**目錄**

**一、簡介………………………………………………………………………P.1**

**二、遊戲介紹…………………………………………………………………P.2**

**三、程式設計…………………………………………………………………P.5**

**四、結語………………………………………………………………………P.13**

**附錄……………………………………………………………………………P.17**

**一、簡介**

1.動機：

因為我從小就喜歡玩火柴人系列的遊戲，加上我們很喜歡這個遊戲的操作，這個遊戲我玩了快六個月的時間，其中有些關卡還有怪物攻擊的模式我都覺得遊戲做的很用心，所以決定做這個遊戲。

2.分工：

蔡育綸：程式撰寫、素材整理、蒐集

蘇佳湧：程式撰寫、建立架構

**二、遊戲介紹**

1.遊戲說明：

(1)按鍵：

上：跳躍。左：向左走。右：向右走。z：翻滾。x：攻擊。c丟手裡劍。e：跟NPC對話

(2)遊戲規則：透過打倒怪物，賺取金幣，打倒雕像即可通關。

(3)特殊功能：人物可以丟手裡劍。

(4)密技：翻滾可以閃躲所有攻擊，手裡劍可以穿牆、觸發開關。

2.遊戲圖形：

|  |  |
| --- | --- |
| Hero |  |
| Sword |  |
| Logo |  |
| 掉落物品 |  |
| Monitor |  |
| 尖刺 |  |
| 門 |  |
| Switch |  |
| Fire Stone |  |
| 仙人掌 |  |
| Cloud |  |
| GasRobot |  |
| Pigeon |  |
| Robot\_A |  |
| Scorpion |  |
| Smile\_SunFlower |  |
| 爆炸特效 |  |
| 雕像 |  |
| NPC\_Old\_Man |  |
| 死亡畫面 |  |
| 火焰特效 |  |

3.遊戲音效：

(1)爆炸：bomb.wav

(2)仙人掌攻擊：cactus\_attack.wav

(3)閃電：cloudy\_cloud\_attack.wav

(4)收集道具：collect\_item.wav

(5)仙人掌被打：damaged\_cactus.wav

(6)打擊：hit\_8.wav, hit\_10.wav

(7)鴿子攻擊：pigeon\_fireball\_disable.wav, pigeon\_fireball\_enable.wav

(8)Robot\_A攻擊：robot\_A\_attack\_1.wav

(9)機器人被打：robot\_damaged.wav

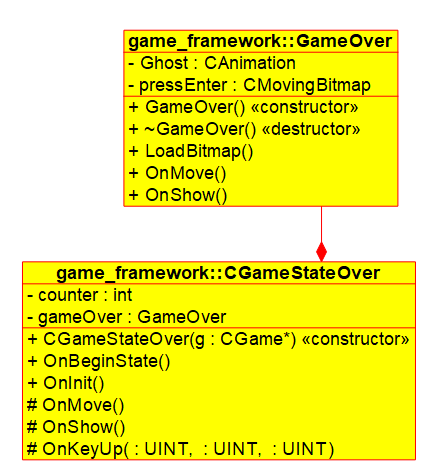
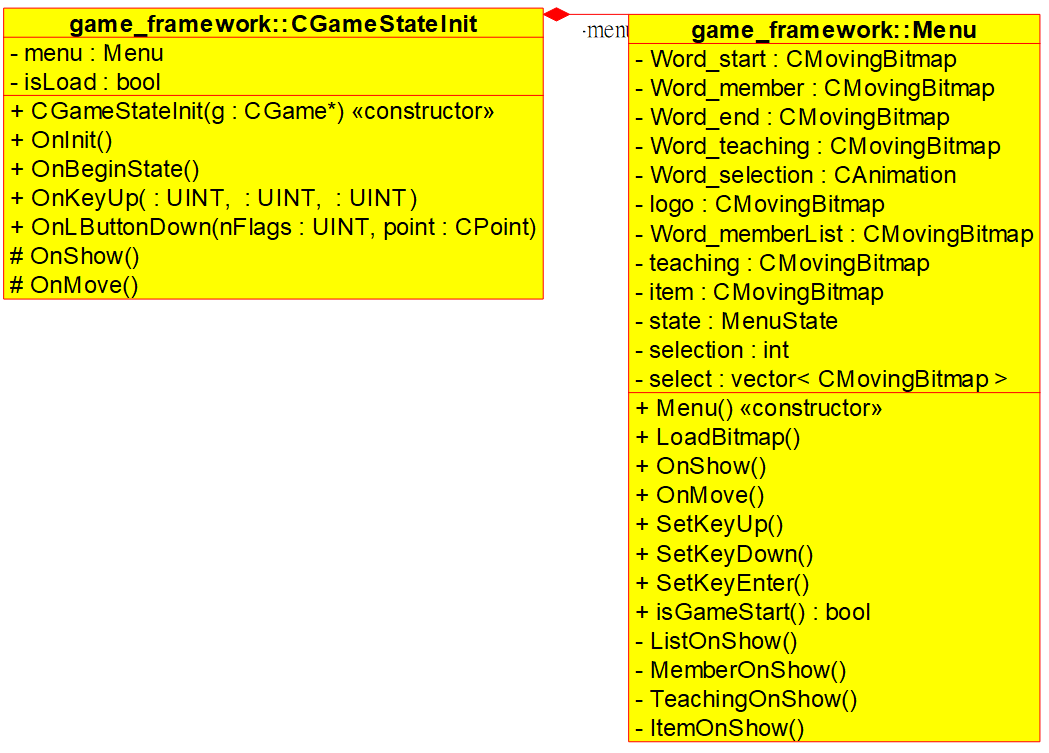
(10)機器人爆炸前：robot\_die\_bomb

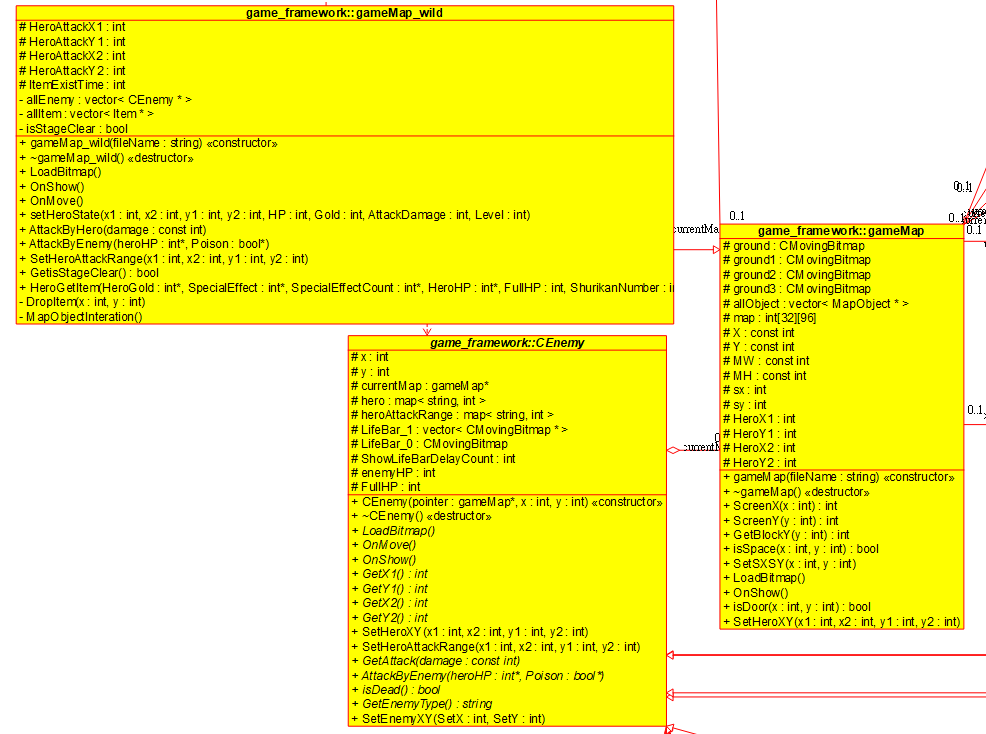
(11)關卡開始：stage\_start

(12)村莊背景音樂：stage1

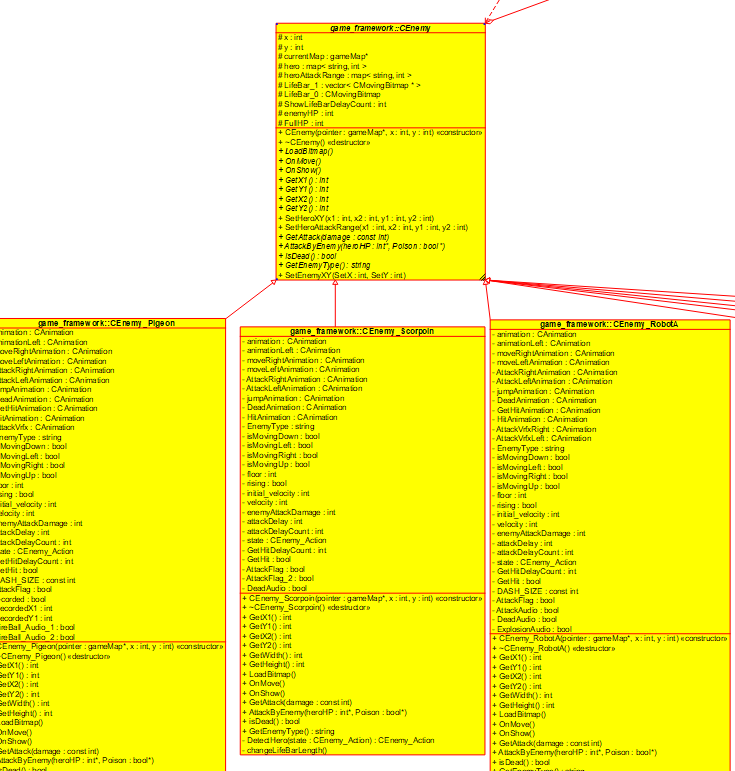
**三、程式設計**

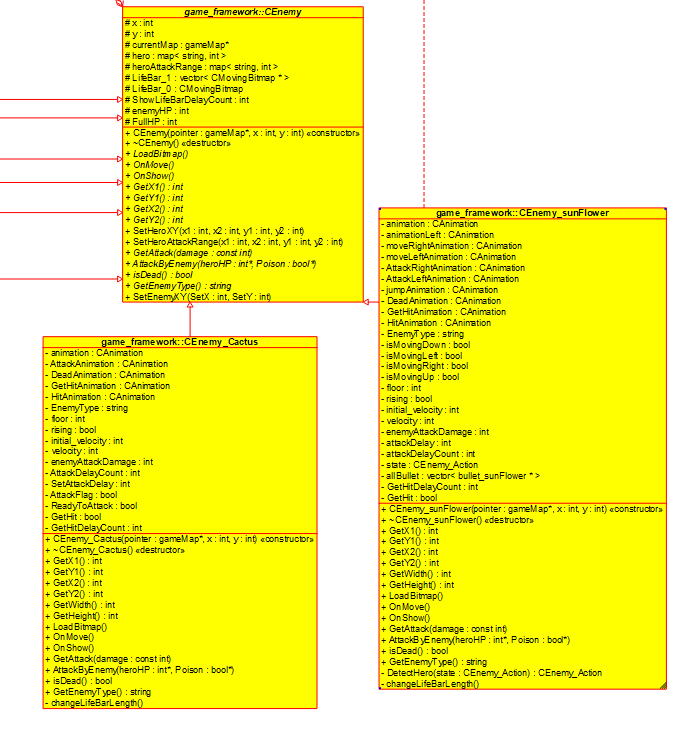
1.程式架構：

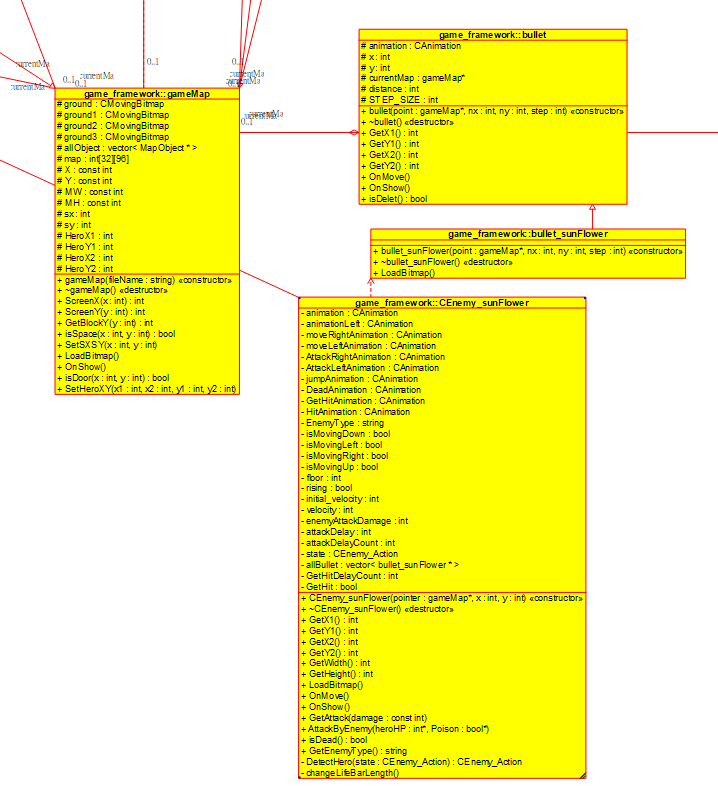


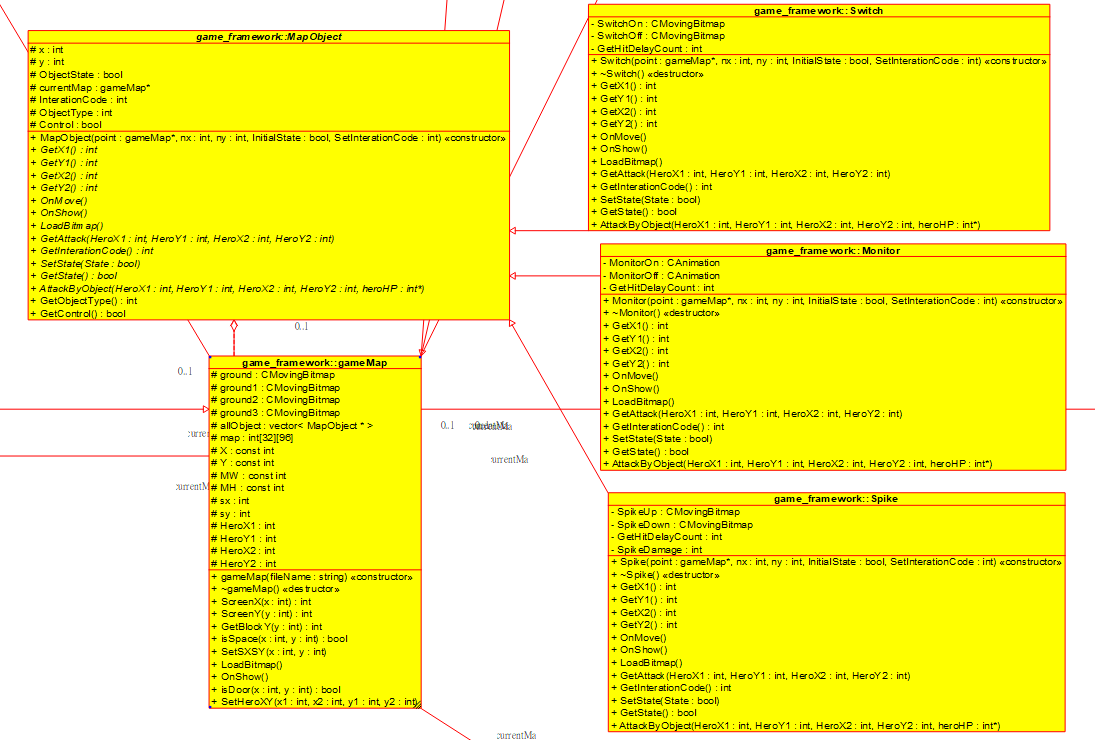


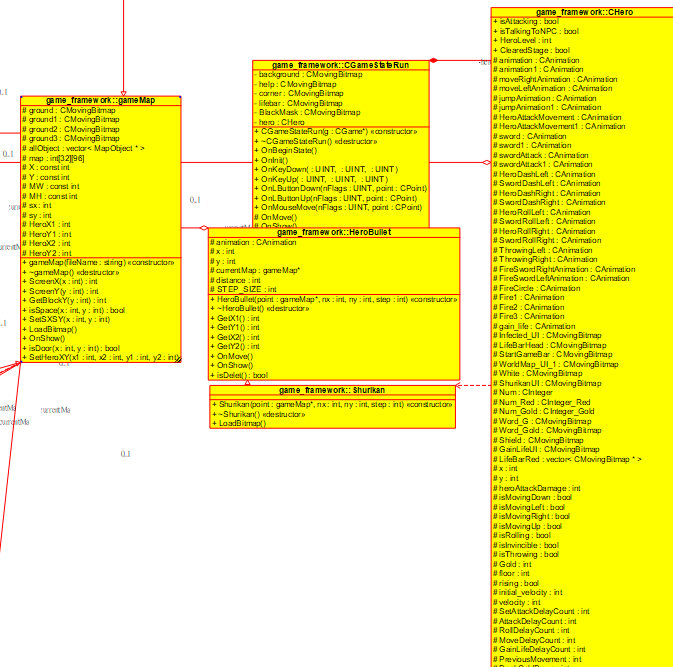


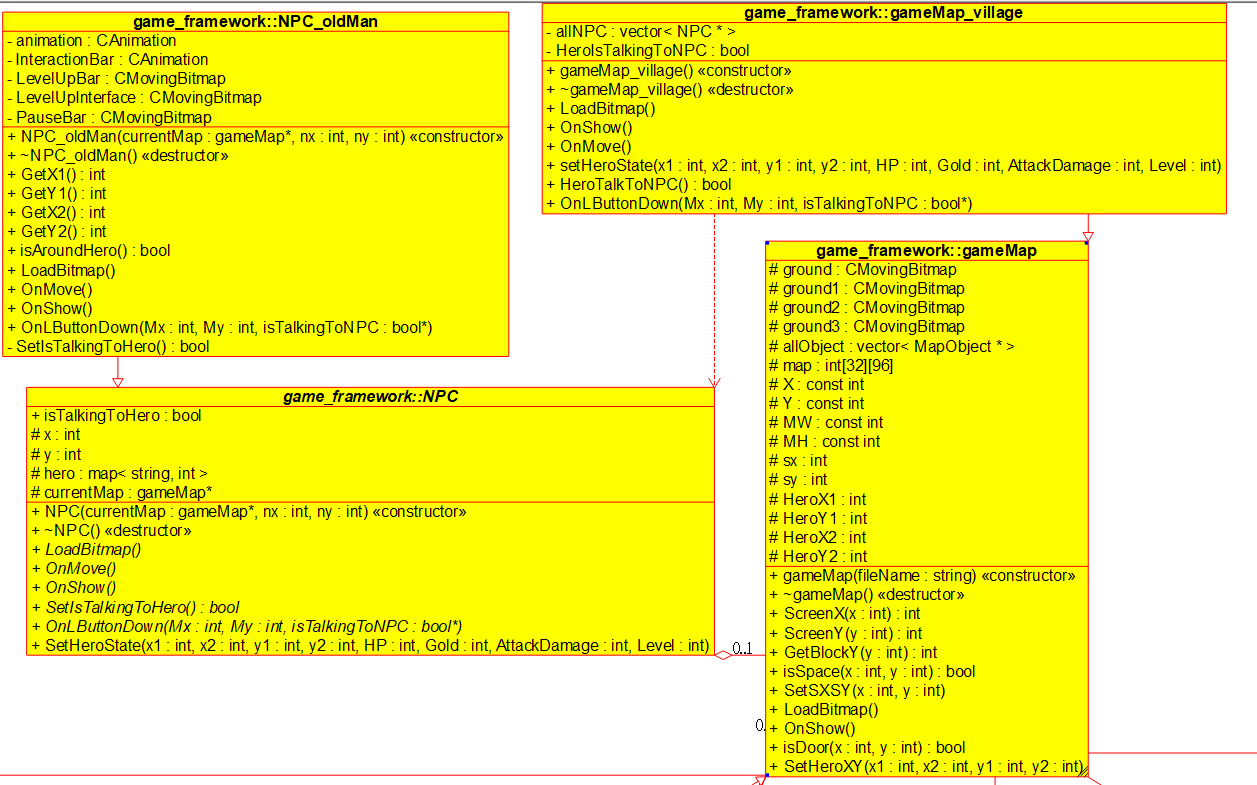














2.程式類別：

|  |  |  |  |
| --- | --- | --- | --- |
| 類別名稱 | .h檔行數 | .cpp檔行數 | 說明 |
| CEnemy | 495 | 3185 | 敵人的功能和 動作 |
| CHero | 208 | 1504 | 主角的功能和 動作 |
| gameMap | 307 | 1092 | 地圖功能 |
| GameOver | 16 | 55 | 遊戲結束畫面 |
| Menu | 41 | 171 | 初始畫面選單 |
| mygame | 130 | 384 | 遊戲控制 |
| NPC | 49 | 126 | NPC的功能 |
| **總行數** | **1246** | **6517** |  |

3.程式技術：

跳躍使用加速度，碰撞演算法、使用列舉紀錄敵人與主角的工作狀態、使用map紀錄主角的四個頂點座標。

**四、結語**

1.問題及解決方法：

(1)動畫播放問題

解決方法：透過CAnimation的isFinalBitmap()和Reset()解決。

(2)追蹤主角發動閃電的功能

解決方法：透過陣列還有delaycount去分段紀錄主角位置之後發動攻擊。

(3)敵人死亡後會掉怪物，但怪物死掉之後就delete掉了

解決方法：在gameMap裡記錄敵人死掉的位置之後產生item。

(4)主角移動時會卡進牆壁裡

解決方法：透過迴圈一次+1而不是一次就+10這樣就不會卡入牆壁裡了。

(5)有delete繼承的class，但還是造成memory leak

解決方法：因為destructor沒有使用virtual所以導致沒有真正的delete，而造成memory leak，所以把destructor加上virtual的宣告就解決了。

2.時間表(不含上課時間)：

|  |  |  |  |
| --- | --- | --- | --- |
| 週次 | 蔡育綸(小時) | 蘇佳湧(小時) | 說明 |
| 1 | 4 | 3 | 完成卷軸地圖、人物移動 |
| 2 | 6 | 6 | 在畫面上新增敵人、主角攻擊敵人 |
| 3 | 5 | 2 | 主角攻擊敵人、敵人攻擊主角 |
| 4 | 4 | 10 | 修改程式架構 |
| 5 | 12 | 6 | 主角攻擊敵人、敵人攻擊主角 |
| 6 | 3 | 3 | 完成第一關，破關後回到城鎮 |
| 7 | 3 | 7 | 改寫CEnemy繼承、NPC對話功能 |
| 8 | 10 | 5 | 完成第一關，破關後回到城鎮 |
| 9 | 18 | 10 | 敵人會射子彈，敵人會移動，增加敵人數量，修改gameMap class |
| 10 | 8 | 5 | 增加buff，增加地圖尖刺，增加地圖斜坡 |
| 11 | 7 | 4 | 讓開關有功能，把斜坡、尖刺完成 |
| 12 | 6 | 5 | 增加關卡數量 |
| 13 | 20 | 6 | 增加敵人，增加關卡 |
| 14 | 5 | 7 | 新增操作說明、收尾 |
| 合計 | 219 | 79 |  |

3.貢獻比例：

蔡育綸：50%

蘇佳湧：50%

4.自我檢核表：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 項目 | 完成否 | 無法完成的原因 |
| 1 | 解決Memory leak | ■已完成 □未完成 |  |
| 2 | 自訂遊戲icon | ■已完成 □未完成 |  |
| 3 | 全螢幕啟動 | ■已完成 □未完成 |  |
| 4 | 有About畫面 | ■已完成 □未完成 |  |
| 5 | 初始畫面說明按鍵及滑鼠之用法與密技 | ■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、行距、頁碼等格式正確 | □已完成 □未完成 |  |
| 9 | 報告封面、側邊格式正確 | □已完成 □未完成 |  |
| 10 | 報告附錄程式格式正確 | □已完成 □未完成 |  |

5.收穫：

蔡育綸：

這學期透過物件導向程式設計實習，我學到了物件與物件互動的方式，透過public的程式給其他class的程式呼叫、繼承跟多型的應用、Source Tree的版本控制、Git的運用，還有物件與物件的碰撞判斷，除錯方法有使用中斷點還有TRACE去看程式的執行結果。

蘇佳湧：

這學期透過物件導向程式設計實習，讓我更加清楚所謂的物件導向OOP是什麼概念，也對class的運用更加了解，還有對OOP的三大特性封裝、繼承、多型的運用更加熟練，還有Git的運用，和怎麼除錯，包含中斷點、使用TRACE，又或者滾回之前上傳的版本，查找bug是在哪個版本中所出現，進而縮小查找範圍。

6.心得、感想：

蔡育綸：

透過這學期的課程，學到了很多有關物件導向程式的知識還有實作，還有團隊的合作，我覺得最實用的東西應該是Source Tree還有git的應用，因為這次的實習不是自己一個人做，所以在版本控制上要下很多功夫，也多虧了有做版本控制，讓我們再找Memory leak時省下了許多時間，我們還有使用了繼承跟多型，透過繼承跟多型讓我們省下了許多時間、還有縮短了程式的長度，雖然中間為了程式的架構花了好幾個禮拜的時間去修改，但是也學到了如何讓class和class之間有更好的互動。透過與隊友的合作，也可以看一下別人寫程式的風格，進而去模仿別人的寫法，並且修正自己寫得不好的地方，下次再寫同樣的程式就知道要怎麼寫了。

蘇佳湧：

透過這學期的課程，讓我了解到原來程式一旦有點規模之後，寫Code的方式就和之前完全不一樣了，在寫Code之前的規劃非常重要，因為沒有良好的規劃，在之後寫程式時，很容易造成問題，例如class和class之間該怎麼互動，如果沒有先訂好函式，就會造成亂Call亂寫的問題出現，程式一團亂，雖然功能有出來，但是不好擴充，不好偵錯，也不好維護，我們也是在一開始，因為不太確定遊戲內容，和主要功能，所以程式沒有規劃得很完善，導致修bug很困難，還會再一些奇怪的地方，出現一些不該出現的函式，我們也因此花費很多的時間，來重新修改程式架構和程式。

