake2[i].hack = true;
		}
		for (int i = 0; i < LAKEGREEN; i++) {
			Lake3[i].hack = true;
		}
		hits_left.Add(10000);
	}
}

void CGameStateRun2::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	const char KEY_LEFT = 0x25; // keyboard 左箭頭
	const char KEY_UP = 0x26; // keyboard 上箭頭
	const char KEY_RIGHT = 0x27; // keyboard 右箭頭
	const char KEY_DOWN = 0x28; // keyboard 下箭頭
	const char KEY_A = 'A';
	const char KEY_W = 'W';
	const char KEY_D = 'D';
	const char KEY_S = 'S';
	if (nChar == KEY_LEFT)
	{
		player1.SetMovingLeft(false);
	}
	if (nChar == KEY_RIGHT)
	{
		player1.SetMovingRight(false);
	}
	if (nChar == KEY_UP)
	{
		player1.SetMovingUp(false);
	}
	if (nChar == KEY_DOWN)
	{
		player1.SetMovingDown(false);
	}
	if (nChar == KEY_A)
	{
		player2.SetMovingLeft(false);
	}
	if (nChar == KEY_D)
	{
		player2.SetMovingRight(false);
	}
	if (nChar == KEY_W)
	{
		player2.SetMovingUp(false);
	}
```

```
		if (nChar == KEY_S)
		{
			player2.SetMovingDown(false);
		}
	}

	void CGameStateRun2::OnShow()
	{
		background.ShowBitmap();
		button1[1].OnShow3();
		gamemap.OnShow();

		for (int i = 0; i < NUMRED; i++)
			diamond1[i].OnShow();
		for (int i = 0; i < NUMICE; i++)
		{
			diamond2[i].OnShow();
		}

		reddoor.OnShow();
		icedoor.OnShow();
		for (int i = 0; i < NUMMOD; i++)
		{
			mood[i].OnShow();
		}
		button[0].OnShow();
		button[1].OnShow2();
		button[2].OnShow();
		button1[0].OnShow();
		button1[2].OnShow();
		for (int i = 0; i < LAKERED; i++)
		{
			Lake1[i].OnShow();
		}
		for (int i = 0; i < LAKEICE; i++)
		{
			Lake2[i].OnShow();
		}
		for (int i = 0; i < LAKEGREEN; i++)
		{
			Lake3[i].OnShow();
		}
		player1.OnShow();
		player2.OnShow();
	}
}

<box.h>
#include "IcePlayer.h"
#include "RedPlayer.h"
#include <fstream>
extern bool current_rank;
namespace game_framework {
/////////////////////////////////////////////////////////////////////////////
// 這個 class 是箱子的 Class
/////////////////////////////////////////////////////////////////////////////
	class box
	{
	public:
		box();
		void init();
		int x, y;
		bool HitEraser(RedPlayer *redplayer);
		bool HitEraser(IcePlayer *iceplayer);
		void LoadBitmap();
		void OnMove();
		void SetMovingLeft(bool flag);
		void OnShow();
		void SetXY(int nx, int ny);
```

```
            void SetIsAlive(bool alive);
            bool isLeftRightEmpty(int x, int y, int value);
            void setfloor();
            bool OnBox(int tx1, int ty1, int tx2, int ty2);
            int GetX1();
            int GetY1();
        protected:
            CMovingBitmap bmp;
            CGameMap gamemap;
            bool isMovingDown;
            bool isMovingLeft;
            bool isMovingRight;
            bool isMovingUp;
            int dx, dy;
            int floor;
            int map[60][80];
            int x_edge[800];
            int y_edge[600];
            bool is_alive;
        private:
            bool HitRectangle(int tx1, int ty1, int tx2, int ty2);
            bool inertia;                                                   //慣性
            int initial_velocity;
            int velocity;
        };
}

<box.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "box.h"
#include <iostream>
#include <fstream>
#include <string.h>
#include <conio.h>
#include <time.h>
#include <atomic>
#include <chrono>
#include <thread>
#include <stdio.h>
#include <sstream>
namespace game_framework {
        box::box()
        {
            init();
        }
        //設置初始值，讀取地圖檔案轉換成陣列
        void box::init()
        {
            gamemap.~CGameMap();
            gamemap.ReadFile(1);
            const int INITIAL_VELOCITY = 6;
            x = y =    0;
            dx = dy = 35;
            floor = 190;
            initial_velocity = INITIAL_VELOCITY;
            velocity = initial_velocity;
            inertia = false;
            isMovingLeft = false;
            for (int i = 0; i < 60; i++)
            {
                for (int j = 0; j < 80; j++)
                {
                    map[i][j] = gamemap.mapCoordinate(j, i);
                }
```

```
	}
	for (int j = 0; j < 800; j++)
	{
		x_edge[j] = j + 1;
	}
	for (int j = 0; j < 600; j++)
	{
		y_edge[j] = j + 1;
	}
}
bool box::HitEraser(RedPlayer *redplayer)
{

	return HitRectangle(redplayer->GetX1(), redplayer->GetY1(),
		redplayer->GetX2(), redplayer->GetY2());
}
bool box::HitEraser(IcePlayer *iceplayer)
{

	return HitRectangle(iceplayer->GetX1(), iceplayer->GetY1(),
		iceplayer->GetX2(), iceplayer->GetY2());
}

bool box::HitRectangle(int tx1, int ty1, int tx2, int ty2)
{
	int x1 = x;				// 玩家的左上角 x 座標
	int y1 = y;				// 玩家的左上角 y 座標
	int x2 = x1 + 45;	// 玩家的右下角 x 座標
	int y2 = y1 + 30;	// 玩家的右下角 y 座標
	return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
}
bool box::OnBox(int tx1, int ty1, int tx2, int ty2)
{
	int x1 = x + dx;
	int y1 = y + dy;
	int x2 = x1 + bmp.Width();
	int y2 = y1 + bmp.Height();

	return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
}
//依據地圖陣列設置重力世界地板
void box::setfloor()
{
	if ((map[(y + 30) / 10][(x) / 10]) && (map[(y + 30) / 10][(x + 30) / 10]) == 1) {
		if ((map[(y + 40) / 10][(x) / 10]) && (map[(y + 40) / 10][(x + 30) / 10]) == 1) {
			if ((map[(y + 50) / 10][(x) / 10]) && (map[(y + 50) / 10][(x + 30) / 10]) == 1) {
				if ((map[(y + 60) / 10][(x) / 10]) && (map[(y + 60) / 10][(x + 30) / 10]) == 1) {
					if ((map[(y + 70) / 10][(x) / 10]) && (map[(y + 70) / 10][(x + 30) / 10]) == 1) {
						floor = (((y + 30) / 10) + 5) * 10;
					}
					else {
						floor = (((y + 30) / 10) + 4) * 10;
					}
				}
				else {
					floor = (((y + 30) / 10) + 3) * 10;
				}
			}
			else {
				floor = (((y + 30) / 10) + 2) * 10;
			}
		}
		else {
			floor = (((y + 30) / 10) + 1) * 10;
		}
	}
	else {
		floor = ((y + 30) / 10) * 10;
	}
	if (floor >= 580) {
		floor = 579;
	}
}
void box::LoadBitmap()
{
	bmp.LoadBitmap(IDB_BOX, RGB(0, 0, 0));
}
//依據地圖陣列檢測牆壁、天花板
bool box::isLeftRightEmpty(int x, int y, int value)
{
	int x_coord = 0, ycoord = 0;
	if (x < 21 || x>778 || y < 21 || y>578)
	{
		return 0;
	}
	bool result = 1;
	if (value == 0) {
		for (int i = 0; i < 800; i++)
		{
			if (x == x_edge[i]) {
				x_coord = i;
			}
		}
		for (int i = 0; i < 600; i++)
		{
			if (y + value == y_edge[i]) {
				ycoord = i;
			}
		}
		result = map[ycoord / 10][x_coord / 10] && result;
	}
	else
	{
		for (int i = 0; i < 800; i++)
		{
			if (x == x_edge[i]) {
				x_coord = i;
			}
		}

		for (int j = 5; j < 35; j += 3)
		{
			for (int i = 0; i < 600; i++)
			{
				if (y + j >= y_edge[i]) {
					ycoord = i;
				}
			}
			result = map[ycoord / 10][x_coord / 10] && result;
		}

	}

	return map[ycoord / 10][x_coord / 10];
}
//移動包含慣性
void box::OnMove()
{
	y = floor - 30;
	if (isMovingLeft) {
		if (isLeftRightEmpty(x - 7, y, 1) && x > 20) {
			x -= 7;
			setfloor();
		}
		inertia = true;
		isMovingLeft = false;
	}
	if (inertia)
```

```
            {
                if (velocity > 0)
                {
                    if (!isLeftRightEmpty(x - 1, y, 0))
                    {
                        velocity--;
                    }
                    else {
                        x -= velocity;
                        velocity--;
                        setfloor();
                    }
                }
                else {
                    inertia = false;
                    velocity = initial_velocity;
                }
            }
        }
        void box::SetIsAlive(bool alive)
        {
            is_alive = alive;
        }

        void box::SetXY(int nx, int ny)
        {
            x = nx;
            y = ny;
        }
        int box::GetX1()
        {
            return x;
        }
        int box::GetY1()
        {
            return y;
        }
        void box::SetMovingLeft(bool flag)
        {
            isMovingLeft = flag;
        }
        void box::OnShow()
        {
            bmp.SetTopLeft(x, y);
            bmp.ShowBitmap();
        }
    }

    <Button.h>
    #include "RedPlayer.h"
    #include "IcePlayer.h"
    #pragma once
    namespace game_framework {
    /////////////////////////////////////////////////////////////////////////////
    // 這個 class 是按鈕和上升桿的 Class
    /////////////////////////////////////////////////////////////////////////////
        class Button
        {
        public:
            Button();
            bool HitPlayer(RedPlayer *player);                              // 是否碰到玩家
            bool HitPlayer(IcePlayer *player);
            bool IsAlive();                                                 // 是否活著
            void LoadBitmap();                                              // 載入圖形
            void OnMove();                                                  // 移動
            void OnMove1();
            void OnMove2();
            void OnMove3();
            void OnShow();                                                  // 將圖形貼到畫面
            void OnShow1();
            void OnShow2();
            void OnShow3();
            void SetXY(int nx, int ny);                                    // 設定座標
            int GetY();
            int GetX();
            void SetIsAlive(bool alive);                                   // 設定是否活著
        protected:
            CMovingBitmap but, mo ,mo2 ,mo3;
            int x, y;                          // 座標
            bool is_alive;                     // 是否活著
        private:
            bool HitRectangle(int tx1, int ty1, int tx2, int ty2);    // 是否碰到參數範圍的矩形
        };
}

<Button.cpp>
#include "RedPlayer.h"
#include "IcePlayer.h"
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "Button.h"

namespace game_framework {
        Button::Button()
        {
            is_alive = true;
            x = y = 0;
        }

        bool Button::HitPlayer(IcePlayer *player)
        {
            return HitRectangle(player->GetX1(), player->GetY1(),
                player->GetX2(), player->GetY2());
        }
        bool Button::HitPlayer(RedPlayer *player)
        {
            return HitRectangle(player->GetX1(), player->GetY1(),
                player->GetX2(), player->GetY2());
        }

        bool Button::HitRectangle(int tx1, int ty1, int tx2, int ty2)
        {
            int x1 = x;                        // 玩家的左上角 x 座標
            int y1 = y;                        // 玩家的左上角 y 座標
            int x2 = x1 + but.Width();         // 玩家的右下角 x 座標
            int y2 = y1 + but.Height();        // 玩家的右下角 y 座標
            return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
        }

        bool Button::IsAlive()
        {
            return is_alive;
        }
        void Button::LoadBitmap()
        {
            but.LoadBitmap(BUTTON_1, RGB(255, 255, 255));
            mo.LoadBitmap(MOOD, RGB(255, 255, 255));
            mo2.LoadBitmap(MOOD_2, RGB(255, 255, 255));
            mo3.LoadBitmap(MOOD_3, RGB(255, 255, 255));
        }
        void Button::OnMove()
        {
            return;
        }
```

```
void Button::OnMove1()
{
	if (is_alive) {
		if (y >= 230) {
			y -= 1;
		}
		else {
			y = y;
		}
	}
	else {
		if (y <= 331) {
			y += 1;
		}
		else {
			y = y;
		}
	}
	return;
}
void Button::OnMove2()
{
	if (is_alive)
		if (y <= 330) {
			y += 1;
		}
		else {
			y = y;
		}
	else {
		if (y >= 250) {
			y -= 1;
		}
		else {
			y = y;
		}
	}
	return;
}
void Button::OnMove3()
{
	if (!is_alive) {
		if (x >= 350) {
			x -= 1;
		}
		else {
			x = x;
		}
	}
	else {
		if (x <= 450) {
			x += 1;
		}
		else {
			x = x;
		}
	}
	return;
}

void Button::SetIsAlive(bool alive)
{
	is_alive = alive;
}

void Button::SetXY(int nx, int ny)
{
	x = nx; y = ny;
}
```

```
	int Button::GetY()
	{
		return y;
	}
	int Button::GetX()
	{
		return x;
	}
	void Button::OnShow()
	{
		if (is_alive) {
			but.SetTopLeft(x, y);
			but.ShowBitmap();
		}
		else {
		}
	}
	void Button::OnShow1()
	{
		if (is_alive) {
			mo.SetTopLeft(x, y);
			mo.ShowBitmap();
		}
		else {
			mo.SetTopLeft(x, y);
			mo.ShowBitmap();
		}
	}
	void Button::OnShow2()
	{
		if (is_alive) {
			mo2.SetTopLeft(x, y);
			mo2.ShowBitmap();
		}
		else {
			mo2.SetTopLeft(x, y);
			mo2.ShowBitmap();
		}
	}
	void Button::OnShow3()
	{
		if (is_alive) {
			mo3.SetTopLeft(x, y);
			mo3.ShowBitmap();
		}
		else {
			mo3.SetTopLeft(x, y);
			mo3.ShowBitmap();
		}
	}
}
```

<Greenlake.h>

```
#include "RedPlayer.h"
#include "IcePlayer.h"
#pragma once
namespace game_framework {
/////////////////////////////////////////////////////////////////////////////
// 這個 class 是綠水的 Class
/////////////////////////////////////////////////////////////////////////////
	class Greenlake
	{
	public:
		Greenlake();
		bool HitPlayer(RedPlayer *player);
		bool HitPlayer(IcePlayer *player);
		void LoadBitmap();
		void OnMove();
```

```
        void OnShow();
        void SetXY(int nx, int ny);
        bool hack;
    protected:
        CMovingBitmap LAKE;
        int x, y;
    private:
        bool HitRectangle(int tx1, int ty1, int tx2, int ty2);
    };
}

<Greenlake.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "RedPlayer.h"
#include "IcePlayer.h"
#include "Greenlake.h"

namespace game_framework {
    Greenlake::Greenlake()
    {
        x = y = 0;
        hack = false;
    }

    bool Greenlake::HitPlayer(RedPlayer *player)
    {
        return HitRectangle(player->GetX1(), player->GetY1(),
            player->GetX2(), player->GetY2());
    }
    bool Greenlake::HitPlayer(IcePlayer *player)
    {

        return HitRectangle(player->GetX1(), player->GetY1(),
            player->GetX2(), player->GetY2());
    }

    bool Greenlake::HitRectangle(int tx1, int ty1, int tx2, int ty2)
    {
        int x1 = x +15;                     // 玩家的左上角 x 座標
        int y1 = y -5;                      // 玩家的左上角 y 座標
        int x2 = x1 + LAKE.Width() -30;     // 玩家的右下角 x 座標
        int y2 = y1 + LAKE.Height() - 5;    // 玩家的右下角 y 座標
        return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
    }

    void Greenlake::LoadBitmap()
    {
        LAKE.LoadBitmap(GREEN_LAKE, RGB(255, 255, 255));            // 載入圖形
    }
    void Greenlake::OnMove()
    {