**附錄**

|  |
| --- |
| NPC.h |
| namespace game\_framework {  class gameMap;  class NPC  {  public:  NPC(gameMap\* currentMap, int nx, int ny);  virtual ~NPC();  virtual void LoadBitmap() = 0; // 載入圖形  virtual void OnMove() = 0; // 移動NPC  virtual void OnShow() = 0; // 將NPC圖形貼到畫面  virtual bool SetIsTalkingToHero() = 0;  virtual void OnLButtonDown(int Mx, int My, bool \*isTalkingToNPC) = 0;  bool isTalkingToHero = false;  //virtual void TouchedByHero(int x1, int x2, int y1, int y2);  void SetHeroState(int x1, int x2, int y1, int y2, int HP, int Gold, int AttackDamage, int Level);  protected:  int x, y; // NPC左上角座標  map<string, int> hero;  gameMap\* currentMap;  };  class NPC\_oldMan : public NPC  {  public:  NPC\_oldMan(gameMap\* currentMap, int nx, int ny);  ~NPC\_oldMan();  int GetX1();  int GetY1();  int GetX2();  int GetY2();  bool isAroundHero();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動NPC  void OnShow(); // 將NPC圖形貼到畫面  void OnLButtonDown(int Mx, int My, bool \*isTalkingToNPC);  private:  CAnimation animation; // NPC的動畫  CAnimation InteractionBar; //互動圖  bool SetIsTalkingToHero();  CMovingBitmap LevelUpBar;  CMovingBitmap LevelUpInterface;  CMovingBitmap PauseBar;  };  } |
| NPC.cpp |
| #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "gamelib.h"  #include "gameMap.h"  #include "NPC.h"  namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // NPC: NPC base class  /////////////////////////////////////////////////////////////////////////////  NPC::NPC(gameMap\* pointer,int x, int y) : currentMap(pointer), x(x), y(y) {}  NPC::~NPC() {}  void NPC::SetHeroState(int x1, int x2, int y1, int y2, int HP, int Gold, int AttackDamage, int Level)  {  hero["x1"] = x1;  hero["x2"] = x2;  hero["y1"] = y1;  hero["y2"] = y2;  hero["HP"] = HP;  hero["Gold"] = Gold;  hero["AttackDamage"] = AttackDamage;  hero["HeroLevel"] = Level;  }  /////////////////////////////////////////////////////////////////////////////  // NPC\_oldMan: NPC old man class  /////////////////////////////////////////////////////////////////////////////  NPC\_oldMan::NPC\_oldMan(gameMap\* pointer, int x, int y) : NPC(pointer, x, y) {}  NPC\_oldMan::~NPC\_oldMan() {}  int NPC\_oldMan::GetX1()  {  return x;  }  int NPC\_oldMan::GetY1()  {  return y;  }  int NPC\_oldMan::GetX2()  {  return x + animation.Width();  }  int NPC\_oldMan::GetY2()  {  return y + animation.Height();  }  void NPC\_oldMan::LoadBitmap()  {  animation.AddBitmap(IDB\_OLD\_MAN\_1, RGB(255, 0, 0));  animation.AddBitmap(IDB\_OLD\_MAN\_2, RGB(255, 0, 0));  animation.AddBitmap(IDB\_OLD\_MAN\_3, RGB(255, 0, 0));  InteractionBar.AddBitmap(IDB\_TALK\_BAR\_1, RGB(255, 255, 255));  InteractionBar.AddBitmap(IDB\_TALK\_BAR\_2, RGB(255, 255, 255));  InteractionBar.AddBitmap(IDB\_TALK\_BAR\_3, RGB(255, 255, 255));  InteractionBar.AddBitmap(IDB\_TALK\_BAR\_4, RGB(255, 255, 255));  LevelUpBar.LoadBitmap(IDB\_UI\_LEVEL\_UP\_BAR);  PauseBar.LoadBitmap(IDB\_UI\_QUIT);  LevelUpInterface.LoadBitmap(IDB\_UI\_LEVEL\_UP\_INTERFACE);  }  void NPC\_oldMan::OnMove()  {  animation.OnMove();  InteractionBar.OnMove();  }  void NPC\_oldMan::OnShow()  {  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  if (isAroundHero())  {  InteractionBar.SetTopLeft(currentMap->ScreenX(x + 80), currentMap->ScreenY(y));  InteractionBar.OnShow();  }  if (isTalkingToHero)  {  LevelUpInterface.SetTopLeft(currentMap->ScreenX(hero["x1"]+50), currentMap->ScreenY(hero["y1"]-190));  LevelUpInterface.ShowBitmap();  }  }  bool NPC\_oldMan::isAroundHero()  {  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()))  return true;  else  return false;  }  bool NPC\_oldMan::SetIsTalkingToHero()  {  if (isAroundHero())  {  isTalkingToHero = true;  return true;  }  return false;  }  void NPC\_oldMan::OnLButtonDown(int Mx, int My, bool \*isTalkingToNPC)  {  if ((Mx <= 630) && (My <= 50) && (Mx >= 609) && (My >= 28))  {  isTalkingToHero = false; //右上角xx  \*isTalkingToNPC = false;  }  if ((Mx <= 625) && (My <= 216) && (Mx >= 561) && (My >= 199))  {  isTalkingToHero = false; //cancel  \*isTalkingToNPC = false;  }  }  } |
| mygame.h |
| /\*  \* mygame.h: 本檔案儲遊戲本身的class的interface  \* Copyright (C) 2002-2008 Woei-Kae Chen <wkc@csie.ntut.edu.tw>  \*  \* This file is part of game, a free game development framework for windows.  \*  \* game is free software; you can redistribute it and/or modify  \* it under the terms of the GNU General Public License as published by  \* the Free Software Foundation; either version 2 of the License, or  \* (at your option) any later version.  \*  \* game is distributed in the hope that it will be useful,  \* but WITHOUT ANY WARRANTY; without even the implied warranty of  \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  \* GNU General Public License for more details.  \*  \* You should have received a copy of the GNU General Public License  \* along with this program; if not, write to the Free Software  \* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  \*  \* 2004-03-02 V4.0  \* 1. Add CGameStateInit, CGameStateRun, and CGameStateOver to  \* demonstrate the use of states.  \* 2005-09-13  \* Rewrite the codes for CBall and CEraser.  \* 2005-09-20 V4.2Beta1.  \* 2005-09-29 V4.2Beta2.  \* 2006-02-08 V4.2  \* 1. Rename OnInitialUpdate() -> OnInit().  \* 2. Replace AUDIO\_CANYON as AUDIO\_NTUT.  \* 3. Add help bitmap to CGameStateRun.  \* 2006-09-09 V4.3  \* 1. Rename Move() and Show() as OnMove and OnShow() to emphasize that they are  \* event driven.  \* 2008-02-15 V4.4  \* 1. Add namespace game\_framework.  \* 2. Replace the demonstration of animation as a new bouncing ball.  \* 3. Use ShowInitProgress(percent) to display loading progress.  \*/  //#include "CHero.h"  enum AUDIO\_ID { // 定義各種音效的編號  AUDIO\_DING, // 0  AUDIO\_LAKE, // 1  AUDIO\_NTUT, // 2  AUDIO\_MENU, // 3  AUDIO\_SUPERCAR, // 4  AUDIO\_GETITEM, // 5  AUDIO\_GAMESTART, // 6  AUDIO\_STAGECLEAR, // 7  AUDIO\_STAGE1, // 8  AUDIO\_CACTUSGETHIT, // 9  AUDIO\_CACTUSATTACK, // 10  AUDIO\_HIT\_8, // 11  AUDIO\_HIT\_10, // 12  AUDIO\_ROBOT\_DAMAGE, // 13  AUDIO\_ROBOT\_A\_ATTACK, // 14  AUDIO\_ROBOT\_DIE, // 15  AUDIO\_CLOUD\_ATTACK, // 16  AUDIO\_PIGEON\_ATTACK\_1, // 17  AUDIO\_PIGEON\_ATTACK\_2, // 18  AUDIO\_BOMB, // 19  AUDIO\_GAMEOVER // 20  };  namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // Constants  /////////////////////////////////////////////////////////////////////////////  /////////////////////////////////////////////////////////////////////////////  // 這個class為遊戲的遊戲開頭畫面物件  // 每個Member function的Implementation都要弄懂  /////////////////////////////////////////////////////////////////////////////  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  void OnMove();  private:  //CMovingBitmap logo; // csie的logo  Menu menu;  bool isLoad;  };  /////////////////////////////////////////////////////////////////////////////  // 這個class為遊戲的遊戲執行物件，主要的遊戲程式都在這裡  // 每個Member function的Implementation都要弄懂  /////////////////////////////////////////////////////////////////////////////  class CGameStateRun : public CGameState {  public:  CGameStateRun(CGame \*g);  ~CGameStateRun();  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);  void OnLButtonDown(UINT nFlags, CPoint point); // 處理滑鼠的動作  void OnLButtonUp(UINT nFlags, CPoint point); // 處理滑鼠的動作  void OnMouseMove(UINT nFlags, CPoint point); // 處理滑鼠的動作  protected:  void OnMove(); // 移動遊戲元素  void OnShow(); // 顯示這個狀態的遊戲畫面  private:  CMovingBitmap background; // 背景圖  CMovingBitmap help; // 說明圖  CMovingBitmap corner; // 角落圖  CMovingBitmap lifebar; // 英雄血量  CMovingBitmap BlackMask; // 半透明效果  CHero hero; // 英雄  };  /////////////////////////////////////////////////////////////////////////////  // 這個class為遊戲的結束狀態(Game Over)  // 每個Member function的Implementation都要弄懂  /////////////////////////////////////////////////////////////////////////////  class CGameStateOver : public CGameState {  public:  CGameStateOver(CGame \*g);  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  void OnKeyUp(UINT, UINT, UINT);  private:  int counter; // 倒數之計數器  GameOver gameOver;  };  } |
| mygame.cpp |
| /\*  \* mygame.cpp: 本檔案儲遊戲本身的class的implementation  \* Copyright (C) 2002-2008 Woei-Kae Chen <wkc@csie.ntut.edu.tw>  \*  \* This file is part of game, a free game development framework for windows.  \*  \* game is free software; you can redistribute it and/or modify  \* it under the terms of the GNU General Public License as published by  \* the Free Software Foundation; either version 2 of the License, or  \* (at your option) any later version.  \*  \* game is distributed in the hope that it will be useful,  \* but WITHOUT ANY WARRANTY; without even the implied warranty of  \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  \* GNU General Public License for more details.  \*  \* You should have received a copy of the GNU General Public License  \* along with this program; if not, write to the Free Software  \* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  \*  \* History:  \* 2002-03-04 V3.1  \* Add codes to demostrate the use of CMovingBitmap::ShowBitmap(CMovingBitmap &).  \* 2004-03-02 V4.0  \* 1. Add CGameStateInit, CGameStateRun, and CGameStateOver to  \* demonstrate the use of states.  \* 2. Demo the use of CInteger in CGameStateRun.  \* 2005-09-13  \* Rewrite the codes for CBall and CEraser.  \* 2005-09-20 V4.2Beta1.  \* 2005-09-29 V4.2Beta2.  \* 1. Add codes to display IDC\_GAMECURSOR in GameStateRun.  \* 2006-02-08 V4.2  \* 1. Revise sample screens to display in English only.  \* 2. Add code in CGameStateInit to demo the use of PostQuitMessage().  \* 3. Rename OnInitialUpdate() -> OnInit().  \* 4. Fix the bug that OnBeginState() of GameStateInit is not called.  \* 5. Replace AUDIO\_CANYON as AUDIO\_NTUT.  \* 6. Add help bitmap to CGameStateRun.  \* 2006-09-09 V4.3  \* 1. Rename Move() and Show() as OnMove and OnShow() to emphasize that they are  \* event driven.  \* 2006-12-30  \* 1. Bug fix: fix a memory leak problem by replacing PostQuitMessage(0) as  \* PostMessage(AfxGetMainWnd()->m\_hWnd, WM\_CLOSE,0,0).  \* 2008-02-15 V4.4  \* 1. Add namespace game\_framework.  \* 2. Replace the demonstration of animation as a new bouncing ball.  \* 3. Use ShowInitProgress(percent) to display loading progress.  \* 2010-03-23 V4.6  \* 1. Demo MP3 support: use lake.mp3 to replace lake.wav.  \*/  #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "audio.h"  #include "gamelib.h"  #include "Counter.h"  #include "NPC.h"  #include "CEnemy.h"  #include "gameMap.h"  #include "CHero.h"  #include "Menu.h"  #include "GameOver.h"  #include "mygame.h"  namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // 這個class為遊戲的遊戲開頭畫面物件  /////////////////////////////////////////////////////////////////////////////  CGameStateInit::CGameStateInit(CGame \*g)  : CGameState(g)  {  isLoad = false;  }  void CGameStateInit::OnInit()  {  //  // 當圖很多時，OnInit載入所有的圖要花很多時間。為避免玩遊戲的人  // 等的不耐煩，遊戲會出現「Loading ...」，顯示Loading的進度。  //  ShowInitProgress(0); // 一開始的loading進度為0%  //  // 開始載入資料  //  menu.LoadBitmap();  //Sleep(300); // 放慢，以便看清楚進度，實際遊戲請刪除此Sleep  //  // 此OnInit動作會接到CGameStaterRun::OnInit()，所以進度還沒到100%  //  CAudio::Instance()->Load(AUDIO\_MENU, "sounds\\menu.wav"); // 載入編號3的聲音load.mp3  isLoad = true;  CAudio::Instance()->Play(AUDIO\_MENU, false);  }  void CGameStateInit::OnBeginState()  {  if(isLoad)  CAudio::Instance()->Play(AUDIO\_MENU, false); // 撥放 mp3  }  void CGameStateInit::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)  {  const char KEY\_ESC = 27;  const char KEY\_ENTER = 13;  const char KEY\_F1 = 0x70; // keyboard F1  const char KEY\_UP = 0x26; // keyboard上箭頭  const char KEY\_DOWN = 0x28; // keyboard下箭頭  if (nChar == KEY\_UP)  menu.SetKeyUp();  if (nChar == KEY\_DOWN)  menu.SetKeyDown();  if (nChar == KEY\_F1)  {  CAudio::Instance()->Stop(AUDIO\_MENU);  GotoGameState(GAME\_STATE\_OVER);  }  if (nChar == KEY\_ENTER)  {  menu.SetKeyEnter();  if (menu.isGameStart())  {  CAudio::Instance()->Stop(AUDIO\_MENU);  GotoGameState(GAME\_STATE\_RUN);  }  }  }  void CGameStateInit::OnLButtonDown(UINT nFlags, CPoint point)  {  }  void CGameStateInit::OnMove()  {  menu.OnMove();  }  void CGameStateInit::OnShow()  {  //  // 貼上logo  //  menu.OnShow();  //  // Demo螢幕字型的使用，不過開發時請盡量避免直接使用字型，改用CMovingBitmap比較好  //  }  /////////////////////////////////////////////////////////////////////////////  // 這個class為遊戲的結束狀態(Game Over)  /////////////////////////////////////////////////////////////////////////////  CGameStateOver::CGameStateOver(CGame \*g)  : CGameState(g)  {  }  void CGameStateOver::OnMove()  {  gameOver.OnMove();  }  void CGameStateOver::OnBeginState()  {  CAudio::Instance()->Stop(AUDIO\_STAGE1); // 撥放 WAVE  CAudio::Instance()->Stop(AUDIO\_DING); // 撥放 WAVE  CAudio::Instance()->Play(AUDIO\_GAMEOVER);  counter = 30 \* 5; // 5 seconds  }  void CGameStateOver::OnInit()  {  //  // 當圖很多時，OnInit載入所有的圖要花很多時間。為避免玩遊戲的人  // 等的不耐煩，遊戲會出現「Loading ...」，顯示Loading的進度。  //  ShowInitProgress(66); // 接個前一個狀態的進度，此處進度視為66%  //  // 開始載入資料  //  gameOver.LoadBitmap();  CAudio::Instance()->Load(AUDIO\_GAMEOVER, "sounds\\game\_over.mp3");  //Sleep(300); // 放慢，以便看清楚進度，實際遊戲請刪除此Sleep  //  // 最終進度為100%  //  ShowInitProgress(100);  }  void CGameStateOver::OnShow()  {  gameOver.OnShow();  }  void CGameStateOver::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)  {  const char KEY\_ENTER = 13;  if (nChar == KEY\_ENTER)  {  CAudio::Instance()->Stop(AUDIO\_GAMEOVER);  GotoGameState(GAME\_STATE\_INIT);  }  }  /////////////////////////////////////////////////////////////////////////////  // 這個class為遊戲的遊戲執行物件，主要的遊戲程式都在這裡  /////////////////////////////////////////////////////////////////////////////  CGameStateRun::CGameStateRun(CGame \*g)  : CGameState(g)  {  }  CGameStateRun::~CGameStateRun()  {  }  void CGameStateRun::OnBeginState()  {  const int BALL\_GAP = 90;  const int BALL\_XY\_OFFSET = 45;  const int BALL\_PER\_ROW = 7;  const int HITS\_LEFT = 10;  const int HERO\_LIFEBAR\_X = 0;  const int HERO\_LIFEBAR\_Y = 0;  const int BACKGROUND\_X = 0;  const int ANIMATION\_SPEED = 15;  hero.Initialize();  background.SetTopLeft(BACKGROUND\_X,0); // 設定背景的起始座標  help.SetTopLeft(0, SIZE\_Y - help.Height()); // 設定說明圖的起始座標  CAudio::Instance()->Play(AUDIO\_STAGE1, true); // 撥放 WAVE  CAudio::Instance()->Play(AUDIO\_DING, false); // 撥放 WAVE  CAudio::Instance()->Stop(AUDIO\_MENU);  }  void CGameStateRun::OnMove() // 移動遊戲元素  {  hero.OnMove();  if (!hero.isAlive())  {  GotoGameState(GAME\_STATE\_OVER);  }  }  void CGameStateRun::OnInit() // 遊戲的初值及圖形設定  {  //  // 當圖很多時，OnInit載入所有的圖要花很多時間。為避免玩遊戲的人  // 等的不耐煩，遊戲會出現「Loading ...」，顯示Loading的進度。  //  ShowInitProgress(33); // 接個前一個狀態的進度，此處進度視為33%  //  // 開始載入資料  //  hero.LoadBitmap();  background.LoadBitmap(IDB\_MAPBACKGROUND); // 載入背景的圖形  //  // 完成部分Loading動作，提高進度  //  ShowInitProgress(50);  //  // 繼續載入其他資料  //  help.LoadBitmap(IDB\_HELP,RGB(255,255,255)); // 載入說明的圖形  CAudio::Instance()->Load(AUDIO\_DING, "sounds\\ding.wav"); // 載入編號0的聲音ding.wav  CAudio::Instance()->Load(AUDIO\_LAKE, "sounds\\lake.mp3"); // 載入編號1的聲音lake.mp3  CAudio::Instance()->Load(AUDIO\_GETITEM, "sounds\\collect\_item.wav"); // 載入編號5的聲音getitem  CAudio::Instance()->Load(AUDIO\_SUPERCAR, "sounds\\super\_car.mp3"); // 載入編號4的聲音ntut.mid  CAudio::Instance()->Load(AUDIO\_GAMESTART, "sounds\\stage\_start.wav");  CAudio::Instance()->Load(AUDIO\_STAGECLEAR, "sounds\\stage\_clear.wav");  CAudio::Instance()->Load(AUDIO\_STAGE1, "sounds\\stage1.wav");  CAudio::Instance()->Load(AUDIO\_CACTUSGETHIT, "sounds\\damaged\_cactus.wav");  CAudio::Instance()->Load(AUDIO\_CACTUSATTACK, "sounds\\cactus\_attack.wav");  CAudio::Instance()->Load(AUDIO\_HIT\_8, "sounds\\hit\_8.wav");  CAudio::Instance()->Load(AUDIO\_HIT\_10, "sounds\\hit\_10.wav");  CAudio::Instance()->Load(AUDIO\_ROBOT\_DAMAGE, "sounds\\robot\_damaged.wav");  CAudio::Instance()->Load(AUDIO\_ROBOT\_A\_ATTACK, "sounds\\robot\_A\_attack\_1.wav");  CAudio::Instance()->Load(AUDIO\_ROBOT\_DIE, "sounds\\robot\_die\_bomb.wav");  CAudio::Instance()->Load(AUDIO\_CLOUD\_ATTACK, "sounds\\cloudy\_cloud\_attack.wav");  CAudio::Instance()->Load(AUDIO\_PIGEON\_ATTACK\_1, "sounds\\pigeon\_fireball\_enable.wav");  CAudio::Instance()->Load(AUDIO\_PIGEON\_ATTACK\_2, "sounds\\pigeon\_fireball\_disable.wav");  CAudio::Instance()->Load(AUDIO\_BOMB, "sounds\\bomb.wav");  //  // 此OnInit動作會接到CGameStaterOver::OnInit()，所以進度還沒到100%  //  }  void CGameStateRun::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\_E = 0x45; // keyboard E  const char KEY\_X = 0x58; // keyboard X  const char KEY\_F1 = 0x70; // keyboard F1  const char KEY\_F2 = 0x71; // keyboard F2  const char KEY\_F3 = 0x72; // keyboard F3  const char KEY\_Z = 0x5A;  const char KEY\_C = 0x43;  const char KEY\_D = 0x44;  if(nChar == KEY\_LEFT)  hero.SetMovingLeft(true);  if (nChar == KEY\_RIGHT) //英雄不能取消翻滾動作  hero.SetMovingRight(true);  if (nChar == KEY\_UP)  hero.SetMovingUp(true);  if (nChar == KEY\_DOWN)  hero.SetMovingDown(true);  if (nChar == KEY\_X)  hero.SetHeroAttack(true);  if (nChar == KEY\_Z)  hero.SetHeroRoll(true);  if (nChar == KEY\_E)  hero.SetTalkingToNPC(true);  if (nChar == KEY\_C)  hero.SetHeroThrow(true);  if (nChar == KEY\_D)  hero.AddHeroGold();  }  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\_X = 0x58; // keyboard X  const char KEY\_E = 0x45; // keyboard E  if (nChar == KEY\_LEFT)  {  hero.SetMovingLeft(false);  }  if (nChar == KEY\_RIGHT)  {  hero.SetMovingRight(false);  }  if (nChar == KEY\_DOWN)  hero.SetMovingDown(false);  }  void CGameStateRun::OnLButtonDown(UINT nFlags, CPoint point) // 處理滑鼠的動作  {  hero.OnLButtonDown(point.x, point.y);  }  void CGameStateRun::OnLButtonUp(UINT nFlags, CPoint point) // 處理滑鼠的動作  {  }  void CGameStateRun::OnMouseMove(UINT nFlags, CPoint point) // 處理滑鼠的動作  {  // 沒事。如果需要處理滑鼠移動的話，寫code在這裡  }  void CGameStateRun::OnShow()  {  //  // 注意：Show裡面千萬不要移動任何物件的座標，移動座標的工作應由Move做才對，  // 否則當視窗重新繪圖時(OnDraw)，物件就會移動，看起來會很怪。換個術語  // 說，Move負責MVC中的Model，Show負責View，而View不應更動Model。  background.ShowBitmap();// 貼上背景圖  help.ShowBitmap(); // 貼上說明圖  hero.OnShow(); // 貼上英雄  }  } |
| Menu.h |
| namespace game\_framework {  enum MenuState  {  LIST,  MEMBER,  TEACHING,  ITEM  };  class Menu  {  public:  Menu();  void LoadBitmap(); // 載入圖形  void OnShow(); // 顯示遊戲畫面  void OnMove(); // 移動元素  void SetKeyUp(); // 按下上鍵  void SetKeyDown(); // 按下下鍵  void SetKeyEnter(); // 按下Enter鍵  bool isGameStart(); // 回傳遊戲是否開始  private:  CMovingBitmap Word\_start; // 字體"開始遊戲"  CMovingBitmap Word\_member; // 字體"人員名單"  CMovingBitmap Word\_end; // 字體"離開遊戲"  CMovingBitmap Word\_teaching; // 字體"操作教學"  CAnimation Word\_selection; // 選擇符號  CMovingBitmap logo; // 遊戲 LOGO  CMovingBitmap Word\_memberList; // 人員名單畫面  CMovingBitmap teaching; // 按鍵教學畫面  CMovingBitmap item; // 物品介紹畫面  MenuState state; // 紀錄狀態  int selection; // 紀錄選擇的index  void ListOnShow(); // 顯示按鍵教學畫面  void MemberOnShow(); // 顯示人員名單畫面  void TeachingOnShow(); // 顯示按鍵教學畫面  void ItemOnShow(); // 顯示物品介紹畫面  vector<CMovingBitmap> select; // 儲存全部選項  };  } |
| Menu.cpp |
| #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "gamelib.h"  #include "Counter.h"  #include "Menu.h"  namespace game\_framework {  Menu::Menu()  {  state = LIST;  selection = 0;  Word\_selection.SetDelayCount(3);  }  void Menu::LoadBitmap()  {  logo.LoadBitmap(IDB\_GREATSWORDLOGO, RGB(255, 255, 255));  Word\_start.LoadBitmap(IDB\_WORD\_START);  Word\_member.LoadBitmap(IDB\_WORD\_MEMBER);  Word\_teaching.LoadBitmap(IDB\_WORD\_TEACHING);  Word\_end.LoadBitmap(IDB\_WORD\_END);  Word\_memberList.LoadBitmap(IDB\_WORD\_MEMBER\_LIST);  teaching.LoadBitmap(IDB\_KEY);  item.LoadBitmap(IDB\_ALL\_ITEM);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_1);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_2);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_3);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_4);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_5);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_4);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_3);  Word\_selection.AddBitmap(IDB\_WORD\_SELECTION\_2);  select.push\_back(Word\_start);  select.push\_back(Word\_teaching);  select.push\_back(Word\_member);  select.push\_back(Word\_end);  }  void Menu::OnMove()  {  Word\_selection.OnMove();  }  void Menu::ListOnShow()  {  int x = (SIZE\_X - select[0].Width()) / 2;  int y = 5;  logo.SetTopLeft((SIZE\_X - logo.Width()) / 2, y);  logo.ShowBitmap();  y += logo.Height() + 20;  for (unsigned int i = 0; i < select.size(); i++)  {  if ((int)i == selection)  {  Word\_selection.SetTopLeft(x - Word\_selection.Width() - 10, y);  Word\_selection.OnShow();  }  select[i].SetTopLeft(x, y);  select[i].ShowBitmap();  y += select[i].Height() + 10;  }  }  void Menu::MemberOnShow()  {  Word\_memberList.SetTopLeft((SIZE\_X - Word\_memberList.Width()) / 2, (SIZE\_Y - Word\_memberList.Height()) / 2);  Word\_memberList.ShowBitmap();  }  void Menu::TeachingOnShow()  {  teaching.SetTopLeft(0, 0);  teaching.ShowBitmap();  }  void Menu::ItemOnShow()  {  item.SetTopLeft(0, 0);  item.ShowBitmap();  }  void Menu::OnShow()  {  switch (state)  {  case LIST:  ListOnShow();  break;  case MEMBER:  MemberOnShow();  break;  case TEACHING:  TeachingOnShow();  break;  case ITEM:  ItemOnShow();  break;  default:  break;  }  }  void Menu::SetKeyUp()  {  if (state == LIST && selection == 0)  selection = select.size() - 1;  else  selection--;  }  void Menu::SetKeyDown()  {  if (state ==LIST && selection == select.size() - 1)  selection = 0;  else  selection++;  }  void Menu::SetKeyEnter()  {  switch (state)  {  case game\_framework::LIST:  switch (selection)  {  case 1:  state = TEACHING;  break;  case 2:  state = MEMBER;  break;  case 3:  PostMessage(AfxGetMainWnd()->m\_hWnd, WM\_CLOSE, 0, 0);  break;  default:  break;  }  break;  case game\_framework::MEMBER:  state = LIST;  break;  case game\_framework::TEACHING:  state = ITEM;  break;  case game\_framework::ITEM:  state = LIST;  break;  default:  break;  }  }  bool Menu::isGameStart()  {  if (state == LIST && selection == 0)  {  return true;  }  return false;  }  } |
| GameOver.h |
| namespace game\_framework {  class GameOver  {  public:  GameOver();  ~GameOver();  void LoadBitmap();  void OnMove();  void OnShow();  private:  CAnimation Ghost;  CMovingBitmap pressEnter;  };  } |
| Game.cpp |
| #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "gamelib.h"  #include "GameOver.h"  namespace game\_framework {  GameOver::GameOver()  {  Ghost.SetDelayCount(4);  }  GameOver::~GameOver()  {  }  void GameOver::LoadBitmap()  {  Ghost.AddBitmap(IDB\_GHOST\_1);  Ghost.AddBitmap(IDB\_GHOST\_2);  Ghost.AddBitmap(IDB\_GHOST\_3);  Ghost.AddBitmap(IDB\_GHOST\_4);  Ghost.AddBitmap(IDB\_GHOST\_5);  Ghost.AddBitmap(IDB\_GHOST\_6);  Ghost.AddBitmap(IDB\_GHOST\_7);  Ghost.AddBitmap(IDB\_GHOST\_8);  Ghost.AddBitmap(IDB\_GHOST\_9);  Ghost.AddBitmap(IDB\_GHOST\_10);  Ghost.AddBitmap(IDB\_GHOST\_11);  Ghost.AddBitmap(IDB\_GHOST\_12);  Ghost.AddBitmap(IDB\_GHOST\_13);  Ghost.AddBitmap(IDB\_GHOST\_14);  Ghost.AddBitmap(IDB\_GHOST\_15);  Ghost.AddBitmap(IDB\_GHOST\_16);  Ghost.AddBitmap(IDB\_GHOST\_17);  Ghost.AddBitmap(IDB\_GHOST\_18);  Ghost.AddBitmap(IDB\_GHOST\_19);  Ghost.AddBitmap(IDB\_GHOST\_20);  pressEnter.LoadBitmap(IDB\_WORD\_PRESS\_ENTER);  }  void GameOver::OnMove()  {  Ghost.OnMove();  }  void GameOver::OnShow()  {  Ghost.SetTopLeft((SIZE\_X - Ghost.Width()) / 2, (SIZE\_Y - Ghost.Height()) / 2);  Ghost.OnShow();  pressEnter.SetTopLeft((SIZE\_X - pressEnter.Width()) / 2, (SIZE\_Y - 100));  pressEnter.ShowBitmap();  }  } |
| gameMap.h |