    }

    void Greenlake::SetXY(int nx, int ny)
    {
        x = nx; y = ny;
    }
    void Greenlake::OnShow()
    {
        LAKE.SetTopLeft(x, y);
        LAKE.ShowBitmap();
    }
}
```

```
<IceDiamond.h>
#include "IcePlayer.h"
#pragma once
namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是冰鑽的 Class
	/////////////////////////////////////////////////////////////////////////////
	class IceDiamond
	{
	public:
		IceDiamond();
		bool HitPlayer(IcePlayer *player);					// 是否碰到玩家
		bool IsAlive();								// 是否活著
		void LoadBitmap();							// 載入圖形
		void OnMove();								// 移動
		void OnShow();								// 將圖形貼到畫面
		void SetXY(int nx, int ny);						// 設定座標
		void SetIsAlive(bool alive);					// 設定是否活著
	protected:
		CMovingBitmap bmp;
		int x, y;				// 座標
		bool is_alive;				// 是否活著
	private:
		bool HitRectangle(int tx1, int ty1, int tx2, int ty2);	// 是否碰到參數範圍的矩形
	};
}

<IceDiamond.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "IcePlayer.h"
#include "IceDiamond.h"

namespace game_framework {
	IceDiamond::IceDiamond()
	{
		is_alive = true;
		x = y = 0;
	}

	bool IceDiamond::HitPlayer(IcePlayer *player)
	{
		return HitRectangle(player->GetX1(), player->GetY1(),
			player->GetX2(), player->GetY2());
	}

	bool IceDiamond::HitRectangle(int tx1, int ty1, int tx2, int ty2)
	{
		int x1 = x;				// 玩家的左上角 x 座標
		int y1 = y;				// 玩家的左上角 y 座標
		int x2 = x1 + bmp.Width();	// 玩家的右下角 x 座標
		int y2 = y1 + bmp.Height();	// 玩家的右下角 y 座標
		return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
	}

	bool IceDiamond::IsAlive()
	{
		return is_alive;
	}

	void IceDiamond::LoadBitmap()
	{
		bmp.LoadBitmap(ICE_DIAMOND, RGB(255, 255, 255));			// 載入圖形
	}
	void IceDiamond::OnMove()
	{
		if (!is_alive)
			return;
	}

	void IceDiamond::SetIsAlive(bool alive)
	{
		is_alive = alive;
	}

	void IceDiamond::SetXY(int nx, int ny)
	{
		x = nx; y = ny;
	}

	void IceDiamond::OnShow()
	{
		if (is_alive) {
			bmp.SetTopLeft(x, y);
			bmp.ShowBitmap();
		}
	}
}

< IceDoor .h>
#include "IcePlayer.h"
#pragma once
namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是冰通關門的 Class
	/////////////////////////////////////////////////////////////////////////////
	class IceDoor
	{
	public:
		IceDoor();
		bool HitPlayer(IcePlayer *player);				// 是否碰到玩家
		bool IsAlive();								// 是否活著
		void LoadBitmap();							// 載入圖形
		void OnMove();								// 移動
		void OnShow();								// 將圖形貼到畫面
		void SetXY(int nx, int ny);					// 設定座標
		void SetIsAlive(bool alive);				// 設定是否活著
	protected:
		CMovingBitmap door, bmp2;
		int x, y;					// 座標
		bool is_alive;				// 是否活著
	private:
		bool HitRectangle(int tx1, int ty1, int tx2, int ty2);	// 是否碰到參數範圍的矩形
	};
}

< IceDoor .cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "IcePlayer.h"
#include "IceDoor.h"

namespace game_framework {
	IceDoor::IceDoor()
	{
		is_alive = true;
		x = y = 0;
	}

	bool IceDoor::HitPlayer(IcePlayer *player)
	{
		return HitRectangle(player->GetX1(), player->GetY1(),
			player->GetX2(), player->GetY2());
	}

	bool IceDoor::HitRectangle(int tx1, int ty1, int tx2, int ty2)
	{
		int x1 = x ;				// 玩家的左上角 x 座標
		int y1 = y ;				// 玩家的左上角 y 座標
		int x2 = x1 + door.Width();		// 玩家的右下角 x 座標
		int y2 = y1 + door.Height();		// 玩家的右下角 y 座標
		return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
	}

	bool IceDoor::IsAlive()
	{
		return is_alive;
	}

	void IceDoor::LoadBitmap()
	{
		door.LoadBitmap(ICE_DOOR_CLOSE, RGB(255, 255, 255));
		bmp2.LoadBitmap(DOOR_OPEN, RGB(255, 255, 255));
	}

	void IceDoor::OnMove()
	{
	}

	void IceDoor::SetIsAlive(bool alive)
	{
		is_alive = alive;
	}

	void IceDoor::SetXY(int nx, int ny)
	{
		x = nx; y = ny;
	}

	void IceDoor::OnShow()
	{
		if (is_alive) {
			door.SetTopLeft(x, y);
			door.ShowBitmap();
		}
		else {
			bmp2.SetTopLeft(x, y);
			bmp2.ShowBitmap();
		}
	}
}

< IceLake .h>
#include "RedPlayer.h"
#pragma once
namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是藍水的 Class
	/////////////////////////////////////////////////////////////////////////////
	class IceLake
	{
	public:
		IceLake();
		bool HitPlayer(RedPlayer *player);				// 是否碰到玩家
		void LoadBitmap();						// 載入圖形
		void OnMove();						// 移動
		void OnShow();						// 將圖形貼到畫面
		void SetXY(int nx, int ny);
		bool hack;							// 設定密技啟動布林
	protected:
		CMovingBitmap LAKE;
		int x, y;				// 座標
	private:
		bool HitRectangle(int tx1, int ty1, int tx2, int ty2);	// 是否碰到參數範圍的矩形
	};
}

< IceLake .cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "RedPlayer.h"
#include "IceLake.h"

namespace game_framework {
	IceLake::IceLake()
	{
		x = y = 0;
		hack = false;
	}

	bool IceLake::HitPlayer(RedPlayer *player)
	{
		return HitRectangle(player->GetX1(), player->GetY1(),
			player->GetX2(), player->GetY2());
	}

	bool IceLake::HitRectangle(int tx1, int ty1, int tx2, int ty2)
	{
		int x1 = x+15;				// 玩家的左上角 x 座標
		int y1 = y-5;				// 玩家的左上角 y 座標
		int x2 = x1 + LAKE.Width()-30;	// 玩家的右下角 x 座標
		int y2 = y1 + LAKE.Height();	// 玩家的右下角 y 座標
		return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
	}

	void IceLake::LoadBitmap()
	{
		LAKE.LoadBitmap(ICE_LAKE, RGB(255, 255, 255));
	}
	void IceLake::OnMove() {

	}



	void IceLake::SetXY(int nx, int ny)
	{
		x = nx; y = ny;
	}

	void IceLake::OnShow()
	{
		LAKE.SetTopLeft(x, y);
		LAKE.ShowBitmap();
	}
}
<IcePlayer.h>
#pragma once
#include "RedPlayer.h"
namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是冰人的 Class
	/////////////////////////////////////////////////////////////////////////////
```

```
    class IcePlayer
    {
    public:
        void SetMovingDown(bool flag);      // 設定是否正在往下移動
        void SetMovingLeft(bool flag);      // 設定是否正在往左移動
        void SetMovingRight(bool flag); // 設定是否正在往右移動
        void SetMovingUp(bool flag);        // 設定是否正在往上移動
        void SetButton(bool flag);
        void SetButton2(bool flag);
        void SetButton3(bool flag);
        void SetMood(bool flag);
        void MoodY(int MY);
        IcePlayer();
        int GetX1();
        int GetY1();
        int GetX2();
        int GetY2();
        void Initialize(int stages);                    // 設定玩家為初始值
        void LoadBitmap();                          // 載入圖形
        void OnMove();                              // 移動玩家
        void OnMove1();
        void OnShow();                              // 將玩家圖形貼到畫面
        void SetXY(int nx, int ny);             // 設定玩家左上角座標
        void SetFloor(int y);
        void SetVelocity(int);
        bool isLeftRightEmpty(int x, int y, int value);
        int getCoordX(int x, int y);
        int getCoordY(int x, int y);
        void setfloor();
        bool isOnButton;
        bool frontBox(int x, int y);
        bool onBox(int x, int y);
        void setFront(bool a);
        void setOnBox(bool a);
        bool butin();
        int map[60][80];
        int x_edge[800];
        int y_edge[600];
    protected:
        CAnimation animation, animation1;           // 玩家的動畫
        CMovingBitmap bit;
        CGameMap gamemap;
        int x, y;                               // 玩家左上角座標
        bool setFloorEnable;
        int floor;
        int velocity;                           // 目前的速度(點/次)
        int initial_velocity;               // 初始速度
        bool rising;                            // true 表上升、false 表下降
        bool isMovingDown;                  // 是否正在往下移動
        bool isMovingLeft;                  // 是否正在往左移動
        bool isMovingRight;                 // 是否正在往右移動
        bool isMovingUp;                    // 是否正在往上移動
        bool isFrontBox;
        bool isOnBox;
        bool isButton;
        bool isButton2;
        bool isButton3;
        bool isMood;
        bool isBox;
        int stage;
    };
}

<IcePlayer.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
```

```
#include "audio.h"
#include "gamelib.h"
#include "IcePlayer.h"

namespace game_framework {
	IcePlayer::IcePlayer()
	{
		Initialize(stage);
	}

	int IcePlayer::GetX1()
	{
		return x;
	}

	int IcePlayer::GetY1()
	{
		return y;
	}

	int IcePlayer::GetX2()
	{
		return x + animation.Width();
	}

	int IcePlayer::GetY2()
	{
		return y + animation.Height();
	}
	void IcePlayer::Initialize(int stages)
	{
		stage = stages;
		gamemap.ReadFile(stages);
		int INITIAL_VELOCITY;
		if (stages == 1)
		{
			INITIAL_VELOCITY = 11;
		}
		else
		{
			INITIAL_VELOCITY = 15;
		}
		const int FLOOR = 462;
		const int X_POS = 42;
		const int Y_POS = 542;
		floor = FLOOR;
		isMovingLeft = isMovingRight = isMovingUp = isMovingDown = isOnBox = isFrontBox = isOnButton= false;
		rising = false;
		initial_velocity = INITIAL_VELOCITY;
		velocity = initial_velocity;
		for (int i = 0; i < 60; i++)
		{
			for (int j = 0; j < 80; j++)
			{
				map[i][j] = gamemap.mapCoordinate(j, i);
			}
		}
		for (int j = 0; j < 800; j++)
		{
			x_edge[j] = j + 1;
		}
		for (int j = 0; j < 600; j++)
		{
			y_edge[j] = j + 1;
		}
	}
	void IcePlayer::LoadBitmap()
	{
		animation.AddBitmap(ICE_RIGHT_RUN_1, RGB(255, 255, 255));
	animation.AddBitmap(ICE_RIGHT_RUN_2, RGB(255, 255, 255));
	animation.AddBitmap(ICE_RIGHT_RUN_3, RGB(255, 255, 255));
	animation1.AddBitmap(ICE_LEFT_RUN_1, RGB(255, 255, 255));
	animation1.AddBitmap(ICE_LEFT_RUN_2, RGB(255, 255, 255));
	animation1.AddBitmap(ICE_LEFT_RUN_3, RGB(255, 255, 255));
	bit.LoadBitmap(ICE_FRONT, RGB(255, 255, 255));
}
bool IcePlayer::frontBox(int bx, int by)
{
	int x1 = bx;
	int y1 = by;
	int x2 = x1 + 35;
	int y2 = y1 + 35;
	return (x + 40 >= x1 && x <= x2 && y + 40 >= y1 && y <= y2);
}
bool IcePlayer::onBox(int bx, int by)
{
	int x1 = bx;
	int y1 = by - 20;
	int x2 = x1 + 35;
	int y2 = y1 + 40;
	return (x + 40 >= x1 && x <= x2 && y + 40 >= y1 && y <= y2);
}
void IcePlayer::setOnBox(bool a) {
	isOnBox = a;
}
void IcePlayer::setFront(bool a)
{
	isFrontBox = a;
}
bool IcePlayer::isLeftRightEmpty(int x, int y, int value)
{
	int x_coord = 0, ycoord = 0;
	if (x < 21 || x>788 || y < 21 || y>578)
	{
		return 0;
	}
	bool result = 1;
	if (value == 0) {
		for (int i = 0; i < 800; i++)
		{
			if (x == x_edge[i]) {
				x_coord = i;
			}
		}
		for (int i = 0; i < 600; i++)
		{
			if (y + value == y_edge[i]) {
				ycoord = i;
			}
		}
		result = map[ycoord / 10][x_coord / 10] && result;
	}
	else
	{
		for (int i = 0; i < 800; i++)
		{
			if (x == x_edge[i]) {
				x_coord = i;
			}
		}

		for (int j = 5; j < 35; j += 3)
		{
			for (int i = 0; i < 600; i++)
			{
				if (y + j == y_edge[i]) {
					ycoord = i;
				}
```

```
                }
                result = map[ycoord / 10][x_coord / 10] && result;
            }

        }

        return map[ycoord / 10][x_coord / 10];
    }
    int IcePlayer::getCoordX(int x, int y)
    {
        int x_coord = 0, ycoord = 0;
        for (int i = 0; i < 800; i++)
        {
            if (x >= x_edge[i]) {
                x_coord = i;
            }
        }
        return x_coord;
    }
    int IcePlayer::getCoordY(int x, int y)
    {
        int x_coord = 0, ycoord = 0;
        for (int i = 0; i < 600; i++)
        {
            if (y >= y_edge[i]) {
                ycoord = i;
            }
        }
        return ycoord;
    }
    void IcePlayer::setfloor()
    {
        if (!isOnButton)
        {
            if ((map[(y + 38) / 10][(x) / 10]) && (map[(y + 38) / 10][(x + 30) / 10]) == 1) {
                if ((map[(y + 48) / 10][(x) / 10]) && (map[(y + 48) / 10][(x + 30) / 10]) == 1) {
                    if ((map[(y + 58) / 10][(x) / 10]) && (map[(y + 58) / 10][(x + 30) / 10]) == 1) {
                        if ((map[(y + 68) / 10][(x) / 10]) && (map[(y + 68) / 10][(x + 30) / 10]) == 1) {
                            if ((map[(y + 78) / 10][(x) / 10]) && (map[(y + 78) / 10][(x + 30) / 10]) == 1) {
                                if ((map[(y + 88) / 10][(x) / 10]) && (map[(y + 88) / 10][(x + 30) / 10]) == 1) {
                                    if ((map[(y + 98) / 10][(x) / 10]) && (map[(y + 98) / 10][(x + 30) / 10]) == 1) {
                                        if ((map[(y + 108) / 10][(x) / 10]) && (map[(y + 108) / 10][(x + 30) / 10]) == 1)
{
                                            if ((map[(y + 118) / 10][(x) / 10]) && (map[(y + 118) / 10][(x + 30) / 10])
== 1) {

                                                floor = (((y + 38) / 10) + 9) * 10;
                                            }
                                            else {
                                                floor = (((y + 38) / 10) + 8) * 10;
                                            }
                                        }
                                        else {
                                            floor = (((y + 38) / 10) + 7) * 10;
                                        }
                                    }
                                    else {
                                        floor = (((y + 38) / 10) + 6) * 10;
                                    }
                                }
                                else {
                                    floor = (((y + 38) / 10) + 5) * 10;
                                }
                            }
                            else {
                                floor = (((y + 38) / 10) + 4) * 10;
                            }
                        }
                        else {
```

```
                        floor = (((y + 38) / 10) + 3) * 10;
                    }
                }
                else {
                    floor = (((y + 38) / 10) + 2) * 10;
                }
            }
            else {
                floor = (((y + 38) / 10) + 1) * 10;
            }
        }
        else {
            floor = ((y + 38) / 10) * 10;
        }
    }
    if (floor >= 580) {
        floor = 579;
    }
    if ((x + 20 >= 122 && x + 20 < 157 && y + 40 >= 21 && y + 40 < 229) && isOnBox)
    {
        floor = 194;
    }
}
void IcePlayer::OnMove1() {
    if (!rising && velocity == initial_velocity) {
        y = floor - 38;