| namespace game\_framework {  class NPC;  class CEnemy;  class Item;  class MapObject;  class gameMap {  public:  gameMap(string fileName);  ~gameMap();  int ScreenX(int x);  int ScreenY(int y);  int GetBlockY(int y);  bool isSpace(int x, int y); // 判斷是否為空氣，是就回傳True  void SetSXSY(int x, int y); // 設定(sx, sy)為螢幕(的左上角)在地圖上的點座標  void LoadBitmap(); // 載入地圖  void OnShow(); // 顯示地圖  bool isDoor(int x, int y);  void SetHeroXY(int x1, int x2, int y1, int y2); // 設定英雄位置  protected:  CMovingBitmap ground; //建立草地圖案  CMovingBitmap ground1; //建立土地圖案  CMovingBitmap ground2; //建立斜坡1圖案  CMovingBitmap ground3; //建立斜坡2圖案  vector<MapObject\*> allObject;  int map[32][96]; //建立一個地圖矩陣的index;  const int X, Y; //大地圖左上角x,y座標  const int MW, MH; //每張小地圖的寬高度  int sx, sy; //(sx, sy)為螢幕(的左上角)在地圖上的點座標  int HeroX1;  int HeroY1;  int HeroX2;  int HeroY2;  };  class gameMap\_village : public gameMap {  public:  gameMap\_village();  ~gameMap\_village();  void LoadBitmap(); //載入地圖  void OnShow(); //顯示地圖  void OnMove();  void setHeroState(int x1, int x2, int y1, int y2, int HP, int Gold, int AttackDamage, int Level);  bool HeroTalkToNPC();  void OnLButtonDown(int Mx, int My, bool \*isTalkingToNPC);  private:  vector<NPC\*> allNPC;  bool HeroIsTalkingToNPC = false;  };  class gameMap\_wild : public gameMap {  public:  gameMap\_wild(string fileName);  virtual ~gameMap\_wild();  virtual void LoadBitmap(); //載入地圖  virtual void OnShow(); //顯示地圖  virtual void OnMove();  virtual void setHeroState(int x1, int x2, int y1, int y2, int HP, int Gold, int AttackDamage, int Level);  virtual void AttackByHero(const int damage); // 攻擊  virtual void AttackByEnemy(int \*heroHP, bool \*Poison);  virtual void SetHeroAttackRange(int x1, int x2, int y1, int y2);  virtual bool GetisStageClear();  virtual void HeroGetItem(int \*HeroGold, int \*SpecialEffect, int \*SpecialEffectCount, int \*HeroHP, int FullHP, int \*ShurikanNumber);  protected:  int HeroAttackX1;  int HeroAttackY1;  int HeroAttackX2;  int HeroAttackY2;  int ItemExistTime = 300;  private:  vector<CEnemy\*> allEnemy;  vector<Item\*> allItem;  bool isStageClear = false;  void DropItem(int x, int y);  void MapObjectInteration(); //改變相對應的地圖物件狀態  };  class Item  {  public:  Item(gameMap\* point, int nx, int ny, int ExistTime);  virtual ~Item() = default;  //~Item();  int GetX1(); // 物品左上角 x 座標  int GetY1(); // 物品左上角 y 座標  int GetX2(); // 物品右下角 x 座標  int GetY2(); // 物品右下角 y 座標  void OnMove(); // 移動物品  void OnShow(); // 將物品圖形貼到畫面  bool isDelete();  virtual int GetItemValue() = 0;  virtual void LoadBitmap() = 0;  protected:  CAnimation animation;  int x;  int y;  gameMap \*currentMap;  int ExistTime = 999999;  };  class Item\_Bronze\_Coin : public Item  {  public:  Item\_Bronze\_Coin(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_Bronze\_Coin();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_Silver\_Coin : public Item  {  public:  Item\_Silver\_Coin(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_Silver\_Coin();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_Golden\_Coin : public Item  {  public:  Item\_Golden\_Coin(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_Golden\_Coin();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_Fire\_Stone : public Item  {  public:  Item\_Fire\_Stone(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_Fire\_Stone();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_RedPot\_Small : public Item  {  public:  Item\_RedPot\_Small(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_RedPot\_Small();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_RedPot\_Medium : public Item  {  public:  Item\_RedPot\_Medium(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_RedPot\_Medium();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_RedPot\_Full : public Item  {  public:  Item\_RedPot\_Full(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_RedPot\_Full();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_RedPot\_Stone : public Item  {  public:  Item\_RedPot\_Stone(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_RedPot\_Stone();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class Item\_Shurikan : public Item  {  public:  Item\_Shurikan(gameMap\* point, int nx, int ny, int ExistTime);  ~Item\_Shurikan();  void LoadBitmap(); // 載入圖形  int GetItemValue();  };  class MapObject  {  public:  MapObject(gameMap\* point, int nx, int ny, bool InitialState, int SetInterationCode);  virtual ~MapObject() = default;  virtual int GetX1() = 0; // 物品左上角 x 座標  virtual int GetY1() = 0; // 物品左上角 y 座標  virtual int GetX2() = 0; // 物品右下角 x 座標  virtual int GetY2() = 0; // 物品右下角 y 座標  virtual void OnMove() = 0; // 移動物品  virtual void OnShow() = 0; // 將物品圖形貼到畫面  virtual void LoadBitmap() = 0;  virtual void GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2) = 0;  virtual int GetInterationCode() = 0;  virtual void SetState(bool State) = 0;  virtual bool GetState() = 0;  virtual void AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP) = 0;  int GetObjectType(); //0:switch,1:spike,2:door  bool GetControl();  protected:  int x;  int y;  bool ObjectState;  gameMap \*currentMap;  int InterationCode = 0; // 預設為0,用於跟其他地圖物件互動ex:-1 to 1 ;-5 to 5  int ObjectType;  bool Control; //true:可控制其他物件,false:不可控制其他物件  };  class Switch : public MapObject  {  public:  Switch(gameMap\* point, int nx, int ny, bool InitialState , int SetInterationCode);  ~Switch();  int GetX1();  int GetY1();  int GetX2();  int GetY2();  void OnMove();  void OnShow();  void LoadBitmap();  void GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2);  int GetInterationCode();  void SetState(bool State);  bool GetState();  void AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP);  private:  CMovingBitmap SwitchOn;  CMovingBitmap SwitchOff;  int GetHitDelayCount = 0;  };  class Monitor : public MapObject  {  public:  Monitor(gameMap\* point, int nx, int ny, bool InitialState, int SetInterationCode);  ~Monitor();  int GetX1();  int GetY1();  int GetX2();  int GetY2();  void OnMove();  void OnShow();  void LoadBitmap();  void GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2);  int GetInterationCode();  void SetState(bool State);  bool GetState();  void AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP);  private:  CAnimation MonitorOn;  CAnimation MonitorOff;  int GetHitDelayCount = 0;  };  class Spike : public MapObject  {  public:  Spike(gameMap\* point, int nx, int ny, bool InitialState , int SetInterationCode);  ~Spike();  int GetX1();  int GetY1();  int GetX2();  int GetY2();  void OnMove();  void OnShow();  void LoadBitmap();  void GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2);  int GetInterationCode();  void SetState(bool State);  bool GetState();  void AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP);  private:  CMovingBitmap SpikeUp;  CMovingBitmap SpikeDown;  int GetHitDelayCount = 0;  int SpikeDamage = 20;  };  class Door : public MapObject  {  public:  Door(gameMap\* point, int nx, int ny, bool InitialState, int SetInterationCode);  ~Door();  int GetX1();  int GetY1();  int GetX2();  int GetY2();  void OnMove();  void OnShow();  void LoadBitmap();  void GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2);  int GetInterationCode();  void SetState(bool State);  bool GetState();  void AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP);  private:  CMovingBitmap DoorPicture;  int InitialY;  };  } |
| gameMap.cpp |
| #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "audio.h"  #include "gamelib.h"  #include "NPC.h"  #include "CEnemy.h"  #include "gameMap.h"  #include <fstream>  #include <string>  #include <iostream>  #include <cstdlib>  #include <ctime>  using namespace std;  namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // gameMap : gameMap base class  /////////////////////////////////////////////////////////////////////////////  gameMap::gameMap(string fileName) : X(0), Y(0), MW(MIN\_MAP\_SIZE), MH(MIN\_MAP\_SIZE), sx(0), sy(0) //圖片為10\*10 初始螢幕畫面位於0,475  {  fstream mapFile;  string mapLocation = "Map\\" + fileName;  for (int i = 0; i < 32; i++)  {  for (int j = 0; j < 96; j++)  {  map[i][j] = 0;  }  }  mapFile.open(mapLocation, ios::in);  if (!mapFile) cout << "error";  int i = 0;  while (!mapFile.eof())  {  string buffer;  getline(mapFile, buffer);  int j = 0;  int count = 0;  while (buffer[j] != '\0')  {  if (buffer[j] == ',')  {  count++;  j++;  }  else  {  map[i][count] = map[i][count] \* 10;  map[i][count] += (int)(buffer[j] - '0');  j++;  }  }  i += 1;  }  }  gameMap::~gameMap()  {  }  void gameMap::SetSXSY(int x, int y) // 設定  {  sx = x;  sy = y;  }  bool gameMap::isSpace(int x, int y) // (x, y)為地圖的點座標  {  int gx = x / MIN\_MAP\_SIZE; // 轉換為格座標(整數除法)  int gy = y / MIN\_MAP\_SIZE; // 轉換為格座標(整數除法)  if (map[gy][gx] == 1 || map[gy][gx] == 2 || map[gy][gx] == 3 || map[gy][gx] == 4 || map[gy][gx] == 8) return false;  return true;  }  int gameMap::GetBlockY(int y)  {  int gy = y / MIN\_MAP\_SIZE; // 轉換為格座標(整數除法)  return gy \* MIN\_MAP\_SIZE;  }  int gameMap::ScreenX(int x) // x 為地圖的點座標  {  return x - sx; // 回傳螢幕的 x 點座標  }  int gameMap::ScreenY(int y) // y 為地圖的 y 點座標  {  return y - sy; // 回傳螢幕的點座標  }  void gameMap::LoadBitmap()  {  ground.LoadBitmap(IDB\_MAPGROUND, RGB(255, 255, 255));//載入草地圖案  ground1.LoadBitmap(IDB\_MAPGROUND1);//載入泥土圖案  ground2.LoadBitmap(IDB\_MAPSLIDE1, RGB(255, 255, 255));//載入斜坡1圖案  ground3.LoadBitmap(IDB\_MAPSLIDE2);//載入斜坡2圖案  }  void gameMap::OnShow()  {  for (int i = 0; i < 32; i++) {  for (int j = 0; j < 96; j++) {  int x = j \* MW - sx; // 算出第(i, j)這一格的 x 螢幕座標  int y = i \* MH - sy; // 算出第(i, j)這一格的 y 螢幕座標  switch (map[i][j])  {  case 4:  ground3.SetTopLeft(x, y); // 指定第(i, j)這一格的座標  ground3.ShowBitmap(); //顯示斜坡2圖案  break;  case 3:  ground2.SetTopLeft(x, y); // 指定第(i, j)這一格的座標  ground2.ShowBitmap(); //顯示斜坡1圖案  break;  case 2:  ground1.SetTopLeft(x, y); // 指定第(i, j)這一格的座標  ground1.ShowBitmap(); //顯示土地圖案  break;  case 1:  ground.SetTopLeft(x, y); // 指定第(i, j)這一格的座標  ground.ShowBitmap(); //顯示草地圖案  break;  case 0:  break;  default:  break;  }  }  }  }  bool gameMap::isDoor(int x, int y)  {  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++)  {  if ((\*i)->GetObjectType() == 2)  {  if ((x <= (\*i)->GetX2()) && (x>=(\*i)->GetX1()) && (y>=(\*i)->GetY1() && y<=(\*i)->GetY2()))  {  return true;  }  }  }  return false;  }  void gameMap::SetHeroXY(int x1, int x2, int y1, int y2)  {  HeroX1 = x1;  HeroY1 = y1;  HeroX2 = x2;  HeroY2 = y2;  }  /////////////////////////////////////////////////////////////////////////////  // gameMap\_village : gameMap\_village class  /////////////////////////////////////////////////////////////////////////////  gameMap\_village::gameMap\_village() : gameMap("Home.txt")  {  allNPC.push\_back(new NPC\_oldMan(this, 200, 350));  }  gameMap\_village::~gameMap\_village()  {  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++) delete (\*i);  }  void gameMap\_village::LoadBitmap()  {  gameMap::LoadBitmap();  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++) (\*i)->LoadBitmap();  }  void gameMap\_village::OnShow()  {  gameMap::OnShow();  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++) (\*i)->OnShow();  }  void gameMap\_village::OnMove()  {  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++) (\*i)->OnMove();  if (allNPC.size() != 0) HeroIsTalkingToNPC = allNPC[0]->isTalkingToHero;  }  bool gameMap\_village::HeroTalkToNPC()  {  bool success = false;  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++)  {  if ((\*i)->SetIsTalkingToHero())  {  success = true;  }  }  return success;  }  void gameMap\_village::OnLButtonDown(int Mx, int My, bool \*isTalkingToNPC)  {  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++)  {  (\*i)->OnLButtonDown(Mx, My, isTalkingToNPC);  }  }  void gameMap\_village::setHeroState(int x1, int x2, int y1, int y2, int HP, int Gold, int AttackDamage, int Level)  {  for (vector<NPC\*>::iterator i = allNPC.begin(); i != allNPC.end(); i++) (\*i)->SetHeroState(x1, x2, y1, y2, HP, Gold, AttackDamage, Level);  }  /////////////////////////////////////////////////////////////////////////////  // gameMap\_wild : gameMap\_wild class  /////////////////////////////////////////////////////////////////////////////  gameMap\_wild::gameMap\_wild(string fileName) : gameMap(fileName)  {  unsigned seed;  seed = (unsigned)time(NULL);  srand(seed);  for (int i = 0; i < 32; i++) {  for (int j = 0; j < 96; j++) {  int x = j \* MW - sx; // 算出第(i, j)這一格的 x 螢幕座標  int y = i \* MH - sy; // 算出第(i, j)這一格的 y 螢幕座標  if (map[i][j] / 10 == 2)  {  allObject.push\_back(new Switch(this, x, y, true, map[i][j] % 10));  }  if (map[i][j] / 10 == 6)  {  allObject.push\_back(new Monitor(this, x, y, true, map[i][j] % 10));  }  if (map[i][j] / 10 == 3)  {  allObject.push\_back(new Spike(this, x, y + 10, true, map[i][j] % 10 \* (-1)));  allObject.push\_back(new Spike(this, x + 10, y + 10, true, map[i][j] % 10 \* (-1)));  }  if (map[i][j] / 10 == 5)  {  allObject.push\_back(new Door(this, x, y, true, map[i][j] % 10 \* (-1)));  }  switch (map[i][j])  {  case 9:  allEnemy.push\_back(new CEnemy\_Statue(this, x, y));  break;  case 10:  allEnemy.push\_back(new CEnemy\_Cloud(this, x, y));  break;  case 40:  allEnemy.push\_back(new CEnemy\_sunFlower(this, x, y));  break;  case 41:  allEnemy.push\_back(new CEnemy\_Cactus(this, x, y));  break;  case 42:  allEnemy.push\_back(new CEnemy\_GasRobot(this, x, y));  break;  case 43:  allEnemy.push\_back(new CEnemy\_RobotA(this, x, y));  break;  case 44:  allEnemy.push\_back(new CEnemy\_Pigeon(this, x, y));  break;  case 45:  allEnemy.push\_back(new CEnemy\_Scorpoin(this, x, y));  break;  case 90:  allItem.push\_back(new Item\_Bronze\_Coin(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 91:  allItem.push\_back(new Item\_Silver\_Coin(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 92:  allItem.push\_back(new Item\_Golden\_Coin(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 93:  allItem.push\_back(new Item\_Fire\_Stone(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 94:  allItem.push\_back(new Item\_RedPot\_Small(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 95:  allItem.push\_back(new Item\_RedPot\_Medium(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 96:  allItem.push\_back(new Item\_RedPot\_Full(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 97:  allItem.push\_back(new Item\_RedPot\_Stone(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 98:  allItem.push\_back(new Item\_Shurikan(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  default:  break;  }  }  }  }  gameMap\_wild::~gameMap\_wild()  {  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) delete (\*i);  for (vector<Item\*>::iterator i = allItem.begin(); i != allItem.end(); i++) delete (\*i);  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++) delete(\*i);  }  void gameMap\_wild::LoadBitmap()  {  gameMap::LoadBitmap();  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->LoadBitmap();  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++) (\*i)->LoadBitmap();  }  void gameMap\_wild::OnShow()  {  gameMap::OnShow();  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->OnShow();  for (vector<Item\*>::iterator i = allItem.begin(); i != allItem.end(); i++) (\*i)->OnShow();  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++) (\*i)->OnShow()  }  void gameMap\_wild::OnMove() {  vector<CEnemy\*>::iterator iter = allEnemy.begin();  MapObjectInteration();  while (iter != allEnemy.end()) //敵人死亡會從vector裡被刪除  {  if ((\*iter)->isDead() && (\*iter)->GetEnemyType() == "Statue")  {  isStageClear = true; //通關完成  }  if ((\*iter)->isDead() && (\*iter)->GetEnemyType() != "Statue") //雕像以外的敵人被打死  {  if ((\*iter)->GetEnemyType() == "Cloud Boss")  {  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++)  {  if ((\*i)->GetInterationCode() == -9)  {  (\*i)->SetState(false);  }  }  }  DropItem(((\*iter)->GetX1() + (\*iter)->GetX2()) / 2, ((\*iter)->GetY1() + (\*iter)->GetY2()) / 2);  delete \*iter;  iter = allEnemy.erase(iter);  }  else  iter++;  }  for (vector<Item\*>::iterator i = allItem.begin(); i != allItem.end(); i++) (\*i)->OnMove();  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->OnMove();  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++) (\*i)->OnMove();  }  void gameMap\_wild::setHeroState(int x1, int x2, int y1, int y2, int HP, int Gold, int AttackDamage, int Level)  {  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->SetHeroXY(x1, x2, y1, y2);  }  void gameMap\_wild::SetHeroAttackRange(int x1, int x2, int y1, int y2)  {  HeroAttackX1 = x1;  HeroAttackY1 = y1;  HeroAttackX2 = x2;  HeroAttackY2 = y2;  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->SetHeroAttackRange(x1, x2, y1, y2);  }  void gameMap\_wild::AttackByHero(const int damage) // 攻擊  {  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->GetAttack(damage);  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++) (\*i)->GetAttack(HeroAttackX1, HeroAttackY1, HeroAttackX2, HeroAttackY2);  }  void gameMap\_wild::AttackByEnemy(int \*heroHP, bool \*Poison)  {  for (vector<CEnemy\*>::iterator i = allEnemy.begin(); i != allEnemy.end(); i++) (\*i)->AttackByEnemy(heroHP, Poison);  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++) (\*i)->AttackByObject(HeroX1, HeroY1, HeroX2, HeroY2, heroHP);  }  void gameMap\_wild::HeroGetItem(int \*HeroCoin, int \*SpecialEffect, int \*SpecialEffectCount, int \*HeroHP, int FullHP, int \*ShurikanNumber)  {  vector<Item\*>::iterator iter = allItem.begin();  while (iter != allItem.end())  {  if (((\*iter)->GetX2() >= HeroX1) && (HeroX2 >= (\*iter)->GetX1()) && ((\*iter)->GetY2() >= HeroY1) && (HeroY2 >= (\*iter)->GetY1()))  {  CAudio::Instance()->Play(5, false);  switch ((\*iter)->GetItemValue())  {  case 1: //火焰石  \*SpecialEffect = 1;  \*SpecialEffectCount = 3;  delete \*iter;  iter = allItem.erase(iter);  break;  case 2: // 小血瓶  if (\*HeroHP + 30 >= FullHP) \*HeroHP = FullHP;  else \*HeroHP += 30;  delete \*iter;  iter = allItem.erase(iter);  break;  case 3: // 中血瓶  if (\*HeroHP + 50 >= FullHP) \*HeroHP = FullHP;  else \*HeroHP += 50;  delete \*iter;  iter = allItem.erase(iter);  break;  case 4: // 大血瓶  \*HeroHP = FullHP;  delete \*iter;  iter = allItem.erase(iter);  break;  case 5: // 手裡劍  \*ShurikanNumber += 10;  delete \*iter;  iter = allItem.erase(iter);  break;  case 9: // 紅石  \*SpecialEffect = 2;  delete \*iter;  iter = allItem.erase(iter);  break;  default: //金幣  \*HeroCoin += (\*iter)->GetItemValue();  delete \*iter;  iter = allItem.erase(iter);  break;  }  }  else if ((\*iter)->isDelete())  {  delete \*iter;  iter = allItem.erase(iter);  }  else  iter++;  }  }  bool gameMap\_wild::GetisStageClear()  {  return isStageClear;  }  void gameMap\_wild::DropItem(int x, int y)  {  int num;  num = (rand() % 9);  switch (num)  {  case 0:  allItem.push\_back(new Item\_Bronze\_Coin(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 1:  allItem.push\_back(new Item\_Silver\_Coin(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 2:  allItem.push\_back(new Item\_Golden\_Coin(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 3:  allItem.push\_back(new Item\_Fire\_Stone(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 4:  allItem.push\_back(new Item\_RedPot\_Small(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 5:  allItem.push\_back(new Item\_RedPot\_Full(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 6:  allItem.push\_back(new Item\_RedPot\_Stone(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  case 7:  allItem.push\_back(new Item\_Shurikan(this, x, y, ItemExistTime));  allItem.back()->LoadBitmap();  break;  default:  break;  }  }  void gameMap\_wild::MapObjectInteration()  {  for (vector<MapObject\*>::iterator i = allObject.begin(); i != allObject.end(); i++)  {  for (vector<MapObject\*>::iterator j = allObject.begin(); j != allObject.end(); j++)  {  if (((\*i)->GetInterationCode() == ((-1)\*(\*j)->GetInterationCode())) && (\*i)->GetControl())  {  (\*j)->SetState((\*i)->GetState());  }  }  }  }  /////////////////////////////////////////////////////////////////////////////  // Item: Item base class  /////////////////////////////////////////////////////////////////////////////  Item::Item(gameMap\* point, int nx, int ny, int ExistTime)  {  x = nx;  y = ny;  currentMap = point;  animation.SetDelayCount(3);  }  int Item::GetX1()  {  return x;  }  int Item::GetY1()  {  return y;  }  int Item::GetX2()  {  return x + animation.Width();  }  int Item::GetY2()  {  return y + animation.Height();  }  bool Item::isDelete()  {  if (ExistTime == 0)  {  return true;  }  return false;  }  void Item::OnMove()  {  animation.OnMove();  if (ExistTime > 0) ExistTime--;  }  void Item::OnShow()  {  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  }  /////////////////////////////////////////////////////////////////////////////  // Item\_Bronze\_Coin : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_Bronze\_Coin::Item\_Bronze\_Coin(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_Bronze\_Coin::~Item\_Bronze\_Coin() {}  void Item\_Bronze\_Coin::LoadBitmap()  {  animation.AddBitmap(IDB\_BRONZECOIN\_0, RGB(0, 162, 232));  animation.AddBitmap(IDB\_BRONZECOIN\_1, RGB(0, 162, 232));  animation.AddBitmap(IDB\_BRONZECOIN\_2, RGB(0, 162, 232));  animation.AddBitmap(IDB\_BRONZECOIN\_3, RGB(0, 162, 232));  }  int Item\_Bronze\_Coin::GetItemValue()  {  return 10;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_Silver\_Coin : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_Silver\_Coin::Item\_Silver\_Coin(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_Silver\_Coin::~Item\_Silver\_Coin() {}  void Item\_Silver\_Coin::LoadBitmap()  {  animation.AddBitmap(IDB\_SILVERCOIN\_0, RGB(0, 162, 232));  animation.AddBitmap(IDB\_SILVERCOIN\_1, RGB(0, 162, 232));  animation.AddBitmap(IDB\_SILVERCOIN\_2, RGB(0, 162, 232));  animation.AddBitmap(IDB\_SILVERCOIN\_3, RGB(0, 162, 232));  }  int Item\_Silver\_Coin::GetItemValue()  {  return 30;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_Silver\_Coin : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_Golden\_Coin::Item\_Golden\_Coin(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_Golden\_Coin::~Item\_Golden\_Coin() {}  void Item\_Golden\_Coin::LoadBitmap()  {  animation.AddBitmap(IDB\_GOLDCOIN\_0, RGB(0, 162, 232));  animation.AddBitmap(IDB\_GOLDCOIN\_1, RGB(0, 162, 232));  animation.AddBitmap(IDB\_GOLDCOIN\_2, RGB(0, 162, 232));  animation.AddBitmap(IDB\_GOLDCOIN\_3, RGB(0, 162, 232));  }  int Item\_Golden\_Coin::GetItemValue()  {  return 100;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_Fire\_Stone : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_Fire\_Stone::Item\_Fire\_Stone(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_Fire\_Stone::~Item\_Fire\_Stone() {}  void Item\_Fire\_Stone::LoadBitmap()  {  animation.AddBitmap(IDB\_FIRESTONE, RGB(255, 255, 255));  }  int Item\_Fire\_Stone::GetItemValue()  {  return 1;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_Redpot\_Stone : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_RedPot\_Stone::Item\_RedPot\_Stone(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_RedPot\_Stone::~Item\_RedPot\_Stone() {}  void Item\_RedPot\_Stone::LoadBitmap()  {  animation.AddBitmap(IDB\_REDSTONE, RGB(63, 72, 204));  }  int Item\_RedPot\_Stone::GetItemValue()  {  return 9;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_RedPot\_Small : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_RedPot\_Small::Item\_RedPot\_Small(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_RedPot\_Small::~Item\_RedPot\_Small() {}  void Item\_RedPot\_Small::LoadBitmap()  {  animation.AddBitmap(IDB\_REDPOTSMALL, RGB(63, 72, 204));  }  int Item\_RedPot\_Small::GetItemValue()  {  return 2;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_RedPot\_Medium : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_RedPot\_Medium::Item\_RedPot\_Medium(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_RedPot\_Medium::~Item\_RedPot\_Medium() {}  void Item\_RedPot\_Medium::LoadBitmap()  {  animation.AddBitmap(IDB\_REDPOTMEDIUM, RGB(63, 72, 204));  }  int Item\_RedPot\_Medium::GetItemValue()  {  return 3;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_RedPot\_Full : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_RedPot\_Full::Item\_RedPot\_Full(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_RedPot\_Full::~Item\_RedPot\_Full() {}  void Item\_RedPot\_Full::LoadBitmap()  {  animation.AddBitmap(IDB\_REDPOTFULL, RGB(63, 72, 204));  }  int Item\_RedPot\_Full::GetItemValue()  {  return 4;  }  /////////////////////////////////////////////////////////////////////////////  // Item\_Shurikan : Item class  /////////////////////////////////////////////////////////////////////////////  Item\_Shurikan::Item\_Shurikan(gameMap\* point, int nx, int ny, int ExistTime) : Item(point, nx, ny, ExistTime) {}  Item\_Shurikan::~Item\_Shurikan() {}  void Item\_Shurikan::LoadBitmap()  {  animation.AddBitmap(IDB\_ITEMSHURIKAN, RGB(63, 72, 204));  }  int Item\_Shurikan::GetItemValue()  {  return 5;  }  /////////////////////////////////////////////////////////////////////////////  // MapObject  /////////////////////////////////////////////////////////////////////////////  MapObject::MapObject(gameMap\* point, int nx, int ny, bool InitialState, int SetInterationCode)  {  x = nx;  y = ny;  currentMap = point;  InterationCode = SetInterationCode;  ObjectState = InitialState;  }  int MapObject::GetObjectType()  {  return ObjectType;  }  bool MapObject::GetControl()  {  return Control;  }  /////////////////////////////////////////////////////////////////////////////  // class Switch : class MapObject  /////////////////////////////////////////////////////////////////////////////  Switch::Switch(gameMap\* point, int nx, int ny, bool InitialState , int SetInterationCode) : MapObject(point, nx, ny, InitialState, SetInterationCode)  {  ObjectType = 0;  Control = true;  }  Switch::~Switch(){}  int Switch::GetX1()  {  return x;  }  int Switch::GetY1()  {  return y;  }  int Switch::GetX2()  {  return x + SwitchOff.Width();  }  int Switch::GetY2()  {  return y + SwitchOff.Height();  }  void Switch::LoadBitmap()  {  SwitchOff.LoadBitmap(IDB\_SWITCH\_OFF, RGB(255, 255, 255));  SwitchOn.LoadBitmap(IDB\_SWITCH\_ON, RGB(255, 255, 255));  }  void Switch::OnMove()  {  if (GetHitDelayCount > 0) GetHitDelayCount--;  }  void Switch::OnShow()  {  if (ObjectState)  {  SwitchOn.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  SwitchOn.ShowBitmap();  }  else  {  SwitchOff.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  SwitchOff.ShowBitmap();  }  }  void Switch::GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2)  {  if ((GetX2() >= HeroX1) && (HeroX2 >= GetX1()) && (GetY2() >= HeroY1) && (HeroY2 >= GetY1()) && GetHitDelayCount == 0)  {  ObjectState = !ObjectState;  CAudio::Instance()->Play(11, false);  GetHitDelayCount = 15;  }  }  int Switch::GetInterationCode()  {  return InterationCode;  }  void Switch::SetState(bool State)  {  ObjectState = State;  }  bool Switch::GetState()  {  return ObjectState;  }  void Switch::AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP)  {  }  /////////////////////////////////////////////////////////////////////////////  // class Monitor : class MapObject  /////////////////////////////////////////////////////////////////////////////  Monitor::Monitor(gameMap\* point, int nx, int ny, bool InitialState, int SetInterationCode) : MapObject(point, nx, ny, InitialState, SetInterationCode)  {  ObjectType = 3;  Control = true;  }  Monitor::~Monitor() {}  int Monitor::GetX1()  {  return x;  }  int Monitor::GetY1()  {  return y;  }  int Monitor::GetX2()  {  return x + MonitorOff.Width();  }  int Monitor::GetY2()  {  return y + MonitorOff.Height();  }  void Monitor::LoadBitmap()  {  MonitorOff.AddBitmap(IDB\_MONITOR\_OFF, RGB(63,72,204));  MonitorOn.AddBitmap(IDB\_MONITOR\_ON\_0, RGB(63, 72, 204));  MonitorOn.AddBitmap(IDB\_MONITOR\_ON\_1, RGB(63, 72, 204));  }  void Monitor::OnMove()  {  if (GetHitDelayCount > 0) GetHitDelayCount--;  MonitorOn.OnMove();  MonitorOff.OnMove();  }  void Monitor::OnShow()  {  if (ObjectState)  {  MonitorOn.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  MonitorOn.OnShow();  }  else  {  MonitorOff.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  MonitorOff.OnShow();  }  }  void Monitor::GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2)  {  if ((GetX2() >= HeroX1) && (HeroX2 >= GetX1()) && (GetY2() >= HeroY1) && (HeroY2 >= GetY1()) && GetHitDelayCount == 0)  {  if (ObjectState)  {  ObjectState = false;  CAudio::Instance()->Play(11, false);  }  GetHitDelayCount = 15;  }  }  int Monitor::GetInterationCode()  {  return InterationCode;  }  void Monitor::SetState(bool State)  {  ObjectState = State;  }  bool Monitor::GetState()  {  return ObjectState;  }  void Monitor::AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP)  {  }  /////////////////////////////////////////////////////////////////////////////  // class Spike : class MapObject  /////////////////////////////////////////////////////////////////////////////  Spike::Spike(gameMap\* point, int nx, int ny, bool InitialState , int SetInterationCode) : MapObject(point, nx, ny, InitialState, SetInterationCode)  {  ObjectType = 1;  Control = false;  }  Spike::~Spike() {}  int Spike::GetX1()  {  return x;  }  int Spike::GetY1()  {  return y;  }  int Spike::GetX2()  {  return x + SpikeUp.Width();  }  int Spike::GetY2()  {  return y + SpikeUp.Height();  }  void Spike::LoadBitmap()  {  SpikeDown.LoadBitmap(IDB\_POPUPSPIKE\_3, RGB(63, 72, 204));  SpikeUp.LoadBitmap(IDB\_POPUPSPIKE\_0, RGB(63, 72, 204));  }  void Spike::OnMove()  {  }  void Spike::OnShow()  {  if (ObjectState)  {  SpikeUp.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  SpikeUp.ShowBitmap();  }  else  {  SpikeDown.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  SpikeDown.ShowBitmap();  }  }  void Spike::GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2)  {  }  int Spike::GetInterationCode()  {  return InterationCode;  }  void Spike::SetState(bool State)  {  ObjectState = State;  }  bool Spike::GetState()  {  return ObjectState;  }  void Spike::AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP)  {  if ((GetX2() >= HeroX1) && (HeroX2 >= GetX1()) && (GetY2() >= HeroY1) && (HeroY2 >= GetY1()) && ObjectState)  {  \*heroHP -= SpikeDamage;  }  }  /////////////////////////////////////////////////////////////////////////////  // class Door : class MapObject  /////////////////////////////////////////////////////////////////////////////  Door::Door(gameMap\* point, int nx, int ny, bool InitialState, int SetInterationCode) : MapObject(point, nx, ny, InitialState, SetInterationCode)  {  InitialY = ny;  ObjectType = 2;  Control = false;  }  Door::~Door() {}  int Door::GetX1()  {  return x;  }  int Door::GetY1()  {  return y;  }  int Door::GetX2()  {  return x + DoorPicture.Width();  }  int Door::GetY2()  {  return y + DoorPicture.Height();  }  void Door::LoadBitmap()  {  DoorPicture.LoadBitmap(IDB\_DOOR, RGB(255, 255, 255));  }  void Door::OnMove()  {  if (ObjectState)  {  y = InitialY;  }  else  {  y = InitialY - 100;  }  }  void Door::OnShow()  {  DoorPicture.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DoorPicture.ShowBitmap();  }  void Door::GetAttack(int HeroX1, int HeroY1, int HeroX2, int HeroY2)  {  }  int Door::GetInterationCode()  {  return InterationCode;  }  void Door::SetState(bool State)  {  ObjectState = State;  }  bool Door::GetState()  {  return ObjectState;  }  void Door::AttackByObject(int HeroX1, int HeroY1, int HeroX2, int HeroY2, int \*heroHP)  {  }  } |
| CHero.h |
| namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // 這個class提供可以用鍵盤或滑鼠控制的英雄  // 看懂就可以改寫成自己的程式了  /////////////////////////////////////////////////////////////////////////////  enum Hero\_Action  {  STAND,  WALK,  RUN,  ROLL,  ATTACK,  TALK,  THROW  };  enum Hero\_Direction  {  LEFT, RIGHT  };  class HeroBullet  {  public:  HeroBullet(gameMap\* point, int nx, int ny, int step);  ~HeroBullet();  int GetX1(); // 子彈左上角 x 座標  int GetY1(); // 子彈左上角 y 座標  int GetX2(); // 子彈右下角 x 座標  int GetY2(); // 子彈右下角 y 座標  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  bool isDelet();  protected:  CAnimation animation;  int x;  int y;  gameMap \*currentMap;  int distance;  int STEP\_SIZE;  };  class Shurikan : public HeroBullet  {  public:  Shurikan(gameMap\* point, int nx, int ny, int step);  ~Shurikan();  void LoadBitmap(); // 載入圖形  };  class CHero  {  public:  CHero();  ~CHero();  int GetX1(); // 英雄左上角 x 座標  int GetY1(); // 英雄左上角 y 座標  int GetX2(); // 英雄右下角 x 座標  int GetY2(); // 英雄右下角 y 座標  int GetWidth(); // 英雄的寬  int GetHeight(); // 英雄的高  int GetCenterX(); // 英雄正中央的 x 座標  int GetCenterY(); // 英雄正中央的 y 座標  void Initialize(); // 設定英雄為初始值  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動英雄  void OnShow(); // 將英雄圖形貼到畫面  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 SetTalkingToNPC(bool flag);// 和NPC對話  void SetHeroRoll(bool flag); // 翻滾  void SetHeroAttack(bool flag); // 設定是否正在往上移動  void SetHeroThrow(bool flag);  void SetXY(int nx, int ny); // 設定英雄左上角座標  void SetHeroHP(int inputHP); // 設定主角HP值  void AddHeroGold();  bool isAttacking; // 正在攻擊  int GetHeroFullHP();  int GetHeroCurrentHP();  int AttackByEnemy();  int HeroGetItem();  void SetPreviousMove(int Movement);  int GetPreviousMove(); //取得上一個動作  void SetMoveDelayCount(int delay);  void ShowNumber(int colot, int Number, int x, int y);//color:1:black,2:red  void HeroLevelUp(); // 升級  void SetEndTalking(); // 關閉與NPC對話  bool isTalkingToNPC; // 與NPC對話  int HeroLevel = 1;  void SelectMap(int MapNumber);  void ResetHeroState(); //重置主角狀態(回到城鎮)  bool ClearedStage = false; //通關結算  void OnLButtonDown(int Mx, int My); //按下滑鼠左鍵  bool GetHeroIsRolling();  bool isAlive(); // 回傳主角是否活著  protected:  CAnimation animation; // 英雄的動畫(向右)  CAnimation animation1; // 英雄的動畫(向左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation jumpAnimation; // 跳躍動畫(向右)  CAnimation jumpAnimation1; // 跳躍動畫(向左)  CAnimation HeroAttackMovement; // 英雄攻擊動畫(向右)  CAnimation HeroAttackMovement1; // 英雄攻擊動畫(向左)  CAnimation sword; // 載入劍的動畫(向右)  CAnimation sword1; // 載入劍的動畫(向左)  CAnimation swordAttack; // 劍的攻擊動畫(向右)  CAnimation swordAttack1; // 劍的攻擊動畫(向左)  CAnimation HeroDashLeft; // 向左衝刺  CAnimation SwordDashLeft;  CAnimation HeroDashRight; // 向右衝刺  CAnimation SwordDashRight;  CAnimation HeroRollLeft; // 向左翻滾  CAnimation SwordRollLeft;  CAnimation HeroRollRight; // 向右翻滾  CAnimation SwordRollRight;  CAnimation ThrowingLeft;  CAnimation ThrowingRight;  CAnimation FireSwordRightAnimation;  CAnimation FireSwordLeftAnimation;  CAnimation FireCircle;  CAnimation Fire1;  CAnimation Fire2;  CAnimation Fire3;  CAnimation gain\_life;  CMovingBitmap Infected\_UI;  CMovingBitmap LifeBarHead; // 血條  CMovingBitmap StartGameBar; // 開始按鈕  CMovingBitmap WorldMap\_UI\_1; // 地圖\_1  CMovingBitmap White;  CMovingBitmap ShurikanUI;  CInteger Num; // 黑色數字  CInteger\_Red Num\_Red; // 紅色數字  CInteger\_Gold Num\_Gold; // 黃色數字  CMovingBitmap Word\_G;  CMovingBitmap Word\_Gold;  CMovingBitmap Shield;  CMovingBitmap GainLifeUI;  //CInteger DamageTaken; // 顯示承受傷害  vector<CMovingBitmap\*> LifeBarRed; // 血條  int x, y; // 英雄左上角座標  int heroAttackDamage; // 英雄攻擊力  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 isRolling; // 是否正在翻滾  bool isInvincible; // 是否為無敵(無法被攻擊)  bool isThrowing;  //string faceDirection; // 人物面對的方向  int Gold; // 金幣  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int SetAttackDelayCount; // 設定攻擊延遲時間  int AttackDelayCount; // 攻擊延遲時間  int RollDelayCount; // 翻滾延遲時間  int MoveDelayCount; // 上個動作保存時間  int GainLifeDelayCount; // 顯示回血特效長度  int PreviousMovement; // 紀錄上一個動作 0:無動作; 1:向左走; 2:向右走  int DashColdDown; // 衝刺冷卻時間  int InvincibleDelayCount; // 無敵時間  int ShowGoldDelayCount; // 金幣出現的時間  int GainHealthDelayCount; // 取得紅石後的回血時間長度  int bleed = 0; // 偵測主角有無扣血  int GetGold = 0; // 得到的金幣  int SpecialEffect = 0; // 0:無效果;1:火焰;2:持續回血  int SpecialEffectCount = 0; // 偵測特殊效果  private:  gameMap\_village\* currentVillage; // 紀錄目前的村莊  gameMap\_wild\* currentWild; // 紀錄目前的野外  gameMap\* currentMap; // 紀錄目前的地圖  CMovingBitmap BlackMask; // 黑色遮罩  CMovingBitmap QuitButton; // 離開按鈕  CMovingBitmap B;  vector<Shurikan\*> allShurikan;  int FullHP; // 主角總血量  int CurrentHP; // 主角目前的血量  void changeLifeBarLength(); // 改變生命條的長度  bool isInHome; // 是否在村莊  bool isSelectingMap; // 是否正在選擇地圖  int AttackRange;  Hero\_Action heroActoin; // 英雄動作  Hero\_Direction heroDirection; // 英雄面對的方向  void setHeroAction(); // 設定英雄動作  void WalkOnMove();  void RollOnMove();  void StandOnMove();  void WalkOnShow();  void RollOnShow();  void StandOnShow();  void AttackOnMove();  void AttackOnShow();  void ThrowOnMove();  void ThrowOnShow();  CAnimation currentAnimation; // 目前的英雄動畫  bool HasFireStone = false;  int ShurikanNumber = 10;  bool Poison = false;  int PoisonDelayCount = 0;  bool isInit;  };  } |
| CHero.cpp |
| #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "audio.h"  #include "gamelib.h"  #include "gameMap.h"  #include "Counter.h"  #include "CHero.h"  namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // CHero class  /////////////////////////////////////////////////////////////////////////////  CHero::CHero()  {  currentVillage = NULL;  currentWild = NULL;  currentMap = NULL;  isInit = true;  for (int i = 0; i < 100; i++) LifeBarRed.push\_back(new CMovingBitmap); //100個血條圖片  FullHP = 200; // 主角預設血量為200  heroAttackDamage = 20; // 主角預設攻擊力為20  Gold = 20;  Initialize();  }  CHero::~CHero()  {  delete currentVillage;  delete currentWild;  for (vector<CMovingBitmap\*>::iterator i = LifeBarRed.begin(); i != LifeBarRed.end(); i++) delete (\*i);  for (vector<Shurikan\*>::iterator i = allShurikan.begin(); i != allShurikan.end(); i++) delete \*i;  }  int CHero::GetX1()  {  return x;  }  int CHero::GetY1()  {  return y;  }  int CHero::GetX2()  {  return x + animation.Width();  }  int CHero::GetY2()  {  return y + animation.Height();  }  int CHero::GetWidth()  {  return animation.Width();  }  int CHero::GetHeight()  {  return animation.Height();  }  int CHero::GetCenterX()  {  return x + animation.Width() / 2;  }  int CHero::GetCenterY()  {  return y + animation.Height() / 2;  }  void CHero::Initialize()  {  const int X\_POS = 35;  const int Y\_POS = 0;  x = X\_POS;  y = Y\_POS;  isMovingLeft = isMovingRight = isMovingUp = isMovingDown = isAttacking = isRolling = isInvincible = isTalkingToNPC = isThrowing = false;  PreviousMovement = 0; //紀錄上一個動作  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  floor = FLOOR;  rising = false;  initial\_velocity = INITIAL\_VELOCITY;  velocity = initial\_velocity;  animation.SetDelayCount(5);  animation1.SetDelayCount(5);  sword.SetDelayCount(3);  HeroDashLeft.SetDelayCount(3);  HeroDashRight.SetDelayCount(3);  swordAttack.SetDelayCount(3);  swordAttack1.SetDelayCount(3);  HeroAttackMovement.SetDelayCount(3);  HeroAttackMovement1.SetDelayCount(3);  sword1.SetDelayCount(3);  moveRightAnimation.SetDelayCount(3);  jumpAnimation.SetDelayCount(4);  jumpAnimation1.SetDelayCount(4);  moveLeftAnimation.SetDelayCount(3);  HeroRollLeft.SetDelayCount(3);  HeroRollRight.SetDelayCount(3);  SwordRollRight.SetDelayCount(5);  SwordRollLeft.SetDelayCount(5);  SwordDashLeft.SetDelayCount(3);  SwordDashRight.SetDelayCount(3);  FireSwordRightAnimation.SetDelayCount(3);  FireSwordLeftAnimation.SetDelayCount(3);  FireCircle.SetDelayCount(3);  Fire1.SetDelayCount(3);  Fire2.SetDelayCount(3);  Fire3.SetDelayCount(3);  gain\_life.SetDelayCount(3);  ThrowingLeft.SetDelayCount(3);  ThrowingRight.SetDelayCount(3);  SetAttackDelayCount = AttackDelayCount = DashColdDown = 15;  GainHealthDelayCount = 0;  GainLifeDelayCount = 0;  ShowGoldDelayCount = 0;  RollDelayCount = 15;  InvincibleDelayCount = 30;  MoveDelayCount = 10; // 主角預設血量為100  CurrentHP = FullHP;  AttackRange = 100; // 主角攻擊範圍  isSelectingMap = false;  AttackRange = 150;  heroDirection = LEFT;  heroActoin = STAND;  if (isInit == false)  {  SelectMap(0);  }  isInit = false;  }  void CHero::LoadBitmap()  {  animation.AddBitmap(IDB\_HeroNoMove\_1, RGB(255, 255, 255));  animation.AddBitmap(IDB\_HeroNoMove\_2, RGB(255, 255, 255));  animation.AddBitmap(IDB\_HeroNoMove\_3, RGB(255, 255, 255));  animation.AddBitmap(IDB\_HeroNoMove\_4, RGB(255, 255, 255));  animation.AddBitmap(IDB\_HeroNoMove\_5, RGB(255, 255, 255));  animation1.AddBitmap(IDB\_HERONOMOVELEFT\_1, RGB(255, 255, 255));  animation1.AddBitmap(IDB\_HERONOMOVELEFT\_2, RGB(255, 255, 255));  animation1.AddBitmap(IDB\_HERONOMOVELEFT\_3, RGB(255, 255, 255));  animation1.AddBitmap(IDB\_HERONOMOVELEFT\_4, RGB(255, 255, 255));  animation1.AddBitmap(IDB\_HERONOMOVELEFT\_5, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_HEROMOVERIGHT\_1, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_HEROMOVERIGHT\_2, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_HEROMOVERIGHT\_3, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_HEROMOVERIGHT\_4, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_HEROMOVERIGHT\_5, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_HEROMOVELEFT\_1, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_HEROMOVELEFT\_2, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_HEROMOVELEFT\_3, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_HEROMOVELEFT\_4, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_HEROMOVELEFT\_5, RGB(255, 255, 255));  //jumpAnimation.AddBitmap(IDB\_HEROJUMP\_3, RGB(255, 255, 255));  jumpAnimation.AddBitmap(IDB\_HEROJUMP\_4, RGB(255, 255, 255));  jumpAnimation.AddBitmap(IDB\_HEROJUMP\_5, RGB(255, 255, 255));  jumpAnimation.AddBitmap(IDB\_HEROJUMP\_6, RGB(255, 255, 255));  jumpAnimation.AddBitmap(IDB\_HEROJUMP\_7, RGB(255, 255, 255));  jumpAnimation1.AddBitmap(IDB\_HEROJUMP\_0\_1, RGB(255, 255, 255));  jumpAnimation1.AddBitmap(IDB\_HEROJUMP\_1\_1, RGB(255, 255, 255));  jumpAnimation1.AddBitmap(IDB\_HEROJUMP\_2\_1, RGB(255, 255, 255));  jumpAnimation1.AddBitmap(IDB\_HEROJUMP\_3\_1, RGB(255, 255, 255));  jumpAnimation1.AddBitmap(IDB\_HEROJUMP\_4\_1, RGB(255, 255, 255));  HeroDashLeft.AddBitmap(IDB\_DASH\_LEFT\_1, RGB(255, 255, 255));  HeroDashLeft.AddBitmap(IDB\_DASH\_LEFT\_2, RGB(255, 255, 255));  HeroDashLeft.AddBitmap(IDB\_DASH\_LEFT\_3, RGB(255, 255, 255));  HeroDashLeft.AddBitmap(IDB\_DASH\_LEFT\_4, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_0, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_1, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_2, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_3, RGB(255, 255, 255));  HeroDashRight.AddBitmap(IDB\_DASH\_RIGHT\_1, RGB(255, 255, 255));  HeroDashRight.AddBitmap(IDB\_DASH\_RIGHT\_2, RGB(255, 255, 255));  HeroDashRight.AddBitmap(IDB\_DASH\_RIGHT\_3, RGB(255, 255, 255));  HeroDashRight.AddBitmap(IDB\_DASH\_RIGHT\_4, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_0, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_1, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_2, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_3, RGB(255, 255, 255));  sword.AddBitmap(IDB\_sword\_1, RGB(255, 255, 255));  sword.AddBitmap(IDB\_sword\_2, RGB(255, 255, 255));  sword.AddBitmap(IDB\_sword\_3, RGB(255, 255, 255));  sword.AddBitmap(IDB\_sword\_4, RGB(255, 255, 255));  sword.AddBitmap(IDB\_sword\_5, RGB(255, 255, 255));  sword.AddBitmap(IDB\_sword\_7, RGB(255, 255, 255));  sword1.AddBitmap(IDB\_SWORD\_1\_1, RGB(255, 255, 255));  sword1.AddBitmap(IDB\_SWORD\_2\_1, RGB(255, 255, 255));  sword1.AddBitmap(IDB\_SWORD\_3\_1, RGB(255, 255, 255));  sword1.AddBitmap(IDB\_SWORD\_4\_1, RGB(255, 255, 255));  sword1.AddBitmap(IDB\_SWORD\_5\_1, RGB(255, 255, 255));  sword1.AddBitmap(IDB\_SWORD\_7\_1, RGB(255, 255, 255));  HeroAttackMovement.AddBitmap(IDB\_HEROATTACK\_1, RGB(255, 255, 255));  HeroAttackMovement.AddBitmap(IDB\_HEROATTACK\_2, RGB(255, 255, 255));  HeroAttackMovement.AddBitmap(IDB\_HEROATTACK\_3, RGB(255, 255, 255));  HeroAttackMovement.AddBitmap(IDB\_HEROATTACK\_4, RGB(255, 255, 255));  HeroAttackMovement1.AddBitmap(IDB\_HEROATTACK\_1\_1, RGB(255, 255, 255));  HeroAttackMovement1.AddBitmap(IDB\_HEROATTACK\_2\_1, RGB(255, 255, 255));  HeroAttackMovement1.AddBitmap(IDB\_HEROATTACK\_3\_1, RGB(255, 255, 255));  HeroAttackMovement1.AddBitmap(IDB\_HEROATTACK\_4\_1, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_1, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_2, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_3, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_4, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_5, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_6, RGB(255, 255, 255));  HeroRollLeft.AddBitmap(IDB\_ROLL\_LEFT\_6, RGB(255, 255, 255));  SwordRollRight.AddBitmap(IDB\_SWORDROLLRIGHT\_0, RGB(255, 255, 255));  SwordRollRight.AddBitmap(IDB\_SWORDROLLRIGHT\_1, RGB(255, 255, 255));  SwordRollRight.AddBitmap(IDB\_SWORDROLLRIGHT\_2, RGB(255, 255, 255));  SwordRollRight.AddBitmap(IDB\_SWORDROLLRIGHT\_3, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_1, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_2, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_3, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_4, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_5, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_6, RGB(255, 255, 255));  HeroRollRight.AddBitmap(IDB\_ROLL\_RIGHT\_6, RGB(255, 255, 255));  SwordRollLeft.AddBitmap(IDB\_SWORDROLLLEFT\_0, RGB(255, 255, 255));  SwordRollLeft.AddBitmap(IDB\_SWORDROLLLEFT\_1, RGB(255, 255, 255));  SwordRollLeft.AddBitmap(IDB\_SWORDROLLLEFT\_2, RGB(255, 255, 255));  SwordRollLeft.AddBitmap(IDB\_SWORDROLLLEFT\_3, RGB(255, 255, 255));  swordAttack.AddBitmap(IDB\_SWORDATTACK\_2,RGB(255, 255, 255));  swordAttack.AddBitmap(IDB\_SWORDATTACK\_3,RGB(255, 255, 255));  swordAttack.AddBitmap(IDB\_SWORDATTACK\_4,RGB(255, 255, 255));  swordAttack.AddBitmap(IDB\_SWORDATTACK\_5,RGB(255, 255, 255));  swordAttack.AddBitmap(IDB\_SWORDATTACK\_6,RGB(255, 255, 255));  swordAttack.AddBitmap(IDB\_SWORDATTACK\_7,RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_0, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_0, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_0, RGB(255, 255, 255));  SwordDashLeft.AddBitmap(IDB\_SWORDDASHLEFT\_0, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_0, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_1, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_2, RGB(255, 255, 255));  SwordDashRight.AddBitmap(IDB\_SWORDDASHRIGHT\_3, RGB(255, 255, 255));  swordAttack1.AddBitmap(IDB\_SWORDATTACK\_3\_1, RGB(255, 255, 255));  swordAttack1.AddBitmap(IDB\_SWORDATTACK\_4\_1, RGB(255, 255, 255));  swordAttack1.AddBitmap(IDB\_SWORDATTACK\_5\_1, RGB(255, 255, 255));  swordAttack1.AddBitmap(IDB\_SWORDATTACK\_6\_1, RGB(255, 255, 255));  swordAttack1.AddBitmap(IDB\_SWORDATTACK\_7\_1, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_0, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_1, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_2, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_3, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_4, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_5, RGB(255, 255, 255));  FireSwordRightAnimation.AddBitmap(IDB\_FIRESWORDRIGHT\_6, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_0, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_1, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_2, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_3, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_4, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_5, RGB(255, 255, 255));  FireSwordLeftAnimation.AddBitmap(IDB\_FIRESWORDLEFT\_6, RGB(255, 255, 255));  FireCircle.AddBitmap(IDB\_FIRECIRCLE\_0, RGB(63, 72, 204));  FireCircle.AddBitmap(IDB\_FIRECIRCLE\_1, RGB(63, 72, 204));  FireCircle.AddBitmap(IDB\_FIRECIRCLE\_2, RGB(63, 72, 204));  FireCircle.AddBitmap(IDB\_FIRECIRCLE\_3, RGB(63, 72, 204));  FireCircle.AddBitmap(IDB\_FIRECIRCLE\_4, RGB(63, 72, 204));  FireCircle.AddBitmap(IDB\_FIRECIRCLE\_5, RGB(63, 72, 204));  Fire1.AddBitmap(IDB\_FIRE\_0, RGB(63, 72, 204));  Fire1.AddBitmap(IDB\_FIRE\_1, RGB(63, 72, 204));  Fire1.AddBitmap(IDB\_FIRE\_2, RGB(63, 72, 204));  Fire1.AddBitmap(IDB\_FIRE\_3, RGB(63, 72, 204));  Fire1.AddBitmap(IDB\_FIRE\_4, RGB(63, 72, 204));  Fire1.AddBitmap(IDB\_FIRE\_5, RGB(63, 72, 204));  Fire2.AddBitmap(IDB\_FIRE\_0, RGB(63, 72, 204));  Fire2.AddBitmap(IDB\_FIRE\_1, RGB(63, 72, 204));  Fire2.AddBitmap(IDB\_FIRE\_2, RGB(63, 72, 204));  Fire2.AddBitmap(IDB\_FIRE\_3, RGB(63, 72, 204));  Fire2.AddBitmap(IDB\_FIRE\_4, RGB(63, 72, 204));  Fire2.AddBitmap(IDB\_FIRE\_5, RGB(63, 72, 204));  Fire3.AddBitmap(IDB\_FIRE\_0, RGB(63, 72, 204));  Fire3.AddBitmap(IDB\_FIRE\_1, RGB(63, 72, 204));  Fire3.AddBitmap(IDB\_FIRE\_2, RGB(63, 72, 204));  Fire3.AddBitmap(IDB\_FIRE\_3, RGB(63, 72, 204));  Fire3.AddBitmap(IDB\_FIRE\_4, RGB(63, 72, 204));  Fire3.AddBitmap(IDB\_FIRE\_5, RGB(63, 72, 204));  gain\_life.AddBitmap(IDB\_GAINLIFE\_0, RGB(255, 255, 255));  gain\_life.AddBitmap(IDB\_GAINLIFE\_1, RGB(255, 255, 255));  gain\_life.AddBitmap(IDB\_GAINLIFE\_2, RGB(255, 255, 255));  gain\_life.AddBitmap(IDB\_GAINLIFE\_3, RGB(255, 255, 255));  gain\_life.AddBitmap(IDB\_GAINLIFE\_4, RGB(255, 255, 255));  ThrowingLeft.AddBitmap(IDB\_HEROTHROWLEFT\_0, RGB(255, 255, 255));  ThrowingLeft.AddBitmap(IDB\_HEROTHROWLEFT\_1, RGB(255, 255, 255));  ThrowingLeft.AddBitmap(IDB\_HEROTHROWLEFT\_2, RGB(255, 255, 255));  ThrowingLeft.AddBitmap(IDB\_HEROTHROWLEFT\_3, RGB(255, 255, 255));  ThrowingRight.AddBitmap(IDB\_HEROTHROWRIGHT\_0, RGB(255, 255, 255));  ThrowingRight.AddBitmap(IDB\_HEROTHROWRIGHT\_1, RGB(255, 255, 255));  ThrowingRight.AddBitmap(IDB\_HEROTHROWRIGHT\_2, RGB(255, 255, 255));  ThrowingRight.AddBitmap(IDB\_HEROTHROWRIGHT\_3, RGB(255, 255, 255));  B.LoadBitmap(IDB\_B, RGB(255, 255, 255));  Infected\_UI.LoadBitmap(IDB\_INFECTED\_UI, RGB(63, 72, 204));  GainLifeUI.LoadBitmapA(IDB\_GAINLIFE\_UI, RGB(255, 255, 255));  LifeBarHead.LoadBitmap(IDB\_LIFEBARHEAD, RGB(255, 255, 255));  StartGameBar.LoadBitmap(IDB\_UI\_GAME\_START1, RGB(255, 0, 0));  WorldMap\_UI\_1.LoadBitmap(IDB\_WORLDMAP\_UI);  QuitButton.LoadBitmap(IDB\_UI\_QUIT, RGB(0, 0, 0));  Num.LoadBitmap();  Num\_Red.LoadBitmap();  Num\_Gold.LoadBitmap();  BlackMask.LoadBitmap(IDB\_BLACKMASK, RGB(27, 36, 46));  Word\_G.LoadBitmap(IDB\_WORD\_G, RGB(255, 255, 255));  Word\_Gold.LoadBitmap(IDB\_WORD\_GOLD, RGB(255, 255, 255));  ShurikanUI.LoadBitmap(IDB\_HEROBULLET\_UI, RGB(63, 72, 204));  for (vector<CMovingBitmap\*>::iterator i = LifeBarRed.begin(); i != LifeBarRed.end(); i++) (\*i)->LoadBitmap(IDB\_LIFEBAR, RGB(255, 255, 255));  //SelectMap(0);  }  int CHero::GetHeroFullHP()  {  return FullHP;  }  int CHero::GetHeroCurrentHP()  {  return CurrentHP;  }  int CHero::GetPreviousMove()  {  return PreviousMovement;  }  void CHero::setHeroAction()  {  //越下面權重越高  heroActoin = STAND;  if (isMovingLeft)  {  if(!isRolling && !isAttacking)  heroDirection = LEFT;  heroActoin = WALK;  }  if (isMovingRight)  {  if(!isRolling && !isAttacking)  heroDirection = RIGHT;  heroActoin = WALK;  }  if (isThrowing && !isAttacking)  {  heroActoin = THROW;  }  if (isAttacking)  {  heroActoin = ATTACK;  }  if (isRolling)  {  isAttacking = false;  heroActoin = ROLL;  }  if (isTalkingToNPC && isInHome)  {  bool success = false;  success = currentVillage->HeroTalkToNPC();  if (success)  heroActoin = TALK;  else  isTalkingToNPC = false;  }  }  void CHero::StandOnMove()  {  switch (heroDirection)  {  case game\_framework::LEFT:  animation1.OnMove();  animation.Reset();  sword1.OnMove();  sword.Reset();  break;  case game\_framework::RIGHT:  animation1.Reset();  animation.OnMove();  sword1.Reset();  sword.OnMove();  break;  }  HeroAttackMovement.Reset();  HeroAttackMovement1.Reset();  swordAttack.Reset();  swordAttack1.Reset();  moveRightAnimation.Reset();  HeroDashLeft.Reset();  HeroDashRight.Reset();  moveLeftAnimation.Reset();  jumpAnimation.Reset();  jumpAnimation1.Reset();  HeroRollLeft.Reset();  HeroRollRight.Reset();  SwordRollRight.Reset();  SwordRollLeft.Reset();  SwordDashRight.Reset();  SwordDashLeft.Reset();  FireSwordRightAnimation.Reset();  FireSwordLeftAnimation.Reset();  ThrowingLeft.Reset();  ThrowingRight.Reset();  }  void CHero::StandOnShow()  {  switch (heroDirection)  {  case game\_framework::LEFT:  sword1.SetTopLeft(currentMap->ScreenX(x + 17), currentMap->ScreenY(y + 30));  sword1.OnShow();  animation1.