    }
}
void IcePlayer::OnMove()
{
    const int STEP_SIZE = 7;
    animation.OnMove();
    animation1.OnMove();
    if (isMood) {
        for (int i = 2; i <= 8; i++) {
            map[37][i] = 0;
        }
    }
    else {
        for (int i = 2; i <= 8; i++) {
            map[37][i] = 1;
        }
    }
    if (stage == 2) {
        if (isButton2) {
            for (int i = 11; i <= 12; i++) {
                for (int j = 35; j <= 44; j++) {
                    map[i][j] = 0;
                }
            }
        }
        else {
            for (int i = 11; i <= 12; i++) {
                for (int j = 35; j <= 44; j++) {
                    map[i][j] = 1;
                }
            }
        }
        if (isButton3) {
            for (int i = 33; i <= 40; i++) {
                for (int j = 40; j <= 41; j++) {
                    map[i][j] = 1;
                }
            }
        }
        else {
            for (int i = 33; i <= 40; i++) {
                for (int j = 40; j <= 41; j++) {
                    map[i][j] = 0;
                }
            }
        }
    }
    if (!rising && velocity == initial_velocity) {
        y = floor - 38;

    }

    if (isMovingLeft)
        if (isLeftRightEmpty(x - STEP_SIZE, y, 1) && x > 20 && isFrontBox == false) {
            x -= STEP_SIZE;
            setfloor();
        }
    if (isMovingRight)
        if (isLeftRightEmpty(x + 38 + STEP_SIZE, y, 1) && x < 778) {
            x += STEP_SIZE;
            setfloor();
        }
    if (isMovingUp) {
        rising = true;
        isMovingUp = false;
    }
    if (rising) {                  // 上升狀態
        if (velocity > 0) {
            if (!isLeftRightEmpty(x, y - 1, 0))
            {
                velocity -= 2;
                setfloor();
            }
            else
            {
                y -= velocity;      // 當速度 > 0 時，y 軸上升(移動 velocity 個點，velocity 的單位為 點/次)
                velocity--;           // 受重力影響，下次的上升速度降低
                setfloor();
            }

        }
        else {
            rising = false; // 當速度 <= 0，上升終止，下次改為下降
            velocity = 1;      // 下降的初速(velocity)為 1
        }

    }
    else {                         // 下降狀態
        if (y < floor - 39) {   // 當 y 座標還沒碰到地板
            if (velocity < 9) {
                y += velocity;      // y 軸下降(移動 velocity 個點，velocity 的單位為 點/次)
                velocity++;           // 受重力影響，下次的下降速度增加
                setfloor();

            }
            else {
                y += velocity;
                setfloor();
            }
        }
        else {
            setfloor();
            y = floor - 38;   // 當 y 座標低於地板，更正為地板上
            rising = false;      // 探底反彈，下次改為上升
            velocity = initial_velocity; // 重設上升初始速度

        }
        isMovingUp = false;
    };
    if (isMovingDown)
        y = floor - 38;
```

```
}
void IcePlayer::SetButton(bool flag) {
        isButton = flag;
}
void IcePlayer::SetButton2(bool flag) {
        isButton2 = flag;
}
void IcePlayer::SetButton3(bool flag) {
        isButton3 = flag;
}
void IcePlayer::SetMood(bool flag) {
        isMood = flag;
}
void IcePlayer::MoodY(int MY) {
        MY = MY;
}
bool IcePlayer::butin() {
        int x1 = x;
        int y1 = y;
        int x2 = x1 + bit.Width();
        int y2 = y1 + bit.Height();
        if (x2 >= 709 && x1 < 790 && y2 >= 192 && y1 <= 292) {
                return 1;
        }
        else {
                return 0;
        }
}
void IcePlayer::SetFloor(int y) {
        floor = y;
}
void IcePlayer::SetMovingDown(bool flag)
{
        isMovingDown = flag;
}

void IcePlayer::SetMovingLeft(bool flag)
{
        isMovingLeft = flag;
}

void IcePlayer::SetMovingRight(bool flag)
{
        isMovingRight = flag;
}

void IcePlayer::SetMovingUp(bool flag)
{
        isMovingUp = flag;
}

void IcePlayer::SetXY(int nx, int ny)
{
        x = nx; y = ny;
}

void IcePlayer::SetVelocity(int velocity) {
        this->velocity = velocity;
        this->initial_velocity = velocity;
}
void IcePlayer::OnShow() {
        if (isMovingLeft) {
                animation1.SetTopLeft(x, y);
                animation1.OnShow();
        }
        else if (isMovingRight)
        {
                animation.SetTopLeft(x, y);
                animation.OnShow();
        }

        else if (!isMovingRight && !isMovingLeft)
        {
            bit.SetTopLeft(x, y);
            bit.ShowBitmap();
        }
    }
}

<Mood.h>
#include "RedPlayer.h"
#include "IcePlayer.h"
#pragma once
namespace game_framework {
    /////////////////////////////////////////////////////////////////////////////
    // 這個 class 是拉桿的 Class
    /////////////////////////////////////////////////////////////////////////////
    class Mood
    {
        public:
            Mood();
            bool HitPlayer(RedPlayer *player);                    // 是否碰到玩家
            bool HitPlayer(IcePlayer *player);
            bool IsAlive();                                       // 是否活著
            void LoadBitmap();                                    // 載入圖形
            void OnMove();                                        // 移動
            void OnMove1();
            void OnShow();                                        // 將圖形貼到畫面
            void OnShow1();
            void SetXY(int nx, int ny);                           // 設定座標
            int ReY();
            void SetIsAlive(bool alive);                          // 設定是否活著
        protected:
            CMovingBitmap rm ,mm ,lm, mo , mo2;
            int x, y;                         // 座標
            bool is_alive;                    // 是否活著
        private:
            bool HitRectangle(int tx1, int ty1, int tx2, int ty2);    // 是否碰到參數範圍的矩形

    };
}

<Mood.cpp>

#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "RedPlayer.h"
#include "IcePlayer.h"
#include "Mood.h"


namespace game_framework {
    Mood::Mood()
    {
        is_alive = true;
        x = y = 0;
    }
    bool Mood::HitPlayer(RedPlayer *player)
    {
        return HitRectangle(player->GetX1(), player->GetY1(),
            player->GetX2(), player->GetY2());
    }
    bool Mood::HitPlayer(IcePlayer *player)
    {
        return HitRectangle(player->GetX1(), player->GetY1(),
                player->GetX2(), player->GetY2());
}

bool Mood::HitRectangle(int tx1, int ty1, int tx2, int ty2)
{
        int x1 = x + rm.Width();                                // 球的左上角 x 座標
        int y1 = y;                             // 球的左上角 y 座標
        int x2 = x + rm.Width();        // 球的右下角 x 座標
        int y2 = y1 + rm.Height();      // 球的右下角 y 座標
        return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
}

bool Mood::IsAlive()
{
        return is_alive;
}

void Mood::LoadBitmap()
{
        rm.LoadBitmap(RIGHT_MOOD, RGB(255, 255, 255));
        mm.LoadBitmap(MID_MOOD, RGB(255, 255, 255));
        lm.LoadBitmap(LEFT_MOOD, RGB(255, 255, 255));
        mo.LoadBitmap(MOOD, RGB(255, 255, 255));

}
void Mood::OnMove()
{
        return;
}
void Mood::OnMove1()
{
        if (is_alive)
                if (y >= 310) {
                        y -= 1;
                }
                else {
                        y = y;
                }
        else {
                if (y <=370) {
                        y += 1;
                }
                else {
                        y = y;
                }
        }
        return;
}

void Mood::SetIsAlive(bool alive)
{
        is_alive = alive;
}

void Mood::SetXY(int nx, int ny)
{
        x = nx; y = ny;
}

int Mood::ReY()
{
        return y;
}

void Mood::OnShow()
{
        if (is_alive) {
                rm.SetTopLeft(x, y);
```

```
				rm.ShowBitmap();
			}
			else {
				lm.SetTopLeft(x, y);
				lm.ShowBitmap();
			}
		}
		void Mood::OnShow1()
		{
			if (is_alive) {
				mo.SetTopLeft(x, y);
				mo.ShowBitmap();
			}
			else {
				mo.SetTopLeft(x, y);
				mo.ShowBitmap();
			}
		}
}

<RedDiamond.h>
#include "RedPlayer.h"
#pragma once
namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是火鑽的 Class
	/////////////////////////////////////////////////////////////////////////////
	class RedDiamond
	{
		public:
			RedDiamond();
			bool HitPlayer(RedPlayer *player);					// 是否碰到玩家
			bool IsAlive();									// 是否活著
			void LoadBitmap();								// 載入圖形
			void OnMove();									// 移動
			void OnShow();									// 將圖形貼到畫面
			void SetXY(int nx, int ny);							// 設定座標
			void SetIsAlive(bool alive);						// 設定是否活著
		protected:
			CMovingBitmap bmp;
			int x, y;					// 座標
			bool is_alive;				// 是否活著
		private:
			bool HitRectangle(int tx1, int ty1, int tx2, int ty2);	// 是否碰到參數範圍的矩形
	};
}

<RedDiamond.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "RedPlayer.h"
#include "RedDiamond.h"

namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// RedDiamond class
	/////////////////////////////////////////////////////////////////////////////
	RedDiamond::RedDiamond()
	{
		is_alive = true;
		x = y = 0;
	}

	bool RedDiamond::HitPlayer(RedPlayer *player)
	{
			return HitRectangle(player->GetX1(), player->GetY1(),
				player->GetX2(), player->GetY2());
		}

		bool RedDiamond::HitRectangle(int tx1, int ty1, int tx2, int ty2)
		{
			int x1 = x ;				// 球的左上角 x 座標
			int y1 = y ;				// 球的左上角 y 座標
			int x2 = x1 + bmp.Width();		// 球的右下角 x 座標
			int y2 = y1 + bmp.Height();		// 球的右下角 y 座標
			return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
		}

		bool RedDiamond::IsAlive()
		{
			return is_alive;
		}

		void RedDiamond::LoadBitmap()
		{
			bmp.LoadBitmap(FIRE_DIAMOND, RGB(255, 255, 255));
		}
		void RedDiamond::OnMove()
		{
		}

		void RedDiamond::SetIsAlive(bool alive)
		{
			is_alive = alive;
		}

		void RedDiamond::SetXY(int nx, int ny)
		{
			x = nx; y = ny;
		}

		void RedDiamond::OnShow()
		{
			if (is_alive) {
				bmp.SetTopLeft(x , y );
				bmp.ShowBitmap();
			}
		}
	}

	<RedDoor.h>
	#include "RedPlayer.h"
	#pragma once
	namespace game_framework {
		/////////////////////////////////////////////////////////////////////////////
		// 這個 class 是火通關門的 Class
		/////////////////////////////////////////////////////////////////////////////
		class RedDoor
		{
			public:
				RedDoor();
				bool HitPlayer(RedPlayer *player);					// 是否碰到玩家
				bool IsAlive();								// 是否活著
				void LoadBitmap();							// 載入圖形
				void OnMove();								// 移動
				void OnShow();								// 將圖形貼到畫面
				void SetXY(int nx, int ny);						// 設定座標
				void SetIsAlive(bool alive);					// 設定是否活著
			protected:
				CMovingBitmap doors,bmp2;
				int x, y;				// 座標
				bool is_alive;				// 是否活著
			private:
				bool HitRectangle(int tx1, int ty1, int tx2, int ty2);	// 是否碰到參數範圍的矩形
	};
}

<RedDoor.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "RedPlayer.h"
#include "RedDoor.h"

namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是火通關門的 Class
	/////////////////////////////////////////////////////////////////////////////
	RedDoor::RedDoor()
	{
		is_alive = true;
		x = y = 0;
	}
	bool RedDoor::HitPlayer(RedPlayer *player)
	{
		return HitRectangle(player->GetX1(), player->GetY1(),
			player->GetX2(), player->GetY2());
	}

	bool RedDoor::HitRectangle(int tx1, int ty1, int tx2, int ty2)
	{
		int x1 = x ;				// 玩家的左上角 x 座標
		int y1 = y ;				// 玩家的左上角 y 座標
		int x2 = x1 + doors.Width();	// 玩家的右下角 x 座標
		int y2 = y1 + doors.Height();	// 玩家的右下角 y 座標
		return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
	}

	bool RedDoor::IsAlive()
	{
		return is_alive;
	}

	void RedDoor::LoadBitmap()
	{
		doors.LoadBitmap(FIRE_DOOR_CLOSE, RGB(255, 255, 255));
		bmp2.LoadBitmap(DOOR_OPEN, RGB(255, 255, 255));
	}

	void RedDoor::OnMove()
	{
	}

	void RedDoor::SetIsAlive(bool alive)
	{
		is_alive = alive;
	}

	void RedDoor::SetXY(int nx, int ny)
	{
		x = nx; y = ny;
	}

	void RedDoor::OnShow()
	{
		if (is_alive) {
			doors.SetTopLeft(x, y);
			doors.ShowBitmap();
		}
		else {
```

```
                bmp2.SetTopLeft(x, y);
                bmp2.ShowBitmap();
            }
        }
}

<RedLake.h>
#include "IcePlayer.h"
#pragma once
namespace game_framework {
    /////////////////////////////////////////////////////////////////////////////
    // 這個 class 是紅水的 Class
    /////////////////////////////////////////////////////////////////////////////
    class RedLake
    {
    public:
        RedLake();
        bool HitPlayer(IcePlayer *player);                          // 是否碰到玩家
        void LoadBitmap();                                           // 載入圖形
        void OnMove();                                               // 移動
        void OnShow();                                               // 將圖形貼到畫面
        void SetXY(int nx, int ny);
        bool hack;// 設定圓心的座標
    protected:
        CMovingBitmap LAKE;
        int x, y;                           // 座標
    private:
        bool HitRectangle(int tx1, int ty1, int tx2, int ty2);      // 是否碰到參數範圍的矩形
    };
}

< RedLake.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "IcePlayer.h"
#include "RedLake.h"

namespace game_framework {
    RedLake::RedLake()
    {
        this->hack = false;
        x = y = 0;
    }

    bool RedLake::HitPlayer(IcePlayer *player)
    {
        return HitRectangle(player->GetX1(), player->GetY1(),
            player->GetX2(), player->GetY2());
    }

    bool RedLake::HitRectangle(int tx1, int ty1, int tx2, int ty2)
    {
        int x1 = x +15;                     // 玩家的左上角 x 座標
        int y1 = y -5;                      // 玩家的左上角 y 座標
        int x2 = x1 + LAKE.Width() -30;     // 玩家的右下角 x 座標
        int y2 = y1 + LAKE.Height() -5;     // 玩家的右下角 y 座標
        return (tx2 >= x1 && tx1 <= x2 && ty2 >= y1 && ty1 <= y2);
    }


    void RedLake::LoadBitmap()
    {
        LAKE.LoadBitmap(FIRE_LAKE, RGB(255, 255, 255));
    }
```

```
	void RedLake::OnMove()
	{
	}

	void RedLake::SetXY(int nx, int ny)
	{
		x = nx; y = ny;

	}

	void RedLake::OnShow()
	{
		LAKE.SetTopLeft(x, y);
		LAKE.ShowBitmap();
	}
}

<RedPlayer.h>
#pragma once
#include <fstream>
extern bool current_rank;
namespace game_framework {
	/////////////////////////////////////////////////////////////////////////////
	// 這個 class 是火人和 CGameMap 的 Class
	/////////////////////////////////////////////////////////////////////////////
	class CGameMap {
	public:
		CGameMap();
		CGameMap(int _stage);
		~CGameMap();
		void LoadBitmap();
		void OnShow();
		void setMap(int x, int y, int v);
		void SetStage(int _stage);
		void ReadFile(int stages);
		int mapCoordinate(int x, int y);
		int GetX();
		int GetY();
		int GetMW();
		int GetMH();
		int GetSizeX();
		int GetSizeY();
	protected:
		CMovingBitmap blue, green;
		int map[60][80];
		int X, Y;
		int MW, MH;
		int sizeX, sizeY;
		int stage;
	};


	class RedPlayer {

	public:
		void SetMovingDown(bool flag);	// 設定是否正在往下移動
		void SetMovingLeft(bool flag);	// 設定是否正在往左移動
		void SetMovingRight(bool flag); // 設定是否正在往右移動
		void SetMovingUp(bool flag);	// 設定是否正在往上移動
		void SetButton(bool flag);
		void SetButton2(bool flag);
		void SetButton3(bool flag);
		void SetMood(bool flag);
		void MoodY(int MY);
		RedPlayer();
		int GetX1();
		int GetY1();
		int GetX2();
		int GetY2();
```