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation1.OnShow();  break;  case game\_framework::RIGHT:  sword.SetTopLeft(currentMap->ScreenX(x - 85), currentMap->ScreenY(y + 30));  sword.OnShow();  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  break;  }  }  void CHero::WalkOnMove()  {  switch (heroDirection)  {  case game\_framework::LEFT:  moveLeftAnimation.OnMove();  moveRightAnimation.Reset();  sword.Reset();  sword1.OnMove();  break;  case game\_framework::RIGHT:  moveLeftAnimation.Reset();  moveRightAnimation.OnMove();  sword.OnMove();  sword1.Reset();  break;  }  animation.Reset();  animation1.Reset();  HeroAttackMovement.Reset();  HeroAttackMovement1.Reset();  swordAttack.Reset();  swordAttack1.Reset();  HeroDashLeft.Reset();  HeroDashRight.Reset();  jumpAnimation.Reset();  jumpAnimation1.Reset();  HeroRollLeft.Reset();  HeroRollRight.Reset();  SwordRollRight.Reset();  SwordRollLeft.Reset();  SwordDashRight.Reset();  SwordDashLeft.Reset();  FireSwordRightAnimation.Reset();  FireSwordLeftAnimation.Reset();  ThrowingLeft.Reset();  ThrowingRight.Reset();  const int STEP\_SIZE = 10;  switch (heroDirection)  {  case game\_framework::LEFT:  for (int i = 0; i < STEP\_SIZE; i++) {  if (currentMap->isSpace(GetX1() - 1, GetY1()) && currentMap->isSpace(GetX1() - 1, GetY2()) && !currentMap->isDoor(GetX1() - 1, GetY1()) && !currentMap->isDoor(GetX1() - 1, GetY2())) // 當x座標還沒碰到牆  {  x -= 1; //正常走  }  }  break;  case game\_framework::RIGHT:  for (int i = 0; i < STEP\_SIZE; i++)  {  if (currentMap->isSpace(GetX2() + 1, GetY1()) && currentMap->isSpace(GetX2() + 1, GetY2()) && !currentMap->isDoor(GetX2() + 1, GetY1()) && !currentMap->isDoor(GetX2() + 1, GetY2())) // 當x座標還沒碰到牆  {  x += 1; //正常走  }  }  break;  }  }  void CHero::WalkOnShow()  {  switch (heroDirection)  {  case game\_framework::LEFT: //向左走  sword1.SetTopLeft(currentMap->ScreenX(x + 17), currentMap->ScreenY(y + 30));  sword1.OnShow();  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  break;  case game\_framework::RIGHT: //向右走  sword.SetTopLeft(currentMap->ScreenX(x - 75), currentMap->ScreenY(y + 30));  sword.OnShow();  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  break;  }  }  void CHero::RollOnMove()  {  const int STEP\_SIZE = 15;  if (HeroRollLeft.IsFinalBitmap() || HeroRollRight.IsFinalBitmap())  {  isRolling = false;  }  switch (heroDirection)  {  case game\_framework::LEFT:  for (int i = 0; i < STEP\_SIZE; i++)  {  if (currentMap->isSpace(GetX1() - 1, GetY1()) && currentMap->isSpace(GetX1() - 1, GetY2()) && !currentMap->isDoor(GetX1() - 1, GetY1()) && !currentMap->isDoor(GetX1() - 1, GetY2())) // 當x座標還沒碰到牆  {  x -= 1; //正常走  }  }  break;  case game\_framework::RIGHT:  for (int i = 0; i < STEP\_SIZE; i++)  {  if (currentMap->isSpace(GetX2() + 1, GetY1()) && currentMap->isSpace(GetX2() + 1, GetY2()) && !currentMap->isDoor(GetX2() + 1, GetY1()) && !currentMap->isDoor(GetX2() + 1, GetY2())) // 當x座標還沒碰到牆  {  x += 1; //正常走  }  }  break;  }  switch (heroDirection)  {  case game\_framework::LEFT:  HeroRollLeft.OnMove();  HeroRollRight.Reset();  SwordRollRight.Reset();  SwordRollLeft.OnMove();  break;  case game\_framework::RIGHT:  HeroRollLeft.Reset();  HeroRollRight.OnMove();  SwordRollRight.OnMove();  SwordRollLeft.Reset();  break;  }  animation.Reset();  animation1.Reset();  sword.Reset();  sword1.Reset();  HeroAttackMovement.Reset();  HeroAttackMovement1.Reset();  swordAttack.Reset();  swordAttack1.Reset();  moveRightAnimation.Reset();  HeroDashLeft.Reset();  HeroDashRight.Reset();  moveLeftAnimation.Reset();  jumpAnimation.Reset();  jumpAnimation1.Reset();  SwordDashRight.Reset();  SwordDashLeft.Reset();  FireSwordRightAnimation.Reset();  FireSwordLeftAnimation.Reset();  ThrowingLeft.Reset();  ThrowingRight.Reset();  }  void CHero::RollOnShow()  {  switch (heroDirection)  {  case game\_framework::LEFT:  SwordRollLeft.SetTopLeft(currentMap->ScreenX(x - 20), currentMap->ScreenY(y - 5));  SwordRollLeft.OnShow();  HeroRollLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HeroRollLeft.OnShow();  break;  case game\_framework::RIGHT:  SwordRollRight.SetTopLeft(currentMap->ScreenX(x - 25), currentMap->ScreenY(y - 5));  SwordRollRight.OnShow();  HeroRollRight.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HeroRollRight.OnShow();  break;  }  }  void CHero::AttackOnMove()  {  if (HeroAttackMovement.IsFinalBitmap() || HeroAttackMovement1.IsFinalBitmap())  {  isAttacking = false;  }  if (!isInHome)  {  switch (heroDirection)  {  case game\_framework::LEFT:  currentWild->SetHeroAttackRange(GetCenterX() - AttackRange - 50, GetCenterX() + AttackRange, GetCenterY() - 50, GetCenterY() + 50);  currentWild->SetHeroAttackRange(GetCenterX() - AttackRange, GetCenterX() + AttackRange + 50, GetCenterY() - 50, GetCenterY() + 50);  break;  case game\_framework::RIGHT:  currentWild->SetHeroAttackRange(GetCenterX() - AttackRange, GetCenterX() + AttackRange + 50, GetCenterY() - 50, GetCenterY() + 50);  break;  }  currentWild->AttackByHero(heroAttackDamage);  }  switch (heroDirection)  {  case game\_framework::LEFT:  HeroAttackMovement.Reset();  HeroAttackMovement1.OnMove();  swordAttack.Reset();  swordAttack1.OnMove();  FireSwordLeftAnimation.OnMove();  FireSwordRightAnimation.Reset();  break;  case game\_framework::RIGHT:  HeroAttackMovement.OnMove();  HeroAttackMovement1.Reset();  swordAttack.OnMove();  swordAttack1.Reset();  FireSwordRightAnimation.OnMove();  FireSwordLeftAnimation.Reset();  break;  }  HeroRollRight.Reset();  SwordRollRight.Reset();  HeroRollLeft.Reset();  SwordRollLeft.Reset();  animation.Reset();  animation1.Reset();  sword.Reset();  sword1.Reset();  moveRightAnimation.Reset();  HeroDashLeft.Reset();  HeroDashRight.Reset();  moveLeftAnimation.Reset();  jumpAnimation.Reset();  jumpAnimation1.Reset();  SwordDashRight.Reset();  SwordDashLeft.Reset();  ThrowingLeft.Reset();  ThrowingRight.Reset();  }  void CHero::AttackOnShow()  {  switch (heroDirection)  {  case game\_framework::LEFT:  if (HasFireStone)  {  FireSwordLeftAnimation.SetTopLeft(currentMap->ScreenX(x - 90), currentMap->ScreenY(y - 10));  FireSwordLeftAnimation.OnShow();  }  swordAttack1.SetTopLeft(currentMap->ScreenX(x - 95), currentMap->ScreenY(y + 10));  swordAttack1.OnShow();  HeroAttackMovement1.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HeroAttackMovement1.OnShow();  break;  case game\_framework::RIGHT:  if (HasFireStone)  {  FireSwordRightAnimation.SetTopLeft(currentMap->ScreenX(x - 50), currentMap->ScreenY(y - 10));  FireSwordRightAnimation.OnShow();  }  swordAttack.SetTopLeft(currentMap->ScreenX(x - 40), currentMap->ScreenY(y + 10));  swordAttack.OnShow();  HeroAttackMovement.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HeroAttackMovement.OnShow();  break;  }  }  void CHero::ThrowOnMove()  {  if (ThrowingLeft.IsFinalBitmap() || ThrowingRight.IsFinalBitmap())  {  isThrowing = false;  }  if (!isInHome)  {  switch (heroDirection)  {  case game\_framework::LEFT:  if (ThrowingLeft.IsFinalBitmap() && ShurikanNumber > 0)  {  allShurikan.push\_back(new Shurikan(currentMap, GetX1() - 10, GetY1() + 40, -15));  allShurikan.back()->LoadBitmap();  ShurikanNumber -= 1;  }  break;  case game\_framework::RIGHT:  if (ThrowingRight.IsFinalBitmap() && ShurikanNumber > 0)  {  allShurikan.push\_back(new Shurikan(currentMap, GetX2(), GetY1() + 40, 15));  allShurikan.back()->LoadBitmap();  ShurikanNumber -= 1;  }  break;  }  }  switch (heroDirection)  {  case game\_framework::LEFT:  ThrowingRight.Reset();  ThrowingLeft.OnMove();  break;  case game\_framework::RIGHT:  ThrowingRight.OnMove();  ThrowingLeft.Reset();  break;  }  HeroAttackMovement.Reset();  HeroAttackMovement1.Reset();  HeroRollRight.Reset();  SwordRollRight.Reset();  HeroRollLeft.Reset();  SwordRollLeft.Reset();  animation.Reset();  animation1.Reset();  sword.Reset();  sword1.Reset();  moveRightAnimation.Reset();  HeroDashLeft.Reset();  HeroDashRight.Reset();  moveLeftAnimation.Reset();  jumpAnimation.Reset();  jumpAnimation1.Reset();  SwordDashRight.Reset();  SwordDashLeft.Reset();  swordAttack.Reset();  FireSwordRightAnimation.Reset();  swordAttack1.Reset();  FireSwordLeftAnimation.Reset();  }  void CHero::ThrowOnShow()  {  switch (heroDirection)  {  case game\_framework::LEFT:  ThrowingLeft.SetTopLeft(currentMap->ScreenX(x - 30), currentMap->ScreenY(y + 10));  ThrowingLeft.OnShow();  break;  case game\_framework::RIGHT:  ThrowingRight.SetTopLeft(currentMap->ScreenX(x - 40), currentMap->ScreenY(y + 10));  ThrowingRight.OnShow();  break;  }  }  void CHero::OnMove()  {  if (isInHome)//在home時  {  CurrentHP = FullHP;  }  else  {  ClearedStage = currentWild->GetisStageClear();  }  const int STEP\_SIZE = 10;  gain\_life.OnMove();  FireSwordRightAnimation.OnMove();  FireSwordLeftAnimation.OnMove();  FireCircle.OnMove();  Fire1.OnMove();  Fire2.OnMove();  Fire3.OnMove();  if (SpecialEffectCount == 0) SpecialEffect = 0; //被攻擊3次後，特殊效果消失  if (ShowGoldDelayCount > 0) ShowGoldDelayCount--;  if (AttackDelayCount != 0) AttackDelayCount--; //攻速  if (RollDelayCount != 0) RollDelayCount--; //翻滾  if (InvincibleDelayCount != 0) InvincibleDelayCount--; //無敵時間  if (DashColdDown != 0) DashColdDown--; //衝刺  if (MoveDelayCount != 0) MoveDelayCount--; //紀錄上個動作的保持時間  if (MoveDelayCount == 0) SetPreviousMove(0); //抹除上個動作紀錄  if (GainLifeDelayCount > 0) GainLifeDelayCount--;  if (PoisonDelayCount != 0)  {  if (PoisonDelayCount % 3 == 0) CurrentHP -= 1;  PoisonDelayCount--;  }  else Poison = false;  if (InvincibleDelayCount == 0) isInvincible = false;  vector<Shurikan\*>::iterator iter = allShurikan.begin(); //手裡劍  while (iter != allShurikan.end())  {  if ((\*iter)->isDelet())  {  delete \*iter;  iter = allShurikan.erase(iter);  }  else  iter++;  }  for (vector<Shurikan\*>::iterator i = allShurikan.begin(); i != allShurikan.end(); i++) (\*i)->OnMove();  iter = allShurikan.begin();  while (iter != allShurikan.end())  {  currentWild->SetHeroAttackRange((\*iter)->GetX1(),(\*iter)->GetX2(),(\*iter)->GetY1(),(\*iter)->GetY2());  currentWild->AttackByHero(heroAttackDamage);  iter++;  }  if (GainHealthDelayCount != 0) //持續回血特效  {  GainHealthDelayCount--;  if (CurrentHP <= FullHP - 1)  {  CurrentHP += 1;  }  else if ((FullHP - 1 <= CurrentHP) && (CurrentHP <= FullHP))  {  CurrentHP = FullHP;  }  }  setHeroAction();  switch (heroActoin)  {  case game\_framework::STAND:  StandOnMove();  break;  case game\_framework::WALK:  WalkOnMove();  break;  case game\_framework::RUN:  break;  case game\_framework::ROLL:  RollOnMove();  break;  case game\_framework::ATTACK:  AttackOnMove();  break;  case game\_framework::THROW:  ThrowOnMove();  break;  case game\_framework::TALK:  StandOnMove();  break;  }  // 重力  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2() + 1) && currentMap->isSpace(GetX2(), GetY2() + 1)) { // 當y座標還沒碰到地板  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  }  else {  isMovingUp = false;  floor = currentMap->GetBlockY(GetY2() + 1) - 1 - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  if (!isRolling && !isInvincible && !isInHome)  {  bleed = AttackByEnemy();  currentWild->SetHeroXY(GetX1(), GetX2(), GetY1(), GetY2());  if (ShowGoldDelayCount > 0)  {  HeroGetItem();  }  else  {  GetGold = HeroGetItem(); //重設顯示數字  }    }  currentMap->SetSXSY(GetCenterX() - SIZE\_X / 2, GetCenterY() - SIZE\_Y / 2);  if (isInHome)  {  currentVillage->setHeroState(GetX1(), GetX2(), GetY1(), GetY2(), FullHP, Gold, heroAttackDamage, HeroLevel);  currentVillage->OnMove();  }  else  {  currentWild->setHeroState(GetX1(), GetX2(), GetY1(), GetY2(), FullHP, Gold, heroAttackDamage, HeroLevel);  currentWild->OnMove();  }  }  void CHero::SetMovingDown(bool flag)  {  isMovingDown = flag;  }  void CHero::SetMovingLeft(bool flag)  {  isMovingLeft = flag;  }  void CHero::SetMovingRight(bool flag)  {  isMovingRight = flag;  }  void CHero::SetMovingUp(bool flag)  {  if (isMovingUp == false && y == floor)  {  isMovingUp = flag;  rising = true;  }  }  void CHero::SetTalkingToNPC(bool flag)  {  if (isTalkingToNPC == false)  {  isTalkingToNPC = flag;  }  }  void CHero::SetEndTalking()  {  isTalkingToNPC = false;  }  void CHero::SetHeroAttack(bool flag)  {  if (isAttacking == false)  {  isAttacking = flag;  }  }  void CHero::SetHeroRoll (bool flag)  {  if (isRolling == false)  {  isRolling = flag;  }  }  void CHero::SetHeroThrow(bool flag)  {  if (isThrowing == false)  {  isThrowing = flag;  }  }  void CHero::SetXY(int nx, int ny)  {  x = nx; y = ny;  }  void CHero::SetPreviousMove(int move)  {  PreviousMovement = move;  }  void CHero::SetMoveDelayCount(int delay)  {  MoveDelayCount = delay;  }  void CHero::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x - 250);  int yMove = currentMap->ScreenY(y - 201);  int counter = xMove;  float lengthOfLifeBar = ((float)CurrentHP / (float)FullHP) \* 300; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBarRed.begin(); i != LifeBarRed.end(); i++)  {  (\*i)->SetTopLeft(xMove, yMove);  if((\*i)->Left() < 45+lengthOfLifeBar) (\*i)->ShowBitmap();  xMove += 3;  }  }  void CHero::OnShow()  {  //處理UI的顯示  if (isInHome) currentVillage->OnShow();  else currentWild->OnShow();  for (vector<Shurikan\*>::iterator i = allShurikan.begin(); i != allShurikan.end(); i++)  (\*i)->OnShow();  LifeBarHead.SetTopLeft(currentMap->ScreenX(x-290), currentMap->ScreenY(y-205)); //顯示血條  LifeBarHead.ShowBitmap(); //顯示血條  ShurikanUI.SetTopLeft(currentMap->ScreenX(x - 290), currentMap->ScreenY(y - 155));  ShurikanUI.ShowBitmap();  ShowNumber(1, ShurikanNumber, currentMap->ScreenX(x - 240), currentMap->ScreenY(y - 145));  if (GainHealthDelayCount != 0)//顯示回血特效  {  GainLifeUI.SetTopLeft(currentMap->ScreenX(x - 250), currentMap->ScreenY(y - 175));  GainLifeUI.ShowBitmap();  }  if (PoisonDelayCount != 0)  {  Infected\_UI.SetTopLeft(currentMap->ScreenX(x - 220), currentMap->ScreenY(y - 175));  Infected\_UI.ShowBitmap();  }  changeLifeBarLength();  ShowNumber(3, Gold, currentMap->ScreenX(x + 250), currentMap->ScreenY(y - 195));  Word\_Gold.SetTopLeft(currentMap->ScreenX(x + 200), currentMap->ScreenY(y - 195));  Word\_Gold.ShowBitmap();  if (isInHome && heroActoin == TALK)  {  ShowNumber(1, Gold, currentMap->ScreenX(x + 190), currentMap->ScreenY(y - 175));  ShowNumber(1, HeroLevel, currentMap->ScreenX(x + 90), currentMap->ScreenY(y - 175));  ShowNumber(1, FullHP, currentMap->ScreenX(x + 110), currentMap->ScreenY(y - 135));  ShowNumber(1, heroAttackDamage, currentMap->ScreenX(x + 140), currentMap->ScreenY(y - 110));  }  ShowNumber(2, CurrentHP, currentMap->ScreenX(x - 280), currentMap->ScreenY(y - 170));  if(bleed != 0 && InvincibleDelayCount!= 0) ShowNumber(2,bleed, currentMap->ScreenX(x - 50), currentMap->ScreenY(y - 50));  if (ShowGoldDelayCount > 0)  {  ShowNumber(3, GetGold, currentMap->ScreenX(x - 50), currentMap->ScreenY(y - 30));  Word\_G.SetTopLeft(currentMap->ScreenX(x + 2), currentMap->ScreenY(y - 30));  Word\_G.ShowBitmap();  }  //處理劍的顯示  if (HasFireStone)  {  switch (SpecialEffectCount)  {  case 3:  Fire1.SetTopLeft(currentMap->ScreenX(x - 30), currentMap->ScreenY(y));  Fire2.SetTopLeft(currentMap->ScreenX(x + 45), currentMap->ScreenY(y));  Fire3.SetTopLeft(currentMap->ScreenX(x + 10), currentMap->ScreenY(y - 45));  Fire1.OnShow();  Fire2.OnShow();  Fire3.OnShow();  break;  case 2:  Fire1.SetTopLeft(currentMap->ScreenX(x - 30), currentMap->ScreenY(y));  Fire2.SetTopLeft(currentMap->ScreenX(x + 45), currentMap->ScreenY(y));  Fire1.OnShow();  Fire2.OnShow();  break;  case 1:  Fire1.SetTopLeft(currentMap->ScreenX(x - 30), currentMap->ScreenY(y));  Fire1.OnShow();  break;  default:  break;  }  FireCircle.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 5));  FireCircle.OnShow();  }  switch (heroActoin)  {  case game\_framework::STAND:  StandOnShow();  break;  case game\_framework::WALK:  WalkOnShow();  break;  case game\_framework::RUN:  break;  case game\_framework::ROLL:  RollOnShow();  break;  case game\_framework::ATTACK:  AttackOnShow();  break;  case game\_framework::THROW:  ThrowOnShow();  break;  case game\_framework::TALK:  StandOnShow();  default:  break;  }  if (GainLifeDelayCount != 0 || GainHealthDelayCount != 0)  {  gain\_life.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 25));  gain\_life.OnShow();  }  if (isInHome)  {  StartGameBar.SetTopLeft(currentMap->ScreenX(x - 100), currentMap->ScreenY(y + 150));  StartGameBar.ShowBitmap();  }  if (isSelectingMap)  {  WorldMap\_UI\_1.SetTopLeft(currentMap->ScreenX(x - 270), currentMap->ScreenY(y - 170));  WorldMap\_UI\_1.ShowBitmap();  }  if (ClearedStage)  {  BlackMask.SetTopLeft(0, 0);  BlackMask.ShowBitmap();  B.SetTopLeft(50, 30);  B.ShowBitmap();  QuitButton.SetTopLeft(currentMap->ScreenX(x + 100), currentMap->ScreenY(y + 170));  QuitButton.ShowBitmap();  }  }  void CHero::SetHeroHP(int inputHP)  {  FullHP = inputHP;  }  int CHero::AttackByEnemy()  {  int hp = CurrentHP;  bool RecordedPoison = Poison;  currentWild->AttackByEnemy(&CurrentHP, &Poison);  if (CurrentHP != hp)  {  if (SpecialEffectCount > 0)  {  SpecialEffectCount -= 1;  if (SpecialEffectCount == 0 && HasFireStone) //火焰石效果消失  {  heroAttackDamage /= 2; //回歸原本的攻擊力  HasFireStone = false;  }  }  isInvincible = true;  InvincibleDelayCount = 30;  }  if (RecordedPoison != Poison)  {  PoisonDelayCount = 300;  }  return hp - CurrentHP;  }  int CHero::HeroGetItem()  {  int Coin = Gold;  int SpecialEffectDectect = SpecialEffect;  int RecordedHP = CurrentHP;  currentWild->HeroGetItem(&Gold, &SpecialEffect, &SpecialEffectCount, &CurrentHP, FullHP, &ShurikanNumber);  if (RecordedHP < CurrentHP) GainLifeDelayCount = 45;  if (Coin < Gold && (Gold - Coin)>=10)  {  ShowGoldDelayCount = 30;  }  if (SpecialEffectDectect != SpecialEffect) //取得特殊效果的道具  {  if (SpecialEffect == 1 && !HasFireStone)  {  heroAttackDamage \*= 2; //火焰石攻擊力加倍  HasFireStone = true;  }  else if (SpecialEffect == 2)  {  GainHealthDelayCount = 1000;  }  }  return Gold - Coin;  }  void CHero::ShowNumber(int color, int num, int x, int y)  {  if (color == 1)  {  Num.SetInteger(num);  Num.SetTopLeft(x, y);  Num.ShowBitmap();  }  else if(color == 2)  {  Num\_Red.SetInteger(num);  Num\_Red.SetTopLeft(x, y);  Num\_Red.ShowBitmap();  }  else  {  Num\_Gold.SetInteger(num);  Num\_Gold.SetTopLeft(x, y);  Num\_Gold.ShowBitmap();  }  }  void CHero::HeroLevelUp()  {  FullHP += 5;  heroAttackDamage += 5;  HeroLevel += 1;  Gold -= 20;  }  void CHero::SelectMap(int MapNumber)  {  if (MapNumber == 0)  isInHome = true;  else  isInHome = false;  switch (MapNumber)  {  case 0:  if(currentVillage!=NULL)  delete currentVillage;  currentVillage = new gameMap\_village();  currentVillage->LoadBitmap();  CAudio::Instance()->Play(8, true);  currentMap = currentVillage;    break;  case 1:    if(currentWild!=NULL)  delete currentWild;  currentWild = new gameMap\_wild("level\_1.txt");  currentWild->LoadBitmap();  CAudio::Instance()->Stop(8);  CAudio::Instance()->Play(6, false);  currentMap = currentWild;  break;    case 2:  if(currentWild!=NULL)  delete currentWild;  currentWild = new gameMap\_wild("level\_2.txt");  currentWild->LoadBitmap();  CAudio::Instance()->Stop(8);  CAudio::Instance()->Play(6, false);  currentMap = currentWild;  break;  case 3:  if (currentWild != NULL)  delete currentWild;  currentWild = new gameMap\_wild("level\_3.txt");  currentWild->LoadBitmap();  CAudio::Instance()->Stop(8);  CAudio::Instance()->Play(6, false);  currentMap = currentWild;  break;  case 4:  if (currentWild != NULL)  delete currentWild;  currentWild = new gameMap\_wild("level\_4.txt");  currentWild->LoadBitmap();  CAudio::Instance()->Stop(8);  CAudio::Instance()->Play(6, false);  currentMap = currentWild;  break;  }  x = 35;  y = 0;  }  void CHero::ResetHeroState()  {  ClearedStage = false;  CurrentHP = FullHP;  isInHome = true;  SelectMap(0);  if (HasFireStone && SpecialEffectCount != 0) heroAttackDamage /= 2;  HasFireStone = false;  SpecialEffect = 0;  SpecialEffectCount = 0;  GainHealthDelayCount = 0;  ShurikanNumber = 10;  Poison = false;  PoisonDelayCount = 0;  x = 35;  y = 0;  }  void CHero::OnLButtonDown(int Mx, int My)  {  if (heroActoin == TALK)  {  currentVillage->OnLButtonDown(Mx, My, &isTalkingToNPC);  if ((Mx <= 619) && (My <= 184) && (Mx >= 524) && (My >= 144) && Gold >= 20) HeroLevelUp();  }  if (isInHome) //在村莊  {  if (isSelectingMap) //按GAME\_START後選擇地圖畫面  {  if ((Mx <= 612) && (My <= 87) && (Mx >= 570) && (My >= 48)) isSelectingMap = false; //右上角xx  if ((Mx <= 222) && (My <= 260) && (Mx >= 187) && (My >= 222)) //第一章地圖  {  SelectMap(1);  isSelectingMap = false;  }  if ((Mx <= 300) && (My <= 261) && (Mx >= 259) && (My >= 219)) //第二章地圖  {  SelectMap(2);  isSelectingMap = false;  }  if ((Mx <= 377) && (My <= 253) && (Mx >= 321) && (My >= 204)) //第三章地圖  {  SelectMap(3);  isSelectingMap = false;  }  if ((Mx <= 449) && (My <= 254) && (Mx >= 395) && (My >= 205)) //第三章地圖  {  SelectMap(4);  isSelectingMap = false;  }  }  else  {  if ((Mx <= 437) && (My <= 415) && (Mx >= 188) && (My >= 355)) isSelectingMap = true;  }  }  if (ClearedStage)  {  if ((Mx <= 487) && (My <= 422) && (Mx >= 396) && (My >= 380))  {  ResetHeroState();  }  }  }  bool CHero::GetHeroIsRolling()  {  return isRolling;  }  bool CHero::isAlive()  {  if (CurrentHP <= 0)  {  ResetHeroState();  Gold = Gold / 2;  return false;  }  return true;  }  void CHero::AddHeroGold()  {  Gold += 100;  }  /////////////////////////////////////////////////////////////////////////////  // HeroBullet  /////////////////////////////////////////////////////////////////////////////  HeroBullet::HeroBullet(gameMap\* point, int nx, int ny, int step)  {  x = nx;  y = ny;  currentMap = point;  distance = 0;  STEP\_SIZE = step;  animation.SetDelayCount(2);  }  HeroBullet::~HeroBullet() {}  int HeroBullet::GetX1()  {  return x;  }  int HeroBullet::GetY1()  {  return y;  }  int HeroBullet::GetX2()  {  return x + animation.Width();  }  int HeroBullet::GetY2()  {  return y + animation.Height();  }  bool HeroBullet::isDelet()  {  if (distance > 500)  {  return true;  }  return false;  }  void HeroBullet::OnMove()  {  animation.OnMove();  x += STEP\_SIZE;  if (STEP\_SIZE > 0)  {  distance += STEP\_SIZE;  }  else  {  distance -= STEP\_SIZE;  }  }  void HeroBullet::OnShow()  {  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  }  /////////////////////////////////////////////////////////////////////////////  // Shurikan : HeroBullet class  /////////////////////////////////////////////////////////////////////////////  Shurikan::Shurikan(gameMap\* point, int nx, int ny, int step) : HeroBullet(point, nx, ny, step) {}  Shurikan::~Shurikan() {}  void Shurikan::LoadBitmap()  {  animation.AddBitmap(IDB\_SHURIKAN\_0, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SHURIKAN\_1, RGB(63, 72, 204));  }  } |
| CEnemy.h |
| namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // 這個class提供可以用鍵盤或滑鼠控制的敵人  // 看懂就可以改寫成自己的程式了  /////////////////////////////////////////////////////////////////////////////  enum CEnemy\_Action  {  STAND\_LEFT,  MOVE\_LEFT,  ATTACK\_LEFT,  STAND\_RIGHT,  MOVE\_RIGHT,  ATTACK\_RIGHT,  GET\_HIT,  DEAD  };  enum Item\_Type  {  BRONZE\_COIN,  SILVER\_COIN,  GOLDEN\_COIN  };  class gameMap;  class CEnemy  {  public:  CEnemy(gameMap\* pointer, int x, int y);  virtual ~CEnemy();  virtual void LoadBitmap() = 0; // 載入圖形  virtual void OnMove() = 0; // 移動敵人  virtual void OnShow() = 0; // 將敵人圖形貼到畫面  virtual int GetX1() = 0;  virtual int GetY1() = 0;  virtual int GetX2() = 0;  virtual int GetY2() = 0;  void SetHeroXY(int x1, int x2, int y1, int y2); // 設定英雄位置  void SetHeroAttackRange(int x1, int x2, int y1, int y2); // 設定英雄攻擊範圍  virtual void GetAttack(const int damage) = 0; // 被主角攻擊  virtual void AttackByEnemy(int\* heroHP, bool \*Poison) = 0; // 敵人攻擊主角  virtual bool isDead() = 0; // 判斷敵人死了沒  virtual string GetEnemyType() = 0; // 回傳敵人種類  void SetEnemyXY(int SetX, int SetY); // 設定敵人所在位置  protected:  int x, y; // 敵人左上角座標  gameMap\* currentMap; // 所在地圖  map<string, int> hero;  map<string, int> heroAttackRange;  vector<CMovingBitmap\*> LifeBar\_1; // 血條  CMovingBitmap LifeBar\_0;  int ShowLifeBarDelayCount;  int enemyHP;  int FullHP;  };  class bullet  {  public:  bullet(gameMap\* point, int nx, int ny, int step);  ~bullet();  int GetX1(); // 子彈左上角 x 座標  int GetY1(); // 子彈左上角 y 座標  int GetX2(); // 子彈右下角 x 座標  int GetY2(); // 子彈右下角 y 座標  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  bool isDelet();  protected:  CAnimation animation;  int x;  int y;  gameMap \*currentMap;  int distance;  int STEP\_SIZE;  };  class bullet\_sunFlower : public bullet  {  public:  bullet\_sunFlower(gameMap\* point, int nx, int ny, int step);  ~bullet\_sunFlower();  void LoadBitmap(); // 載入圖形  };  class CEnemy\_sunFlower : public CEnemy  {  public:  CEnemy\_sunFlower(gameMap\* pointer, int x, int y);  ~CEnemy\_sunFlower();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \*Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation animationLeft; // 敵人的動畫(左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation AttackRightAnimation;// 攻擊右邊  CAnimation AttackLeftAnimation; // 攻擊左邊  CAnimation jumpAnimation; // 跳躍動畫  CAnimation DeadAnimation; // 死亡動畫  CAnimation GetHitAnimation; // 被打到的動畫  CAnimation HitAnimation; // 打擊效果  string EnemyType = "Sun\_Flower";  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  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  CEnemy\_Action DetectHero(CEnemy\_Action state);  int attackDelay;  int attackDelayCount;  CEnemy\_Action state;  vector<bullet\_sunFlower\*> allBullet;  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  };  class CEnemy\_Cloud : public CEnemy  {  public:  CEnemy\_Cloud(gameMap\* pointer, int x, int y);  ~CEnemy\_Cloud();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \*Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation animationLeft; // 敵人的動畫(左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation AttackRightAnimation;// 攻擊右邊  CAnimation AttackLeftAnimation; // 攻擊左邊  CAnimation jumpAnimation; // 跳躍動畫  CAnimation DeadAnimation; // 死亡動畫  CAnimation HitAnimation; // 打擊效果  CAnimation LightningCloud;  CAnimation TrackLightningCloud1;  string EnemyType = "Cloud Boss";  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  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  CEnemy\_Action DetectHero(CEnemy\_Action state);  int attackDelay;  int attackDelayCount;  CEnemy\_Action state;  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  int AttackFlag = false;  int HeroXArray[4];  void SetHeroXArray(int ArrayNumber, int ArrayValue);  int HeroXCounter = 0;  int HeroXTrackCounter = 0;  int SetHeroXTrackCounter = 120;  int TrackLightningDelayCount;  int SetTrackLightningDelayCount = 80;  void ResetTrackLightningCloudAnimation(int CloudNumber);  void ShowTrackLightningCloud(int CloudNumber);  void OnMoveTrackLightningCloud(int CloudNumber);  bool LightningActivated = false;  int LightningStrikeDelayCount = 10;  int ShowLightningCloudNumber = 0;  bool LightningStrike = false;  bool AttackAudio\_1 = false;  bool AttackAudio\_2 = false;  };  class CEnemy\_GasRobot : public CEnemy  {  public:  CEnemy\_GasRobot(gameMap\* pointer, int x, int y);  ~CEnemy\_GasRobot();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \*Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation animationLeft; // 敵人的動畫(左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation AttackRightAnimation;// 攻擊右邊  CAnimation AttackLeftAnimation; // 攻擊左邊  CAnimation jumpAnimation; // 跳躍動畫  CAnimation DeadAnimation; // 死亡動畫  CAnimation GasAnimation;  CAnimation GasRobotFireRightAnimation;  CAnimation GasRobotFireLeftAnimation;  CAnimation HitAnimation; // 打擊效果  CAnimation CountDownNumber;  CAnimation ExplosionAnimation;  string EnemyType = "RobotA";  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  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  CEnemy\_Action DetectHero(CEnemy\_Action state);  int attackDelay;  int attackDelayCount;  CEnemy\_Action state;  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  bool AttackFlag = false;  bool ShowGas = false;  bool AttackAudio = false;  bool DeadAudio = false;  bool ExplosionAudio = false;  };  class CEnemy\_RobotA : public CEnemy  {  public:  CEnemy\_RobotA(gameMap\* pointer, int x, int y);  ~CEnemy\_RobotA();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \* Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation animationLeft; // 敵人的動畫(左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation AttackRightAnimation;// 攻擊右邊  CAnimation AttackLeftAnimation; // 攻擊左邊  CAnimation jumpAnimation; // 跳躍動畫  CAnimation DeadAnimation; // 死亡動畫  CAnimation GetHitAnimation; // 被打到的動畫  CAnimation HitAnimation; // 打擊效果  CAnimation AttackVrfxRight;  CAnimation AttackVrfxLeft;  string EnemyType = "RobotA";  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  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  CEnemy\_Action DetectHero(CEnemy\_Action state);  int attackDelay;  int attackDelayCount;  CEnemy\_Action state;  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  const int DASH\_SIZE = 100;  bool AttackFlag = false;  bool AttackAudio = false;  bool DeadAudio = false;  bool ExplosionAudio = false;  };  class CEnemy\_Pigeon : public CEnemy  {  public:  CEnemy\_Pigeon(gameMap\* pointer, int x, int y);  ~CEnemy\_Pigeon();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \* Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation animationLeft; // 敵人的動畫(左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation AttackRightAnimation;// 攻擊右邊  CAnimation AttackLeftAnimation; // 攻擊左邊  CAnimation jumpAnimation; // 跳躍動畫  CAnimation DeadAnimation; // 死亡動畫  CAnimation GetHitAnimation; // 被打到的動畫  CAnimation HitAnimation; // 打擊效果  CAnimation AttackVrfx;  string EnemyType = "Pigeon";  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  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  CEnemy\_Action DetectHero(CEnemy\_Action state);  int attackDelay;  int attackDelayCount;  CEnemy\_Action state;  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  const int DASH\_SIZE = 100;  bool AttackFlag = false;  bool recorded = false;  int RecordedX1;  int RecordedY1;  bool FireBall\_Audio\_1 = false;  bool FireBall\_Audio\_2 = false;  };  class CEnemy\_Scorpoin : public CEnemy  {  public:  CEnemy\_Scorpoin(gameMap\* pointer, int x, int y);  ~CEnemy\_Scorpoin();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \*Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation animationLeft; // 敵人的動畫(左)  CAnimation moveRightAnimation; // 向右移動動畫  CAnimation moveLeftAnimation; // 向左移動動畫  CAnimation AttackRightAnimation;// 攻擊右邊  CAnimation AttackLeftAnimation; // 攻擊左邊  CAnimation jumpAnimation; // 跳躍動畫  CAnimation DeadAnimation; // 死亡動畫  CAnimation HitAnimation; // 打擊效果  string EnemyType = "Scorpoin";  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  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  CEnemy\_Action DetectHero(CEnemy\_Action state);  int attackDelay;  int attackDelayCount;  CEnemy\_Action state;  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  bool AttackFlag = false;  bool AttackFlag\_2 = false;  bool DeadAudio = false;  };  class CEnemy\_Cactus : public CEnemy  {  public:  CEnemy\_Cactus(gameMap\* pointer, int x, int y);  ~CEnemy\_Cactus();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \*Poison);  bool isDead();  string GetEnemyType();  private:  CAnimation animation; // 敵人的動畫  CAnimation AttackAnimation;// 攻擊  CAnimation DeadAnimation;  CAnimation GetHitAnimation; // 被打到的動畫  CAnimation HitAnimation; // 打擊效果  string EnemyType = "Cactus";  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage; //敵人攻擊力  int AttackDelayCount; // 攻擊頻率  int SetAttackDelay;  bool AttackFlag; // 等於true(發出尖刺動畫出現)時才攻擊到主角  bool ReadyToAttack; // 開始攻擊動作  bool GetHit = false;  int GetHitDelayCount = 0;  void changeLifeBarLength(); // 改變生命條的長度  };  class CEnemy\_Statue : public CEnemy  {  public:  CEnemy\_Statue(gameMap\* pointer, int x, int y);  ~CEnemy\_Statue();  int GetX1(); // 敵人左上角 x 座標  int GetY1(); // 敵人左上角 y 座標  int GetX2(); // 敵人右下角 x 座標  int GetY2(); // 敵人右下角 y 座標  int GetWidth(); //  int GetHeight();  void LoadBitmap(); // 載入圖形  void OnMove(); // 移動敵人  void OnShow(); // 將敵人圖形貼到畫面  void GetAttack(const int damage); // 被攻擊  void AttackByEnemy(int\* heroHP, bool \*Poison);  bool isDead();  string GetEnemyType();  private:  CMovingBitmap Statue;  CMovingBitmap Statue\_Broken;  CAnimation HitAnimation; // 打擊效果  string EnemyType = "Statue";  int floor; // 地板的Y座標  bool rising; // true表上升、false表下降  int initial\_velocity; // 初始速度  int velocity; // 目前的速度(點/次)  int enemyAttackDamage = 0; //敵人攻擊力  int GetHitDelayCount = 0;  bool GetHit = false;  void changeLifeBarLength(); // 改變生命條的長度  };  } |
| CEnemy.cpp |
| #include "stdafx.h"  #include "Resource.h"  #include <mmsystem.h>  #include <ddraw.h>  #include "audio.h"  #include "gamelib.h"  #include "CEnemy.h"  #include "gameMap.h"  namespace game\_framework {  /////////////////////////////////////////////////////////////////////////////  // CEnemy: Enemy base class  /////////////////////////////////////////////////////////////////////////////  CEnemy::CEnemy(gameMap\* pointer, int x, int y) : currentMap(pointer), x(x), y(y) {};  CEnemy::~CEnemy()  {  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) delete (\*i);  };  void CEnemy::SetHeroXY(int x1, int x2, int y1, int y2)  {  hero["x1"] = x1;  hero["x2"] = x2;  hero["y1"] = y1;  hero["y2"] = y2;  }  void CEnemy::SetEnemyXY(int SetX, int SetY)  {  x = SetX;  y = SetY;  }  void CEnemy::SetHeroAttackRange(int x1, int x2, int y1, int y2)  {  heroAttackRange["x1"] = x1;  heroAttackRange["x2"] = x2;  heroAttackRange["y1"] = y1;  heroAttackRange["y2"] = y2;  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_sunFlower: Enemy sunFlower class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_sunFlower::CEnemy\_sunFlower(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  isMovingRight = true;  rising = false;  animation.SetDelayCount(5);  animationLeft.SetDelayCount(5);  moveRightAnimation.SetDelayCount(3);  moveLeftAnimation.SetDelayCount(3);  DeadAnimation.SetDelayCount(3);  AttackLeftAnimation.SetDelayCount(2);  AttackRightAnimation.SetDelayCount(2);  HitAnimation.SetDelayCount(2);  GetHitAnimation.SetDelayCount(3);  enemyHP = 350; //敵人預設生命值  FullHP = enemyHP;  enemyAttackDamage = 30; //敵人預設攻擊力  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  attackDelayCount = attackDelay = 70;  velocity = initial\_velocity;  state = STAND\_LEFT;  ShowLifeBarDelayCount = 0;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_sunFlower::~CEnemy\_sunFlower()  {  for (vector<bullet\_sunFlower\*>::iterator i = allBullet.begin(); i != allBullet.end(); i++) delete \*i;  }  int CEnemy\_sunFlower::GetX1()  {  return x;  }  int CEnemy\_sunFlower::GetY1()  {  return y;  }  int CEnemy\_sunFlower::GetX2()  {  return x + animation.Width();  }  int CEnemy\_sunFlower::GetY2()  {  return y + animation.Height();  }  int CEnemy\_sunFlower::GetWidth()  {  return animation.Width();  }  int CEnemy\_sunFlower::GetHeight()  {  return animation.Height();  }  void CEnemy\_sunFlower::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount==0)  {  GetHitAnimation.Reset();  CAudio::Instance()->Play(11, false);  GetHitDelayCount = 15;  enemyHP -= damage;  state = GET\_HIT;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_sunFlower::AttackByEnemy(int \*heroHP, bool \*Poison)  {  vector<bullet\_sunFlower\*>::iterator iter = allBullet.begin();  while (iter != allBullet.end())  {  if (((\*iter)->GetX2() >= hero["x1"]) && (hero["x2"] >= (\*iter)->GetX1()) && ((\*iter)->GetY2() >= hero["y1"]) && (hero["y2"] >= (\*iter)->GetY1()))  {  \*heroHP -= enemyAttackDamage;  delete \*iter;  iter = allBullet.erase(iter);  }  else  iter++;  }  }  CEnemy\_Action CEnemy\_sunFlower::DetectHero(CEnemy\_Action state)  {  if(attackDelayCount <= 0)  {  if ((GetX2() - GetWidth()/2 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_LEFT)  AttackLeftAnimation.Reset();  return ATTACK\_LEFT;  }  if ((GetX2() + 200 >= hero["x1"]) && (hero["x2"] >= GetX1() + GetWidth() / 2) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_RIGHT)  AttackRightAnimation.Reset();  return ATTACK\_RIGHT;  }  }  if ((GetX1() - 190 >= hero["x1"]) && (hero["x2"] >= GetX1() - 270) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_LEFT;  }  if ((GetX2() + 270 >= hero["x1"]) && (hero["x2"] >= GetX2() + 190) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_RIGHT;  }  if (state == MOVE\_LEFT || state == ATTACK\_LEFT || state == STAND\_LEFT)  {  return STAND\_LEFT;  }  if (state == MOVE\_RIGHT || state == ATTACK\_RIGHT || state == STAND\_RIGHT)  {  return STAND\_RIGHT;  }  if (state == GET\_HIT)  {  return GET\_HIT;  }  return STAND\_LEFT;  }  string CEnemy\_sunFlower::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_sunFlower::LoadBitmap()  {  animation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_2, RGB(255, 255, 255));  animation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_7, RGB(255, 255, 255));  animationLeft.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_2, RGB(255, 255, 255));  animationLeft.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_7, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_1, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_2, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_3, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_4, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_5, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_6, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_7, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_8, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SUNFLOWERRIGHTWALK\_9, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_0, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_1, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_2, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_3, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_4, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_5, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_6, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_7, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SUNFLOWERLEFTWALK\_8, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_0, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_1, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_2, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_3, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_4, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_5, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SUNFLOWERATTACKRIGHT\_6, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_0, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_1, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_2, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_3, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_4, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_5, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SUNFLOWERATTACKLEFT\_6, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SUNFLOWER\_DEAD\_0, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SUNFLOWER\_DEAD\_1, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SUNFLOWER\_DEAD\_2, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SUNFLOWER\_DEAD\_3, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SUNFLOWER\_DEAD\_4, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SUNFLOWER\_DEAD\_4, RGB(255, 255, 255));  GetHitAnimation.AddBitmap(IDB\_SUNFLOWERGETHITRIGHT\_0, RGB(255, 255, 255));  GetHitAnimation.AddBitmap(IDB\_SUNFLOWERGETHITRIGHT\_1, RGB(255, 255, 255));  GetHitAnimation.AddBitmap(IDB\_SUNFLOWERGETHITRIGHT\_2, RGB(255, 255, 255));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_sunFlower::OnMove()  {  const int STEP\_SIZE = 2;  animation.OnMove();  animationLeft.OnMove();  moveRightAnimation.OnMove();  moveLeftAnimation.OnMove();  AttackLeftAnimation.OnMove();  AttackRightAnimation.OnMove();  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  if (GetHitDelayCount > 0) GetHitDelayCount--;  else if (GetHitDelayCount == 0) HitAnimation.Reset();  vector<bullet\_sunFlower\*>::iterator iter = allBullet.begin();  while (iter != allBullet.end())  {  if ((\*iter)->isDelet())  {  delete \*iter;  iter = allBullet.erase(iter);  }  else  iter++;  }  for (vector<bullet\_sunFlower\*>::iterator i = allBullet.begin(); i != allBullet.end(); i++)  (\*i)->OnMove();  if (attackDelayCount > 0) attackDelayCount--;  state = DetectHero(state);  if (state == MOVE\_LEFT)  {  if (currentMap->isSpace(GetX1(), GetY1()) && currentMap->isSpace(GetX1(), GetY2() - 10) && !currentMap->isDoor(GetX1(),GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x -= STEP\_SIZE;  }  if (state == MOVE\_RIGHT)  {  if (currentMap->isSpace(GetX2(), GetY1()) && currentMap->isSpace(GetX2(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x += STEP\_SIZE;  }  if (state == ATTACK\_LEFT && AttackLeftAnimation.IsFinalBitmap())  {  allBullet.push\_back(new bullet\_sunFlower(currentMap, GetX1(), GetY1() + 40, -5));  allBullet.back()->LoadBitmap();  }  if (state == ATTACK\_RIGHT && AttackRightAnimation.IsFinalBitmap())  {  allBullet.push\_back(new bullet\_sunFlower(currentMap, GetX2(), GetY1() + 40, 5));  allBullet.back()->LoadBitmap();  }  if (state == GET\_HIT && !GetHitAnimation.IsFinalBitmap() && !HitAnimation.IsFinalBitmap())  {  state = GET\_HIT;  GetHitAnimation.OnMove();  HitAnimation.OnMove();  }  if (enemyHP <= 0 && !DeadAnimation.IsFinalBitmap())  {  state = DEAD;  DeadAnimation.OnMove();  }  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2()) && currentMap->isSpace(GetX2(), GetY2())) { // 當y座標還沒碰到地板  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  }  else {  floor = currentMap->GetBlockY(GetY2()) - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  }  void CEnemy\_sunFlower::OnShow()  {  for (vector<bullet\_sunFlower\*>::iterator i = allBullet.begin(); i != allBullet.end(); i++)  (\*i)->OnShow();  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  switch (state)  {  case STAND\_LEFT:  animationLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animationLeft.OnShow();  break;  case STAND\_RIGHT:  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  break;  case MOVE\_LEFT:  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  break;  case ATTACK\_LEFT:  if(AttackLeftAnimation.IsFinalBitmap()) attackDelayCount = attackDelay;  AttackLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackLeftAnimation.OnShow();  break;  case MOVE\_RIGHT:  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  break;  case ATTACK\_RIGHT:  if (AttackRightAnimation.IsFinalBitmap())attackDelayCount = attackDelay;  AttackRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackRightAnimation.OnShow();  break;  case GET\_HIT:  GetHitAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  GetHitAnimation.OnShow();  if (!HitAnimation.IsFinalBitmap())  {  HitAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HitAnimation.OnShow();  }  //if (GetHitAnimation.IsFinalBitmap()) GetHit = false;  break;  case DEAD:  if (enemyHP <= 0)  {  if (!DeadAnimation.IsFinalBitmap())  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  }  }  break;  }  }  bool CEnemy\_sunFlower::isDead()  {  if (enemyHP <= 0 && DeadAnimation.IsFinalBitmap()) return true;  else return false;  }  void CEnemy\_sunFlower::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {    (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_Cloud: Enemy Cloud class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_Cloud::CEnemy\_Cloud(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  isMovingRight = true;  rising = false;  animation.SetDelayCount(5);  animationLeft.SetDelayCount(5);  moveRightAnimation.SetDelayCount(3);  moveLeftAnimation.SetDelayCount(3);  DeadAnimation.SetDelayCount(3);  AttackLeftAnimation.SetDelayCount(4);  AttackRightAnimation.SetDelayCount(4);  LightningCloud.SetDelayCount(3);  TrackLightningCloud1.SetDelayCount(3);  HitAnimation.SetDelayCount(2);  enemyHP = 15000; //敵人預設生命值  FullHP = enemyHP;  enemyAttackDamage = 300; //敵人預設攻擊力  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  attackDelayCount = attackDelay = 150;  HeroXTrackCounter = SetHeroXTrackCounter;  TrackLightningDelayCount = SetTrackLightningDelayCount;  velocity = initial\_velocity;  state = STAND\_LEFT;  ShowLifeBarDelayCount = 0;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  for (int i = 0; i < 4; i++) HeroXArray[i] = 0; //初始化array  }  CEnemy\_Cloud::~CEnemy\_Cloud()  {  }  int CEnemy\_Cloud::GetX1()  {  return x;  }  int CEnemy\_Cloud::GetY1()  {  return y;  }  int CEnemy\_Cloud::GetX2()  {  return x + animation.Width();  }  int CEnemy\_Cloud::GetY2()  {  return y + animation.Height();  }  int CEnemy\_Cloud::GetWidth()  {  return animation.Width();  }  int CEnemy\_Cloud::GetHeight()  {  return animation.Height();  }  void CEnemy\_Cloud::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount == 0)  {  CAudio::Instance()->Play(11, false);  GetHitDelayCount = 15;  enemyHP -= damage;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_Cloud::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if ((GetX2() + 150 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1() - 100) && AttackFlag) //主動攻擊  {  \*heroHP -= enemyAttackDamage;  }  if ((HeroXArray[ShowLightningCloudNumber] + 100 >= hero["x1"]) && (hero["x2"] >= HeroXArray[ShowLightningCloudNumber]) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1() - 100) && LightningStrike) //被動雷雲  {  \*heroHP -= 100;  }  }  CEnemy\_Action CEnemy\_Cloud::DetectHero(CEnemy\_Action state)  {  if (attackDelayCount <= 0)  {  if ((GetX2() - GetWidth() / 2 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_LEFT)  {  AttackLeftAnimation.Reset();  LightningCloud.Reset();  }  return ATTACK\_LEFT;  }  if ((GetX2() + 200 >= hero["x1"]) && (hero["x2"] >= GetX1() + GetWidth() / 2) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_RIGHT)  {  AttackRightAnimation.Reset();  LightningCloud.Reset();  }    return ATTACK\_RIGHT;  }  }  if ((GetX1() - 190 >= hero["x1"]) && (hero["x2"] >= GetX1() - 270) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_LEFT;  }  if ((GetX2() + 270 >= hero["x1"]) && (hero["x2"] >= GetX2() + 190) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_RIGHT;  }  if (state == MOVE\_LEFT || state == ATTACK\_LEFT || state == STAND\_LEFT)  {  return STAND\_LEFT;  }  if (state == MOVE\_RIGHT || state == ATTACK\_RIGHT || state == STAND\_RIGHT)  {  return STAND\_RIGHT;  }  if (state == GET\_HIT)  {  return GET\_HIT;  }  return STAND\_LEFT;  }  string CEnemy\_Cloud::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_Cloud::LoadBitmap()  {  animation.AddBitmap(IDB\_CLOUDBOSS\_0, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_1, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_2, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_3, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_4, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_5, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_6, RGB(63, 72, 204));  animation.AddBitmap(IDB\_CLOUDBOSS\_7, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_0, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_1, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_2, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_3, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_4, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_5, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_6, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_CLOUDBOSS\_LEFT\_7, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_0, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_1, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_2, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_3, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_4, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_5, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_CLOUDWALKRIGHT\_6, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_0, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_1, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_2, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_3, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_4, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_5, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_CLOUDWALKLEFT\_6, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_1, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_2, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_3, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_4, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_5, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_6, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_7, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_8, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_CLOUDYATTACKRIGHT\_9, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_1, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_2, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_3, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_4, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_5, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_6, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_7, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_8, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_CLOUDYATTACKLEFT\_9, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_CLOUDYDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_CLOUDYDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_CLOUDYDEAD\_2, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_CLOUDYDEAD\_3, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_CLOUDYDEAD\_4, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_CLOUDYDEAD\_5, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_0, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_1, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_2, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_3, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_4, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_5, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_6, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_8, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_9, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_10, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_11, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_12, RGB(63, 