```
            void Initialize(int stages);                    // 設定玩家為初始值
            void LoadBitmap();                          // 載入圖形
            void OnMove();                              // 移動玩家
            void OnMove1();
            void OnShow();                              // 將玩家圖形貼到畫面
            void SetXY(int nx, int ny);             // 設定玩家左上角座標
            void SetFloor(int y);
            void SetVelocity(int);
            bool isLeftRightEmpty(int x, int y, int value);
            int getCoordX(int x, int y);
            int getCoordY(int x, int y);
            void setfloor();
            bool frontBox(int x, int y);
            bool onBox(int x, int y);
            bool butin();
            bool isOnButton;
            void setFront(bool a);
            void setOnBox(bool a);
            int map[60][80];
            int x_edge[800];
            int y_edge[600];
            int floor;
        protected:
            CAnimation animation, animation1;           // 玩家的動畫
            CMovingBitmap bit;
            CGameMap gamemap;
            int x, y;                               // 玩家左上角座標
            int velocity;                       // 目前的速度(點/次)
            int initial_velocity;       // 初始速度
            bool rising;                        // true 表上升、false 表下降
            bool isMovingDown;                      // 是否正在往下移動
            bool isMovingLeft;                  // 是否正在往左移動
            bool isMovingRight;                     // 是否正在往右移動
            bool isMovingUp;                    // 是否正在往上移動
            bool isFrontBox;
            bool isOnBox;
            bool isButton;
            bool isButton2;
            bool isButton3;
            bool isMood;
            bool isBox;
            int stage;
        };
}

<RedPlayer.cpp>
#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "RedPlayer.h"
#include "box.h"
#include <iostream>
#include <fstream>
#include <string.h>
#include <conio.h>
#include <time.h>
#include <atomic>
#include <chrono>
#include <thread>
#include <stdio.h>
#include <sstream>

namespace game_framework {
    CGameMap::CGameMap(int _stage) {

    }
```

```
CGameMap::CGameMap() :X(10), Y(10), MW(10), MH(10) {
	sizeX = 600 / X;
	sizeY = 800 / Y;
	stage = 1;
}

void CGameMap::ReadFile(int stages) {
	std::stringstream filename;
	std::stringstream ss;
	string str;
	ss << stages;
	ss >> str;
	filename << "map\\Run" << str << ".txt";
	ifstream is(filename.str(), std::ifstream::binary);
	std::string line;
	int i = 0;
	while (getline(is, line)) {
		for (int j = 0; j < sizeY; j++) {
			map[i][j] = line[j] - '0';
		}
		i++;
	}
	is.close();
}

void CGameMap::LoadBitmap() {
	blue.LoadBitmap(DOT, 1);
}

void CGameMap::OnShow() {
	for (int i = 0; i < 80; i++) {
		for (int j = 0; j < 60; j++) {
			switch (map[j][i]) {
			case 1:
				break;
			case 0:
				blue.SetTopLeft((10 * i), (10 * j));
				blue.ShowBitmap();
				break;
			}
		}
	}
}

void CGameMap::setMap(int x, int y, int v) {
	map[y][x] = v;
}

void CGameMap::SetStage(int _stage) {
	stage = _stage;
}

int CGameMap::mapCoordinate(int x, int y) {
	return map[y][x];
}

int CGameMap::GetX() {
	return X;
}

int CGameMap::GetY() {
	return Y;
}

int CGameMap::GetMH() {
	return MH;
}
```

```
int CGameMap::GetMW() {
	return MW;
}

int CGameMap::GetSizeX() {
	return sizeX;
}

int CGameMap::GetSizeY() {
	return sizeY;
}

CGameMap::~CGameMap() {
}

RedPlayer::RedPlayer()
{
	Initialize(stage);
}

int RedPlayer::GetX1()
{
	return x;
}

int RedPlayer::GetY1()
{
	return y;
}

int RedPlayer::GetX2()
{
	return x + animation.Width();
}

int RedPlayer::GetY2()
{
	return y + animation.Height();
}
void RedPlayer::Initialize(int stages)
{
	stage = stages;
	gamemap.~CGameMap();
	gamemap.ReadFile(stages);
	int INITIAL_VELOCITY;	// 初始上升速度
	if (stages == 1)
	{
		INITIAL_VELOCITY = 11;
	}
	else
	{
		INITIAL_VELOCITY = 15;
	}
	const int FLOOR = 579;// 地板座標
	const int X_POS = 42;
	const int Y_POS = 542;
	floor = FLOOR;
	isMovingLeft = isMovingRight = isMovingUp = isMovingDown = isOnBox = isFrontBox = isOnButton = false;
	rising = false;
	initial_velocity = INITIAL_VELOCITY;
	velocity = initial_velocity;
	for (int i = 0; i < 60; i++)
	{
		for (int j = 0; j < 80; j++)
		{
			map[i][j] = gamemap.mapCoordinate(j, i);
		}
	}
	for (int j = 0; j < 800; j++)
```

```
        {
            x_edge[j] = j + 1;
        }
        for (int j = 0; j < 600; j++)
        {
            y_edge[j] = j + 1;
        }
}
void RedPlayer::LoadBitmap()
{
        animation.AddBitmap(FIRE_RIGHT_RUN_1, RGB(255, 255, 255));
        animation.AddBitmap(FIRE_RIGHT_RUN_2, RGB(255, 255, 255));
        animation.AddBitmap(FIRE_RIGHT_RUN_3, RGB(255, 255, 255));
        animation1.AddBitmap(FIRE_LEFT_RUN_1, RGB(255, 255, 255));
        animation1.AddBitmap(FIRE_LEFT_RUN_2, RGB(255, 255, 255));
        animation1.AddBitmap(FIRE_LEFT_RUN_3, RGB(255, 255, 255));
        bit.LoadBitmap(FIRE_FRONT, RGB(255, 255, 255));
}
bool RedPlayer::frontBox(int bx, int by)
{
        int x1 = bx;
        int y1 = by;
        int x2 = x1 + 35;
        int y2 = y1 + 35;
        return (x + 40 >= x1 && x <= x2 && y + 40 >= y1 && y <= y2);
}
bool RedPlayer::onBox(int bx, int by)
{
        int x1 = bx;
        int y1 = by - 20;
        int x2 = x1 + 35;
        int y2 = y1 + 40;
        return (x + 40 >= x1 && x <= x2 && y + 40 >= y1 && y <= y2);
}
void RedPlayer::setOnBox(bool a) {
        isOnBox = a;
}
void RedPlayer::setFront(bool a)
{
        isFrontBox = a;
}
bool RedPlayer::isLeftRightEmpty(int x, int y, int value)
{
        int x_coord = 0, ycoord = 0;
        if (x < 21 || x>788 || y < 21 || y>578)
        {
            return 0;
        }
        bool result = 1;
        if (value == 0) {
            for (int i = 0; i < 800; i++)
            {
                if (x == x_edge[i]) {
                    x_coord = i;
                }
            }
            for (int i = 0; i < 600; i++)
            {
                if (y + value == y_edge[i]) {
                    ycoord = i;
                }
            }
            result = map[ycoord / 10][x_coord / 10] && result;
        }
        else
        {
            for (int i = 0; i < 800; i++)
            {
                if (x == x_edge[i]) {
                    x_coord = i;
                }
            }

            for (int j = 5; j < 35; j += 3)
            {
                for (int i = 0; i < 600; i++)
                {
                    if (y + j == y_edge[i]) {
                        ycoord = i;
                    }
                }
                result = map[ycoord / 10][x_coord / 10] && result;
            }