72, 204));  LightningCloud.AddBitmap(IDB\_LIGHTNING\_13, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_0, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_1, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_2, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_3, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_4, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_5, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_6, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_8, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_9, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_10, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_11, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_12, RGB(63, 72, 204));  TrackLightningCloud1.AddBitmap(IDB\_LIGHTNING\_13, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_Cloud::OnMove()  {  const int STEP\_SIZE = 2;  animation.OnMove();  animationLeft.OnMove();  moveRightAnimation.OnMove();  moveLeftAnimation.OnMove();  AttackLeftAnimation.OnMove();  AttackRightAnimation.OnMove();  TrackLightningCloud1.OnMove();  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  if (GetHitDelayCount > 0) GetHitDelayCount--;  else if (GetHitDelayCount == 0) HitAnimation.Reset();  if (attackDelayCount > 0) attackDelayCount--;  if (TrackLightningDelayCount > 0 && !LightningActivated) //追蹤閃電計數部分  {  TrackLightningDelayCount--;    if (TrackLightningDelayCount % 20 == 0) //每40個畫面寫入陣列  {  SetHeroXArray(HeroXCounter, hero["x1"]); //Array[HeroCounter] = hero[x1] //紀錄英雄位置  HeroXCounter++; //changeArrayNumber  }  if (TrackLightningDelayCount == 0)  {  LightningActivated = true;  TrackLightningDelayCount = SetTrackLightningDelayCount;  HeroXCounter = 0;  }  }  else if (LightningActivated) //被動閃電DelayCount  {  if (LightningStrikeDelayCount > 0) LightningStrikeDelayCount--;  }  if (LightningCloud.IsFinalBitmap() || LightningCloud.GetCurrentBitmapNumber()==0) //無法取消攻擊動作  {  AttackAudio\_1 = false;  state = DetectHero(state);  }  if (state == MOVE\_LEFT)  {  if (currentMap->isSpace(GetX1(), GetY1()) && currentMap->isSpace(GetX1(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x -= STEP\_SIZE;  }  if (state == MOVE\_RIGHT)  {  if (currentMap->isSpace(GetX2(), GetY1()) && currentMap->isSpace(GetX2(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x += STEP\_SIZE;  }  if (state == ATTACK\_LEFT || state == ATTACK\_RIGHT)  {  LightningCloud.OnMove();  if (LightningCloud.GetCurrentBitmapNumber() >= 7 && !AttackAudio\_1)  {  CAudio::Instance()->Play(16, false);  AttackAudio\_1 = true;  }  if (LightningCloud.GetCurrentBitmapNumber() == 7)  {  AttackFlag = true;  }  else  {  AttackFlag = false;  }  }  if (state == GET\_HIT && !HitAnimation.IsFinalBitmap())  {  state = GET\_HIT;  HitAnimation.OnMove();  }  if (enemyHP <= 0 && !DeadAnimation.IsFinalBitmap())  {  state = DEAD;  DeadAnimation.OnMove();  }  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2()) && currentMap->isSpace(GetX2(), GetY2())) { // 當y座標還沒碰到地板  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  }  else {  floor = currentMap->GetBlockY(GetY2()) - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  }  void CEnemy\_Cloud::OnShow()  {  if (state != DEAD && LightningActivated && LightningStrikeDelayCount == 0)  {  ShowTrackLightningCloud(ShowLightningCloudNumber);  }  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  switch (state)  {  case STAND\_LEFT:  animationLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animationLeft.OnShow();  break;  case STAND\_RIGHT:  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  break;  case MOVE\_LEFT:  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  break;  case ATTACK\_LEFT:  if (LightningCloud.GetCurrentBitmapNumber() == 7) attackDelayCount = attackDelay;  LightningCloud.SetTopLeft(currentMap->ScreenX(x - 200), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x - 100), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x + 100), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x + 200), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  AttackLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackLeftAnimation.OnShow();  break;  case MOVE\_RIGHT:  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  break;  case ATTACK\_RIGHT:  if (LightningCloud.GetCurrentBitmapNumber() == 7)attackDelayCount = attackDelay;  LightningCloud.SetTopLeft(currentMap->ScreenX(x - 200), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x - 100), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x + 100), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  LightningCloud.SetTopLeft(currentMap->ScreenX(x + 200), currentMap->ScreenY(y - 72));  LightningCloud.OnShow();  AttackRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackRightAnimation.OnShow();  break;  case DEAD:  if (enemyHP <= 0)  {  if (!DeadAnimation.IsFinalBitmap())  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  }  }  break;  }  }  bool CEnemy\_Cloud::isDead()  {  if (enemyHP <= 0 && DeadAnimation.IsFinalBitmap()) return true;  else return false;  }  void CEnemy\_Cloud::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  void CEnemy\_Cloud::SetHeroXArray(int ArrayNumber, int ArrayValue)  {  HeroXArray[ArrayNumber] = ArrayValue;  }  void CEnemy\_Cloud::ResetTrackLightningCloudAnimation(int CloudNumber)  {  switch (CloudNumber)  {  case 0:  TrackLightningCloud1.Reset();  break;  default:  break;  }  }  void CEnemy\_Cloud::ShowTrackLightningCloud(int CloudNumber)  {  if (!TrackLightningCloud1.IsFinalBitmap())  {  if (TrackLightningCloud1.GetCurrentBitmapNumber() >= 7 && !AttackAudio\_2) //避免音效重複撥放  {  CAudio::Instance()->Play(16, false);  AttackAudio\_2 = true;  }  if (TrackLightningCloud1.GetCurrentBitmapNumber()==7) //被動閃電造成攻擊  {  LightningStrike = true;  }  else  {  LightningStrike = false;  }  TrackLightningCloud1.SetTopLeft(currentMap->ScreenX(HeroXArray[CloudNumber] - 50), currentMap->ScreenY(y - 72));  TrackLightningCloud1.OnShow();  }  else  {  AttackAudio\_2 = false;  if (ShowLightningCloudNumber <= 2)  {  ShowLightningCloudNumber += 1;  LightningStrikeDelayCount = 10;  }  else  {  ShowLightningCloudNumber = 0;  LightningActivated = false;  }    }  }  void CEnemy\_Cloud::OnMoveTrackLightningCloud(int CloudNumber)  {  if (CloudNumber > 0)  {  TrackLightningCloud1.OnMove();  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_GasRobot: Enemy class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_GasRobot::CEnemy\_GasRobot(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  isMovingRight = true;  rising = false;  animation.SetDelayCount(5);  animationLeft.SetDelayCount(5);  GasRobotFireRightAnimation.SetDelayCount(1);  GasRobotFireLeftAnimation.SetDelayCount(1);  GasAnimation.SetDelayCount(2);  moveRightAnimation.SetDelayCount(3);  moveLeftAnimation.SetDelayCount(3);  DeadAnimation.SetDelayCount(3);  AttackLeftAnimation.SetDelayCount(2);  AttackRightAnimation.SetDelayCount(2);  CountDownNumber.SetDelayCount(10);  ExplosionAnimation.SetDelayCount(3);  HitAnimation.SetDelayCount(2);  enemyHP = 450; //敵人預設生命值  FullHP = enemyHP;  enemyAttackDamage = 10; //敵人預設攻擊力  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  attackDelayCount = attackDelay = 150;  velocity = initial\_velocity;  state = STAND\_LEFT;  ShowLifeBarDelayCount = 0;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_GasRobot::~CEnemy\_GasRobot()  {  }  int CEnemy\_GasRobot::GetX1()  {  return x;  }  int CEnemy\_GasRobot::GetY1()  {  return y;  }  int CEnemy\_GasRobot::GetX2()  {  return x + animation.Width();  }  int CEnemy\_GasRobot::GetY2()  {  return y + animation.Height();  }  int CEnemy\_GasRobot::GetWidth()  {  return animation.Width();  }  int CEnemy\_GasRobot::GetHeight()  {  return animation.Height();  }  void CEnemy\_GasRobot::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount == 0)  {  CAudio::Instance()->Play(13, false);  GetHitDelayCount = 15;  enemyHP -= damage;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_GasRobot::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if ((GetX2() + 50 >= hero["x1"]) && (hero["x2"] >= GetX1() - 50) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag) //施放毒氣  {  \*Poison = true;  }  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && ExplosionAnimation.GetCurrentBitmapNumber() == 1) //死後爆炸造成傷害  {  \*heroHP -= 100;  }  }  CEnemy\_Action CEnemy\_GasRobot::DetectHero(CEnemy\_Action state)  {  if (attackDelayCount <= 0)  {  if ((GetX2() - GetWidth() / 2 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_LEFT)  AttackLeftAnimation.Reset();  return ATTACK\_LEFT;  }  if ((GetX2() + 200 >= hero["x1"]) && (hero["x2"] >= GetX1() + GetWidth() / 2) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_RIGHT)  AttackRightAnimation.Reset();  return ATTACK\_RIGHT;  }  }  if ((GetX1() - 190 >= hero["x1"]) && (hero["x2"] >= GetX1() - 270) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_LEFT;  }  if ((GetX2() + 270 >= hero["x1"]) && (hero["x2"] >= GetX2() + 190) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_RIGHT;  }  if (state == MOVE\_LEFT || state == ATTACK\_LEFT || state == STAND\_LEFT)  {  return STAND\_LEFT;  }  if (state == MOVE\_RIGHT || state == ATTACK\_RIGHT || state == STAND\_RIGHT)  {  return STAND\_RIGHT;  }  if (state == GET\_HIT)  {  return GET\_HIT;  }  return STAND\_LEFT;  }  string CEnemy\_GasRobot::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_GasRobot::LoadBitmap()  {  animation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_0, RGB(63, 72, 204));  animation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_1, RGB(63, 72, 204));  animation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_2, RGB(63, 72, 204));  animation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_3, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_GASROBOTSTANDLEFT\_0, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_GASROBOTSTANDLEFT\_1, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_GASROBOTSTANDLEFT\_2, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_GASROBOTSTANDLEFT\_3, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_0, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_1, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_2, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_GASROBOTSTANDRIGHT\_3, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_GASROBOTSTANDLEFT\_0, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_GASROBOTSTANDLEFT\_1, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_GASROBOTSTANDLEFT\_2, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_GASROBOTSTANDLEFT\_3, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_1, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_2, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_3, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_4, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_5, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_6, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_GASROBOTATTACKRIGHT\_6, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_1, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_2, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_3, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_4, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_5, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_6, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_GASROBOTATTACKLEFT\_6, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_GASROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_GASROBOTDEAD\_1, RGB(63, 72, 204));  GasRobotFireRightAnimation.AddBitmap(IDB\_GASROBOTFIRERIGHT\_0, RGB(63, 72, 204));  GasRobotFireRightAnimation.AddBitmap(IDB\_GASROBOTFIRERIGHT\_1, RGB(63, 72, 204));  GasRobotFireLeftAnimation.AddBitmap(IDB\_GASROBOTFIRELEFT\_0, RGB(63, 72, 204));  GasRobotFireLeftAnimation.AddBitmap(IDB\_GASROBOTFIRELEFT\_1, RGB(63, 72, 204));  GasAnimation.AddBitmap(IDB\_GAS\_0, RGB(63, 72, 204));  GasAnimation.AddBitmap(IDB\_GAS\_1, RGB(63, 72, 204));  GasAnimation.AddBitmap(IDB\_GAS\_2, RGB(63, 72, 204));  GasAnimation.AddBitmap(IDB\_GAS\_3, RGB(63, 72, 204));  GasAnimation.AddBitmap(IDB\_GAS\_4, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_0, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_1, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_2, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_3, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_4, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_5, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_6, RGB(63, 72, 204));  ExplosionAnimation.AddBitmap(IDB\_EXPLOSION\_7, RGB(63, 72, 204));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_5, RGB(255, 255, 255));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_4, RGB(255, 255, 255));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_3, RGB(255, 255, 255));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_2, RGB(255, 255, 255));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_1, RGB(255, 255, 255));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_0, RGB(255, 255, 255));  CountDownNumber.AddBitmap(IDB\_COUNTDOWN\_0, RGB(255, 255, 255));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_GasRobot::OnMove()  {  const int STEP\_SIZE = 2;  animation.OnMove();  animationLeft.OnMove();  GasRobotFireRightAnimation.OnMove();  GasRobotFireLeftAnimation.OnMove();  moveRightAnimation.OnMove();  moveLeftAnimation.OnMove();  GasAnimation.OnMove();  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  if (GetHitDelayCount > 0) GetHitDelayCount--;  else if (GetHitDelayCount == 0) HitAnimation.Reset();  if (attackDelayCount > 0) attackDelayCount--;  if (AttackRightAnimation.GetCurrentBitmapNumber() == 0 && AttackLeftAnimation.GetCurrentBitmapNumber() == 0)  {  state = DetectHero(state);  }  if (state == MOVE\_LEFT)  {  if (currentMap->isSpace(GetX1(), GetY1()) && currentMap->isSpace(GetX1(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x -= STEP\_SIZE;  }  if (state == MOVE\_RIGHT)  {  if (currentMap->isSpace(GetX2(), GetY1()) && currentMap->isSpace(GetX2(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x += STEP\_SIZE;  }  if (state == GET\_HIT && !HitAnimation.IsFinalBitmap())  {  state = GET\_HIT;  HitAnimation.OnMove();  }  if (state == ATTACK\_LEFT)  {  AttackLeftAnimation.OnMove();  if (AttackLeftAnimation.IsFinalBitmap() && !ShowGas)  {  ShowGas = true;  GasAnimation.Reset();  AttackFlag = true;  }  if (AttackLeftAnimation.IsFinalBitmap())  {  attackDelayCount = attackDelay;  AttackLeftAnimation.Reset();  }  }  if (state == ATTACK\_RIGHT)  {  AttackRightAnimation.OnMove();  if (AttackRightAnimation.IsFinalBitmap() && !ShowGas)  {  ShowGas = true;  GasAnimation.Reset();  AttackFlag = true;  }  if (AttackRightAnimation.IsFinalBitmap())  {  attackDelayCount = attackDelay;  AttackRightAnimation.Reset();  }  }  if (enemyHP <= 0)  {  state = DEAD;  if (!CountDownNumber.IsFinalBitmap())  {  DeadAnimation.OnMove();  CountDownNumber.OnMove();  if (!DeadAudio)  {  CAudio::Instance()->Play(15, false);  DeadAudio = true;  }  }  else  {  ExplosionAnimation.OnMove();  if (!ExplosionAudio)  {  CAudio::Instance()->Play(19, false);  ExplosionAudio = true;  }  }  }  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2()) && currentMap->isSpace(GetX2(), GetY2())) { // 當y座標還沒碰到地板  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  }  else {  floor = currentMap->GetBlockY(GetY2()) - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  }  void CEnemy\_GasRobot::OnShow()  {  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  switch (state)  {  case STAND\_LEFT:  animationLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animationLeft.OnShow();  GasRobotFireLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 80));  GasRobotFireLeftAnimation.OnShow();  break;  case STAND\_RIGHT:  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  GasRobotFireRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 80));  GasRobotFireRightAnimation.OnShow();  break;  case MOVE\_LEFT:  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  GasRobotFireLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 80));  GasRobotFireLeftAnimation.OnShow();  break;  case ATTACK\_LEFT:  AttackLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackLeftAnimation.OnShow();  GasRobotFireLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 80));  GasRobotFireLeftAnimation.OnShow();  break;  case MOVE\_RIGHT:  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  GasRobotFireRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 80));  GasRobotFireRightAnimation.OnShow();  break;  case ATTACK\_RIGHT:    AttackRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackRightAnimation.OnShow();  GasRobotFireRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 80));  GasRobotFireRightAnimation.OnShow();  break;  case DEAD:  if (enemyHP <= 0)  {  if (!CountDownNumber.IsFinalBitmap())  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  CountDownNumber.SetTopLeft(currentMap->ScreenX(x + 70), currentMap->ScreenY(y + 65));  CountDownNumber.OnShow();  }  else  {  if (!ExplosionAnimation.IsFinalBitmap())  {  ExplosionAnimation.SetTopLeft(currentMap->ScreenX(x - 50), currentMap->ScreenY(y - 80));  ExplosionAnimation.OnShow();  }  }  }  break;  }  if (ShowGas)  {  GasAnimation.SetTopLeft(currentMap->ScreenX(x - 80), currentMap->ScreenY(y - 50));  GasAnimation.OnShow();  if (GasAnimation.IsFinalBitmap())  {  ShowGas = false;  AttackFlag = false;  }  }  }  bool CEnemy\_GasRobot::isDead()  {  if (enemyHP <= 0 && ExplosionAnimation.IsFinalBitmap()) return true;  else return false;  }  void CEnemy\_GasRobot::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_RobotA: Enemy class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_RobotA::CEnemy\_RobotA(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  isMovingRight = true;  rising = false;  animation.SetDelayCount(5);  animationLeft.SetDelayCount(5);  moveRightAnimation.SetDelayCount(3);  moveLeftAnimation.SetDelayCount(3);  DeadAnimation.SetDelayCount(3);  AttackLeftAnimation.SetDelayCount(3);  AttackRightAnimation.SetDelayCount(3);  HitAnimation.SetDelayCount(2);  GetHitAnimation.SetDelayCount(3);  AttackVrfxLeft.SetDelayCount(3);  AttackVrfxRight.SetDelayCount(3);  enemyHP = 550; //敵人預設生命值  FullHP = enemyHP;  enemyAttackDamage = 60; //敵人預設攻擊力  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  attackDelayCount = attackDelay = 70;  velocity = initial\_velocity;  state = STAND\_LEFT;  ShowLifeBarDelayCount = 0;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_RobotA::~CEnemy\_RobotA()  {  //for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) delete \*i;  }  int CEnemy\_RobotA::GetX1()  {  return x;  }  int CEnemy\_RobotA::GetY1()  {  return y;  }  int CEnemy\_RobotA::GetX2()  {  return x + animation.Width();  }  int CEnemy\_RobotA::GetY2()  {  return y + animation.Height();  }  int CEnemy\_RobotA::GetWidth()  {  return animation.Width();  }  int CEnemy\_RobotA::GetHeight()  {  return animation.Height();  }  void CEnemy\_RobotA::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount == 0)  {  GetHitAnimation.Reset();  CAudio::Instance()->Play(13, false);  GetHitDelayCount = 15;  enemyHP -= damage;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_RobotA::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag) //攻擊  {  \*heroHP -= enemyAttackDamage;  }  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && DeadAnimation.GetCurrentBitmapNumber() == 16) //死亡後爆炸造成傷害  {  \*heroHP -= 100;  }  }  CEnemy\_Action CEnemy\_RobotA::DetectHero(CEnemy\_Action state)  {  if (attackDelayCount <= 0)  {  if ((GetX2() - GetWidth() / 2 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_LEFT)  AttackLeftAnimation.Reset();  return ATTACK\_LEFT;  }  if ((GetX2() + 200 >= hero["x1"]) && (hero["x2"] >= GetX1() + GetWidth() / 2) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_RIGHT)  AttackRightAnimation.Reset();  return ATTACK\_RIGHT;  }  }  if ((GetX1() - 190 >= hero["x1"]) && (hero["x2"] >= GetX1() - 270) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_LEFT;  }  if ((GetX2() + 270 >= hero["x1"]) && (hero["x2"] >= GetX2() + 190) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_RIGHT;  }  if (state == MOVE\_LEFT || state == ATTACK\_LEFT || state == STAND\_LEFT)  {  return STAND\_LEFT;  }  if (state == MOVE\_RIGHT || state == ATTACK\_RIGHT || state == STAND\_RIGHT)  {  return STAND\_RIGHT;  }  return STAND\_LEFT;  }  string CEnemy\_RobotA::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_RobotA::LoadBitmap()  {  animation.AddBitmap(IDB\_ROBOTSTANDRIGHT\_0, RGB(63, 72, 204));  animation.AddBitmap(IDB\_ROBOTSTANDRIGHT\_1, RGB(63, 72, 204));  animation.AddBitmap(IDB\_ROBOTSTANDRIGHT\_2, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_ROBOTSTANDLEFT\_0, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_ROBOTSTANDLEFT\_1, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_ROBOTSTANDLEFT\_2, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_ROBOTMOVERIGHT\_0, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_ROBOTMOVERIGHT\_1, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_ROBOTMOVERIGHT\_2, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_ROBOTMOVERIGHT\_3, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_ROBOTMOVELEFT\_0, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_ROBOTMOVELEFT\_1, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_ROBOTMOVELEFT\_2, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_ROBOTMOVELEFT\_3, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_0, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_1, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_2, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_3, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_4, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_5, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_6, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_7, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_8, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_9, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_10, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_11, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_12, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_13, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_ROBOTATTACKRIGHT\_13, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_0, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_1, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_2, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_3, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_4, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_5, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_6, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_7, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_8, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_9, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_10, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_11, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_12, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_13, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_ROBOTATTACKLEFT\_13, RGB(63, 72, 204));  AttackVrfxLeft.AddBitmap(IDB\_ROBOTATTACKVRFXLEFT\_0, RGB(63, 72, 204));  AttackVrfxLeft.AddBitmap(IDB\_ROBOTATTACKVRFXLEFT\_1, RGB(63, 72, 204));  AttackVrfxLeft.AddBitmap(IDB\_ROBOTATTACKVRFXLEFT\_2, RGB(63, 72, 204));  AttackVrfxLeft.AddBitmap(IDB\_ROBOTATTACKVRFXLEFT\_3, RGB(63, 72, 204));  AttackVrfxLeft.AddBitmap(IDB\_ROBOTATTACKVRFXLEFT\_4, RGB(63, 72, 204));  AttackVrfxLeft.AddBitmap(IDB\_ROBOTATTACKVRFXLEFT\_5, RGB(63, 72, 204));  AttackVrfxRight.AddBitmap(IDB\_ROBOTATTACKVRFXRIGHT\_0, RGB(63, 72, 204));  AttackVrfxRight.AddBitmap(IDB\_ROBOTATTACKVRFXRIGHT\_1, RGB(63, 72, 204));  AttackVrfxRight.AddBitmap(IDB\_ROBOTATTACKVRFXRIGHT\_2, RGB(63, 72, 204));  AttackVrfxRight.AddBitmap(IDB\_ROBOTATTACKVRFXRIGHT\_3, RGB(63, 72, 204));  AttackVrfxRight.AddBitmap(IDB\_ROBOTATTACKVRFXRIGHT\_4, RGB(63, 72, 204));  AttackVrfxRight.AddBitmap(IDB\_ROBOTATTACKVRFXRIGHT\_5, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_ROBOTDEAD\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_1, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_2, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_3, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_4, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_5, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_6, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_EXPLOSION\_7, RGB(63, 72, 204));  GetHitAnimation.AddBitmap(IDB\_ROBOTSTANDLEFT\_0, RGB(63, 72, 204));  GetHitAnimation.AddBitmap(IDB\_ROBOTSTANDLEFT\_1, RGB(63, 72, 204));  GetHitAnimation.AddBitmap(IDB\_ROBOTSTANDLEFT\_2, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_RobotA::OnMove()  {  const int STEP\_SIZE = 2;  animation.OnMove();  animationLeft.OnMove();  moveRightAnimation.OnMove();  