        }

        return map[ycoord / 10][x_coord / 10];
    }
    int RedPlayer::getCoordX(int x, int y)
    {
        int x_coord = 0, ycoord = 0;
        for (int i = 0; i < 800; i++)
        {
            if (x >= x_edge[i]) {
                x_coord = i;
            }
        }
        return x_coord;
    }
    int RedPlayer::getCoordY(int x, int y)
    {
        int x_coord = 0, ycoord = 0;
        for (int i = 0; i < 600; i++)
        {
            if (y >= y_edge[i]) {
                ycoord = i;
            }
        }
        return ycoord;
    }
    void RedPlayer::setfloor()
    {
        if (!isOnButton)
        {
            if ((map[(y + 38) / 10][(x) / 10]) && (map[(y + 38) / 10][(x + 30) / 10]) == 1) {
                if ((map[(y + 48) / 10][(x) / 10]) && (map[(y + 48) / 10][(x + 30) / 10]) == 1) {
                    if ((map[(y + 58) / 10][(x) / 10]) && (map[(y + 58) / 10][(x + 30) / 10]) == 1) {
                        if ((map[(y + 68) / 10][(x) / 10]) && (map[(y + 68) / 10][(x + 30) / 10]) == 1) {
                            if ((map[(y + 78) / 10][(x) / 10]) && (map[(y + 78) / 10][(x + 30) / 10]) == 1) {
                                if ((map[(y + 88) / 10][(x) / 10]) && (map[(y + 88) / 10][(x + 30) / 10]) == 1) {
                                    if ((map[(y + 98) / 10][(x) / 10]) && (map[(y + 98) / 10][(x + 30) / 10]) == 1) {
                                        if ((map[(y + 108) / 10][(x) / 10]) && (map[(y + 108) / 10][(x + 30) / 10]) == 1)
{
                                            if ((map[(y + 118) / 10][(x) / 10]) && (map[(y + 118) / 10][(x + 30) / 10])
== 1) {

                                                floor = (((y + 38) / 10) + 9) * 10;
                                            }
                                            else {
                                                floor = (((y + 38) / 10) + 8) * 10;
                                            }
                                        }
                                        else {
                                            floor = (((y + 38) / 10) + 7) * 10;
                                        }
                                    }
                                    else {
                                        floor = (((y + 38) / 10) + 6) * 10;
                                    }
                                }
                                else {
                                    floor = (((y + 38) / 10) + 5) * 10;
                                }
                            }
                            else {
                                floor = (((y + 38) / 10) + 4) * 10;
                            }
                        }
                        else {
                            floor = (((y + 38) / 10) + 3) * 10;
                        }
                    }
                    else {
                        floor = (((y + 38) / 10) + 2) * 10;
                    }
                }
                else {
                    floor = (((y + 38) / 10) + 1) * 10;
                }
            }
            else {
                floor = ((y + 38) / 10) * 10;
            }
        }

        if (floor >= 580) {
            floor = 579;
        }
        if ((x + 20 >= 122 && x + 20 < 157 && y + 40 >= 21 && y + 40 < 229) && isOnBox)
        {
            floor = 194;
        }
    }

    void RedPlayer::OnMove1() {
        if (!rising && velocity == initial_velocity) {
            y = floor - 38;
        }
    }
    void RedPlayer::OnMove()
    {
        const int STEP_SIZE = 7;
        animation.OnMove();
        animation1.OnMove();
        if (isMood) {
            for (int i = 2; i <= 8; i++) {
                map[37][i] = 0;
            }
        }
        else {
            for (int i = 2; i <= 8; i++) {
                map[37][i] = 1;
            }
        }
        if (stage == 2) {
            if (isButton2) {
                for (int i = 11; i <= 12; i++) {
                    for (int j = 35; j <= 44; j++) {
                        map[i][j] = 0;
                    }
                }
            }
            else {
                for (int i = 11; i <= 12; i++) {
                    for (int j = 35; j <= 44; j++) {
                        map[i][j] = 1;
                    }
                }
            }
            if (isButton3) {
                for (int i = 33; i <= 40; i++) {
                    for (int j = 40; j <= 41; j++) {
                        map[i][j] = 1;
                    }
                }
            }
            else {
                for (int i = 33; i <= 40; i++) {
                    for (int j = 40; j <= 41; j++) {
                        map[i][j] = 0;
                    }
                }
            }
        }
        if (!rising && velocity == initial_velocity) {
            y = floor - 38;

        }
        if (isMovingLeft)
            if (isLeftRightEmpty(x - STEP_SIZE, y, 1) && x > 20 && isFrontBox == false) {
                x -= STEP_SIZE;
                setfloor();
            }
        if (isMovingRight)
            if (isLeftRightEmpty(x + 38 + STEP_SIZE, y, 1) && x < 778) {
                x += STEP_SIZE;
                setfloor();
            }
        if (isMovingUp) {
            rising = true;
            isMovingUp = false;
        }
        if (rising) {                // 上升狀態
            if (velocity > 0) {
                if (!isLeftRightEmpty(x, y - 1, 0))
                {
                    velocity -= 2;
                    setfloor();
                }
                else
                {
                    y -= velocity;      // 當速度 > 0 時，y 軸上升(移動 velocity 個點，velocity 的單位為 點/次)
                    velocity--;         // 受重力影響，下次的上升速度降低
                    setfloor();
                }

            }
            else {
                rising = false; // 當速度 <= 0，上升終止，下次改為下降
                velocity = 1;     // 下降的初速(velocity)為 1
            }

        }
        else {                      // 下降狀態
            if (y < floor - 39) {   // 當 y 座標還沒碰到地板
                if (velocity < 9) {
                    y += velocity;      // y 軸下降(移動 velocity 個點，velocity 的單位為 點/次)
                    velocity++;         // 受重力影響，下次的下降速度增加
                    setfloor();

                }
                else {
                    y += velocity;
                    setfloor();
                }
            }
```

```
		else {
			setfloor();
			y = floor - 38;   // 當 y 座標低於地板，更正為地板上
			rising = false;      // 探底反彈，下次改為上升
			velocity = initial_velocity;
			// 重設上升初始速度
		}
		isMovingUp = false;
	};
	if (isMovingDown)
		y = floor - 38;
}
void RedPlayer::SetButton(bool flag) {
	isButton = flag;
}
void RedPlayer::SetButton2(bool flag) {
	isButton2 = flag;
}
void RedPlayer::SetButton3(bool flag) {
	isButton3 = flag;
}
void RedPlayer::SetMood(bool flag) {
	isMood = flag;
}
void RedPlayer::MoodY(int MY) {
	MY = MY;
}
bool RedPlayer::butin() {
	int x1 = x;
	int y1 = y;
	int x2 = x1 + bit.Width();
	int y2 = y1 + bit.Height();
	if (x2 >= 709 && x1 < 790 && y2 >= 192 && y1 <= 292) {
		return 1;
	}
	else {
		return 0;
	}
}

void RedPlayer::SetFloor(int y) {
	floor = y;
}
void RedPlayer::SetMovingDown(bool flag)
{
	isMovingDown = flag;
}

void RedPlayer::SetMovingLeft(bool flag)
{
	isMovingLeft = flag;
}

void RedPlayer::SetMovingRight(bool flag)
{
	isMovingRight = flag;
}

void RedPlayer::SetMovingUp(bool flag)
{
	isMovingUp = flag;
}

void RedPlayer::SetXY(int nx, int ny)
{
	x = nx; y = ny;
}

void RedPlayer::SetVelocity(int velocity) {
        this->velocity = velocity;
        this->initial_velocity = velocity;
    }
    void RedPlayer::OnShow() {
        if (isMovingLeft) {
            animation1.SetTopLeft(x, y);
            animation1.OnShow();
        }
        else if (isMovingRight)
        {
            animation.SetTopLeft(x, y);
            animation.OnShow();
        }

        else if (!isMovingRight && !isMovingLeft)
        {
            bit.SetTopLeft(x, y);
            bit.ShowBitmap();
        }
    }
}
```