moveLeftAnimation.OnMove();  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  if (GetHitDelayCount > 0) GetHitDelayCount--;  else if (GetHitDelayCount == 0) HitAnimation.Reset();  if (attackDelayCount > 0) attackDelayCount--;  if (AttackLeftAnimation.GetCurrentBitmapNumber() == 0 && AttackRightAnimation.GetCurrentBitmapNumber() == 0) //無法取消攻擊動作  {  state = DetectHero(state);  }  if (state == MOVE\_LEFT)  {  if (currentMap->isSpace(GetX1(), GetY1()) && currentMap->isSpace(GetX1(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x -= STEP\_SIZE;  }  if (state == MOVE\_RIGHT)  {  if (currentMap->isSpace(GetX2(), GetY1()) && currentMap->isSpace(GetX2(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x += STEP\_SIZE;  }  if (state == ATTACK\_LEFT) //向左攻擊  {  AttackLeftAnimation.OnMove();  if (AttackLeftAnimation.GetCurrentBitmapNumber() >= 7)  {  if (!AttackAudio)  {  CAudio::Instance()->Play(14, false);  AttackAudio = true;  }  }  if (AttackLeftAnimation.GetCurrentBitmapNumber() >= 9)  {  AttackFlag = true;  if (AttackLeftAnimation.GetCurrentBitmapNumber() == 9) //開始衝刺  {  for (int i = 0; i < DASH\_SIZE; i++) {  if (currentMap->isSpace(GetX1() - 1, GetY1()) && currentMap->isSpace(GetX1() - 1, GetY2() - 10) && !currentMap->isDoor(GetX1() - 1, GetY1()) && !currentMap->isDoor(GetX1() - 1, GetY2())) // 當x座標還沒碰到牆  {  x -= 1; //正常走  }  }  }  AttackVrfxLeft.OnMove();  }  else  {  AttackFlag = false;  AttackVrfxLeft.Reset();  }  }  if (state == ATTACK\_RIGHT) //向右攻擊  {  AttackRightAnimation.OnMove();  if (AttackRightAnimation.GetCurrentBitmapNumber() >= 7)  {  if (!AttackAudio)  {  CAudio::Instance()->Play(14, false);  AttackAudio = true;  }  }  if (AttackRightAnimation.GetCurrentBitmapNumber() >= 9)  {  AttackFlag = true;  if (AttackRightAnimation.GetCurrentBitmapNumber() == 9) //開始衝刺  {  for (int i = 0; i < DASH\_SIZE; i++)  {  if (currentMap->isSpace(GetX2() + 1, GetY1()) && currentMap->isSpace(GetX2() + 1, GetY2() - 10) && !currentMap->isDoor(GetX2() + 1, GetY1()) && !currentMap->isDoor(GetX2() + 1, GetY2())) // 當x座標還沒碰到牆  {  x += 1; //正常走  }  }  }  AttackVrfxRight.OnMove();  }  else  {  AttackFlag = false;  AttackVrfxRight.Reset();  }  }  if (enemyHP <= 0 && !DeadAnimation.IsFinalBitmap())  {  state = DEAD;  DeadAnimation.OnMove();  if (!DeadAudio)  {  CAudio::Instance()->Play(15, false);  DeadAudio = true;  }  if (DeadAnimation.GetCurrentBitmapNumber() == 16 && !ExplosionAudio)  {  CAudio::Instance()->Play(19, false);  ExplosionAudio = true;  }  }  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2()) && currentMap->isSpace(GetX2(), GetY2())) { // 當y座標還沒碰到地板  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  }  else {  floor = currentMap->GetBlockY(GetY2()) - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  }  void CEnemy\_RobotA::OnShow()  {  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  switch (state)  {  case STAND\_LEFT:  animationLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animationLeft.OnShow();  break;  case STAND\_RIGHT:  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  break;  case MOVE\_LEFT:  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  break;  case ATTACK\_LEFT:  if (AttackLeftAnimation.IsFinalBitmap())  {  AttackLeftAnimation.Reset();  attackDelayCount = attackDelay;  AttackFlag = false;  AttackAudio = false;  }  AttackLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackLeftAnimation.OnShow();  if (AttackLeftAnimation.GetCurrentBitmapNumber() >= 9)  {  AttackVrfxLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 30));  AttackVrfxLeft.OnShow();  }  break;  case MOVE\_RIGHT:  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  break;  case ATTACK\_RIGHT:  if (AttackRightAnimation.IsFinalBitmap())  {  AttackRightAnimation.Reset();  attackDelayCount = attackDelay;  AttackFlag = false;  AttackAudio = false;  }  AttackRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackRightAnimation.OnShow();  if (AttackRightAnimation.GetCurrentBitmapNumber() >= 9)  {  AttackVrfxRight.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y + 30));  AttackVrfxRight.OnShow();  }  break;  case GET\_HIT:  GetHitAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  GetHitAnimation.OnShow();  if (!HitAnimation.IsFinalBitmap())  {  HitAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HitAnimation.OnShow();  }  break;  case DEAD:  if (enemyHP <= 0)  {  if (!DeadAnimation.IsFinalBitmap())  {  if (DeadAnimation.GetCurrentBitmapNumber() >= 16)  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x - 50), currentMap->ScreenY(y - 50));  DeadAnimation.OnShow();  }  else  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  }  }  }  break;  }  }  bool CEnemy\_RobotA::isDead()  {  if (enemyHP <= 0 && DeadAnimation.IsFinalBitmap()) return true;  else return false;  }  void CEnemy\_RobotA::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_Pigeon: Enemy class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_Pigeon::CEnemy\_Pigeon(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  isMovingRight = true;  rising = false;  animation.SetDelayCount(5);  animationLeft.SetDelayCount(5);  moveRightAnimation.SetDelayCount(3);  moveLeftAnimation.SetDelayCount(3);  DeadAnimation.SetDelayCount(3);  AttackLeftAnimation.SetDelayCount(3);  AttackRightAnimation.SetDelayCount(3);  HitAnimation.SetDelayCount(2);  AttackVrfx.SetDelayCount(3);  enemyHP = 250; //敵人預設生命值  FullHP = enemyHP;  enemyAttackDamage = 50; //敵人預設攻擊力  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  attackDelayCount = attackDelay = 70;  velocity = initial\_velocity;  state = STAND\_LEFT;  ShowLifeBarDelayCount = 0;  RecordedX1 = 0;  RecordedY1 = 0;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_Pigeon::~CEnemy\_Pigeon()  {  }  int CEnemy\_Pigeon::GetX1()  {  return x;  }  int CEnemy\_Pigeon::GetY1()  {  return y;  }  int CEnemy\_Pigeon::GetX2()  {  return x + animation.Width();  }  int CEnemy\_Pigeon::GetY2()  {  return y + animation.Height();  }  int CEnemy\_Pigeon::GetWidth()  {  return animation.Width();  }  int CEnemy\_Pigeon::GetHeight()  {  return animation.Height();  }  void CEnemy\_Pigeon::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount == 0)  {  CAudio::Instance()->Play(11, false);  GetHitDelayCount = 15;  enemyHP -= damage;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_Pigeon::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if ((RecordedX1 - 60 + AttackVrfx.Width() >= hero["x1"]) && (hero["x2"] >= RecordedX1 - 60) && (RecordedY1 - 50 + AttackVrfx.Height() >= hero["y1"]) && (hero["y2"] >= RecordedY1 - 50) && AttackFlag)  {  \*heroHP -= enemyAttackDamage;  }  }  CEnemy\_Action CEnemy\_Pigeon::DetectHero(CEnemy\_Action state)  {  if (attackDelayCount <= 0)  {  if ((GetX2() - GetWidth() / 2 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_LEFT)  AttackLeftAnimation.Reset();  return ATTACK\_LEFT;  }  if ((GetX2() + 200 >= hero["x1"]) && (hero["x2"] >= GetX1() + GetWidth() / 2) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_RIGHT)  AttackRightAnimation.Reset();  return ATTACK\_RIGHT;  }  }  if ((GetX1() - 190 >= hero["x1"]) && (hero["x2"] >= GetX1() - 270) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_LEFT;  }  if ((GetX2() + 270 >= hero["x1"]) && (hero["x2"] >= GetX2() + 190) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_RIGHT;  }  if (state == MOVE\_LEFT || state == ATTACK\_LEFT || state == STAND\_LEFT)  {  return STAND\_LEFT;  }  if (state == MOVE\_RIGHT || state == ATTACK\_RIGHT || state == STAND\_RIGHT)  {  return STAND\_RIGHT;  }  return STAND\_LEFT;  }  string CEnemy\_Pigeon::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_Pigeon::LoadBitmap()  {  animation.AddBitmap(IDB\_PIGEONSTANDRIGHT\_0, RGB(63, 72, 204));  animation.AddBitmap(IDB\_PIGEONSTANDRIGHT\_1, RGB(63, 72, 204));  animation.AddBitmap(IDB\_PIGEONSTANDRIGHT\_2, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_PIGEONSTANDLEFT\_0, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_PIGEONSTANDLEFT\_1, RGB(63, 72, 204));  animationLeft.AddBitmap(IDB\_PIGEONSTANDLEFT\_2, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_PIGEONMOVERIGHT\_0, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_PIGEONMOVERIGHT\_1, RGB(63, 72, 204));  moveRightAnimation.AddBitmap(IDB\_PIGEONMOVERIGHT\_2, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_PIGEONMOVELEFT\_0, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_PIGEONMOVELEFT\_1, RGB(63, 72, 204));  moveLeftAnimation.AddBitmap(IDB\_PIGEONMOVELEFT\_2, RGB(63, 72, 204));  AttackRightAnimation.AddBitmap(IDB\_PIGEONATTACKRIGHT, RGB(63, 72, 204));  AttackLeftAnimation.AddBitmap(IDB\_PIGEONATTACKLEFT, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_0, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_1, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_2, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_3, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_4, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_5, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_6, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_7, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_8, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_9, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_10, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_11, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_12, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_13, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_14, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_15, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_16, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_17, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_18, RGB(63, 72, 204));  AttackVrfx.AddBitmap(IDB\_PIGEONATTACKVFX\_19, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_PIGEONDEAD\_0, RGB(63, 72, 204));  DeadAnimation.AddBitmap(IDB\_PIGEONDEAD\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_Pigeon::OnMove()  {  const int STEP\_SIZE = 2;  animation.OnMove();  animationLeft.OnMove();  moveRightAnimation.OnMove();  moveLeftAnimation.OnMove();  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  if (GetHitDelayCount > 0) GetHitDelayCount--;  else if (GetHitDelayCount == 0) HitAnimation.Reset();  if (attackDelayCount > 0) attackDelayCount--;  else if (attackDelayCount == 0 && !recorded) //攻擊前且尚未紀錄主角位置  {  RecordedX1 = hero["x1"];  RecordedY1 = hero["y1"];  recorded = true;  }  if (AttackVrfx.GetCurrentBitmapNumber()==0)  {  state = DetectHero(state);  }  if (state == MOVE\_LEFT)  {  if (currentMap->isSpace(GetX1(), GetY1()) && currentMap->isSpace(GetX1(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x -= STEP\_SIZE;  }  if (state == MOVE\_RIGHT)  {  if (currentMap->isSpace(GetX2(), GetY1()) && currentMap->isSpace(GetX2(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x += STEP\_SIZE;  }  if (state == ATTACK\_LEFT)  {  if (!FireBall\_Audio\_1)  {  CAudio::Instance()->Play(17, false);  FireBall\_Audio\_1 = true;  }  AttackLeftAnimation.OnMove();  AttackVrfx.OnMove();  if (AttackVrfx.GetCurrentBitmapNumber() > 9 && AttackVrfx.GetCurrentBitmapNumber() < 15)  {  if (!FireBall\_Audio\_2)  {  CAudio::Instance()->Play(18, false);  FireBall\_Audio\_2 = true;  }  AttackFlag = true;  }  else  {  AttackFlag = false;  }  }  if (state == ATTACK\_RIGHT)  {  if (!FireBall\_Audio\_1)  {  CAudio::Instance()->Play(17, false);  FireBall\_Audio\_1 = true;  }  AttackRightAnimation.OnMove();  AttackVrfx.OnMove();  if (AttackVrfx.GetCurrentBitmapNumber() > 9 && AttackVrfx.GetCurrentBitmapNumber() < 15)  {  if (!FireBall\_Audio\_2)  {  CAudio::Instance()->Play(18, false);  FireBall\_Audio\_2 = true;  }  AttackFlag = true;  }  else  {  AttackFlag = false;  }  }  if (enemyHP <= 0 && !DeadAnimation.IsFinalBitmap())  {  state = DEAD;  DeadAnimation.OnMove();  }  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2()) && currentMap->isSpace(GetX2(), GetY2())) { // 當y座標還沒碰到地板  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  }  else {  floor = currentMap->GetBlockY(GetY2()) - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  }  void CEnemy\_Pigeon::OnShow()  {  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  switch (state)  {  case STAND\_LEFT:  animationLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animationLeft.OnShow();  break;  case STAND\_RIGHT:  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  break;  case MOVE\_LEFT:  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  break;  case ATTACK\_LEFT:  if (AttackVrfx.IsFinalBitmap())  {  AttackLeftAnimation.Reset();  AttackVrfx.Reset();  attackDelayCount = attackDelay;  AttackFlag = false;  recorded = false;  FireBall\_Audio\_1 = false;  FireBall\_Audio\_2 = false;  }  AttackLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackLeftAnimation.OnShow();  AttackVrfx.SetTopLeft(currentMap->ScreenX(RecordedX1 - 60), currentMap->ScreenY(RecordedY1 - 50));  AttackVrfx.OnShow();  break;  case MOVE\_RIGHT:  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  break;  case ATTACK\_RIGHT:  if (AttackVrfx.IsFinalBitmap())  {  AttackRightAnimation.Reset();  AttackVrfx.Reset();  attackDelayCount = attackDelay;  AttackFlag = false;  recorded = false;  FireBall\_Audio\_1 = false;  FireBall\_Audio\_2 = false;  }  AttackRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackRightAnimation.OnShow();  AttackVrfx.SetTopLeft(currentMap->ScreenX(RecordedX1 - 60), currentMap->ScreenY(RecordedY1 - 50));  AttackVrfx.OnShow();  break;  case DEAD:  if (enemyHP <= 0)  {  if (!DeadAnimation.IsFinalBitmap())  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  }  }  break;  }  }  bool CEnemy\_Pigeon::isDead()  {  if (enemyHP <= 0 && DeadAnimation.IsFinalBitmap()) return true;  else return false;  }  void CEnemy\_Pigeon::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_Scorpin: Enemy class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_Scorpoin::CEnemy\_Scorpoin(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  isMovingRight = true;  rising = false;  animation.SetDelayCount(5);  animationLeft.SetDelayCount(5);  moveRightAnimation.SetDelayCount(3);  moveLeftAnimation.SetDelayCount(3);  DeadAnimation.SetDelayCount(3);  AttackLeftAnimation.SetDelayCount(4);  AttackRightAnimation.SetDelayCount(4);  HitAnimation.SetDelayCount(2);  enemyHP = 650; //敵人預設生命值  FullHP = enemyHP;  enemyAttackDamage = 70; //敵人預設攻擊力  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  attackDelayCount = attackDelay = 150;  velocity = initial\_velocity;  state = STAND\_LEFT;  ShowLifeBarDelayCount = 0;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_Scorpoin::~CEnemy\_Scorpoin()  {  }  int CEnemy\_Scorpoin::GetX1()  {  return x;  }  int CEnemy\_Scorpoin::GetY1()  {  return y;  }  int CEnemy\_Scorpoin::GetX2()  {  return x + animation.Width();  }  int CEnemy\_Scorpoin::GetY2()  {  return y + animation.Height();  }  int CEnemy\_Scorpoin::GetWidth()  {  return animation.Width();  }  int CEnemy\_Scorpoin::GetHeight()  {  return animation.Height();  }  void CEnemy\_Scorpoin::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount == 0)  {  GetHitDelayCount = 15;  enemyHP -= damage;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_Scorpoin::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if (state == ATTACK\_RIGHT)  {  if ((GetX2() + 50 >= hero["x1"]) && (hero["x2"] >= GetX2() - 30) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag\_2) //毒液攻擊  {  \*Poison = true;  }  if ((GetX2() + 50 >= hero["x1"]) && (hero["x2"] >= GetX2() - 30) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag) //前腳攻擊  {  \*heroHP -= enemyAttackDamage;  }  }  if (state == ATTACK\_LEFT)  {  if ((GetX1() + 30 >= hero["x1"]) && (hero["x2"] >= GetX1() - 50) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag\_2) //毒液攻擊  {  \*Poison = true;  }  if ((GetX1() + 30 >= hero["x1"]) && (hero["x2"] >= GetX1() - 50) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag) //前腳攻擊  {  \*heroHP -= enemyAttackDamage;  }  }  }  CEnemy\_Action CEnemy\_Scorpoin::DetectHero(CEnemy\_Action state)  {  if (attackDelayCount <= 0)  {  if ((GetX2() - GetWidth() / 2 >= hero["x1"]) && (hero["x2"] >= GetX1() - 200) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_LEFT)  AttackLeftAnimation.Reset();  return ATTACK\_LEFT;  }  if ((GetX2() + 200 >= hero["x1"]) && (hero["x2"] >= GetX1() + GetWidth() / 2) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  if (state != ATTACK\_RIGHT)  AttackRightAnimation.Reset();  return ATTACK\_RIGHT;  }  }  if ((GetX1() - 190 >= hero["x1"]) && (hero["x2"] >= GetX1() - 270) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_LEFT;  }  if ((GetX2() + 270 >= hero["x1"]) && (hero["x2"] >= GetX2() + 190) && (GetY2() + 100 >= hero["y1"]) && (hero["y2"] >= GetY1() - 100))  {  return MOVE\_RIGHT;  }  if (state == MOVE\_LEFT || state == ATTACK\_LEFT || state == STAND\_LEFT)  {  return STAND\_LEFT;  }  if (state == MOVE\_RIGHT || state == ATTACK\_RIGHT || state == STAND\_RIGHT)  {  return STAND\_RIGHT;  }  if (state == GET\_HIT)  {  return GET\_HIT;  }  return STAND\_LEFT;  }  string CEnemy\_Scorpoin::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_Scorpoin::LoadBitmap()  {  animation.AddBitmap(IDB\_SCORPOINSTANDRIGHT\_0, RGB(255, 255, 255));  animation.AddBitmap(IDB\_SCORPOINSTANDRIGHT\_1, RGB(255, 255, 255));  animation.AddBitmap(IDB\_SCORPOINSTANDRIGHT\_2, RGB(255, 255, 255));  animationLeft.AddBitmap(IDB\_SCORPOINSTANDLEFT\_0, RGB(255, 255, 255));  animationLeft.AddBitmap(IDB\_SCORPOINSTANDLEFT\_1, RGB(255, 255, 255));  animationLeft.AddBitmap(IDB\_SCORPOINSTANDLEFT\_2, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_0, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_1, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_2, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_3, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_4, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_5, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_6, RGB(255, 255, 255));  moveRightAnimation.AddBitmap(IDB\_SCORPOINMOVERIGHT\_7, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_0, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_1, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_2, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_3, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_4, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_5, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_6, RGB(255, 255, 255));  moveLeftAnimation.AddBitmap(IDB\_SCORPOINMOVELEFT\_7, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_0, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_1, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_2, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_3, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_4, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_5, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_6, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_7, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_8, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_9, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_10, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_11, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_12, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_13, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_14, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_15, RGB(255, 255, 255));  AttackRightAnimation.AddBitmap(IDB\_SCORPOINATTAACKRIGHT\_15, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_0, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_1, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_2, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_3, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_4, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_5, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_6, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_7, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_8, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_9, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_10, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_11, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_12, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_13, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_14, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_15, RGB(255, 255, 255));  AttackLeftAnimation.AddBitmap(IDB\_SCORPOINATTAACKLEFT\_15, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SCORPOINDEAD\_0, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SCORPOINDEAD\_1, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SCORPOINDEAD\_2, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SCORPOINDEAD\_3, RGB(255, 255, 255));  DeadAnimation.AddBitmap(IDB\_SCORPOINDEAD\_4, RGB(255, 255, 255));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_Scorpoin::OnMove()  {  const int STEP\_SIZE = 2;  animation.OnMove();  animationLeft.OnMove();  moveRightAnimation.OnMove();  moveLeftAnimation.OnMove();  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  if (GetHitDelayCount > 0) GetHitDelayCount--;  else if (GetHitDelayCount == 0) HitAnimation.Reset();  if (attackDelayCount > 0) attackDelayCount--;  if (AttackRightAnimation.GetCurrentBitmapNumber() == 0 && AttackLeftAnimation.GetCurrentBitmapNumber() == 0) //攻擊中無法取消攻擊動作  {  state = DetectHero(state);  }  if (state == MOVE\_LEFT)  {  if (currentMap->isSpace(GetX1(), GetY1()) && currentMap->isSpace(GetX1(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x -= STEP\_SIZE;  }  if (state == MOVE\_RIGHT)  {  if (currentMap->isSpace(GetX2(), GetY1()) && currentMap->isSpace(GetX2(), GetY2() - 10) && !currentMap->isDoor(GetX1(), GetY1()) && !currentMap->isDoor(GetX1(), GetY2() - 10)) // 當座標還沒碰到牆  x += STEP\_SIZE;  }  if (state == ATTACK\_RIGHT)  {  AttackRightAnimation.OnMove();  if (AttackLeftAnimation.GetCurrentBitmapNumber() == 5 || AttackLeftAnimation.GetCurrentBitmapNumber() == 6)  {  AttackFlag = true;  }  else if (AttackLeftAnimation.GetCurrentBitmapNumber() == 11 || AttackLeftAnimation.GetCurrentBitmapNumber() == 12)  {  AttackFlag\_2 = true;  }  else  {  AttackFlag = false;  AttackFlag\_2 = false;  }  if (AttackRightAnimation.IsFinalBitmap())  {  attackDelayCount = attackDelay;  AttackRightAnimation.Reset();  }  }  if (state == ATTACK\_LEFT)  {  AttackLeftAnimation.OnMove();  if (AttackLeftAnimation.GetCurrentBitmapNumber() == 5 || AttackLeftAnimation.GetCurrentBitmapNumber() == 6)  {  AttackFlag = true;  }  else if (AttackLeftAnimation.GetCurrentBitmapNumber() == 11 || AttackLeftAnimation.GetCurrentBitmapNumber() == 12)  {  AttackFlag\_2 = true;  }  else  {  AttackFlag = false;  AttackFlag\_2 = false;  }  if (AttackLeftAnimation.IsFinalBitmap())  {  attackDelayCount = attackDelay;  AttackLeftAnimation.Reset();  }  }  if (enemyHP <= 0 && !DeadAnimation.IsFinalBitmap())  {  state = DEAD;  DeadAnimation.OnMove();  if (!DeadAudio)  {  DeadAudio = true;  }  }  if (rising) { // 上升狀態  if (velocity > 0) {  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  if (!currentMap->isSpace(GetX1(), GetY1()) || !currentMap->isSpace(GetX2(), GetY1())) // 當x座標碰到天花板  {  rising = false;  velocity = 1;  }  }  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 (currentMap->isSpace(GetX1(), GetY2()) && currentMap->isSpace(GetX2(), GetY2())) { // 當y座標還沒碰到地板  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  }  else {  floor = currentMap->GetBlockY(GetY2()) - GetHeight();  y = floor; // 當y座標低於地板，更正為地板上  velocity = initial\_velocity; // 重設上升初始速度  }  }  }  void CEnemy\_Scorpoin::OnShow()  {  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  switch (state)  {  case STAND\_LEFT:  animationLeft.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animationLeft.OnShow();  break;  case STAND\_RIGHT:  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  break;  case MOVE\_LEFT:  moveLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveLeftAnimation.OnShow();  break;  case ATTACK\_LEFT:  AttackLeftAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackLeftAnimation.OnShow();  break;  case MOVE\_RIGHT:  moveRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  moveRightAnimation.OnShow();  break;  case ATTACK\_RIGHT:  AttackRightAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackRightAnimation.OnShow();  break;;  case DEAD:  if (enemyHP <= 0)  {  if (!DeadAnimation.IsFinalBitmap())  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  }  }  break;  }  }  bool CEnemy\_Scorpoin::isDead()  {  if (enemyHP <= 0 && DeadAnimation.IsFinalBitmap()) return true;  else return false;  }  void CEnemy\_Scorpoin::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_Cactus: Enemy Cactus class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_Cactus::CEnemy\_Cactus(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  velocity = initial\_velocity;  AttackDelayCount = 0; // 設定攻擊頻率  AttackFlag = false;  ReadyToAttack = false;  enemyAttackDamage = 40;  AttackAnimation.SetDelayCount(4);  DeadAnimation.SetDelayCount(3);  animation.SetDelayCount(5);  GetHitAnimation.SetDelayCount(3);  HitAnimation.SetDelayCount(3);  ShowLifeBarDelayCount = 0;  enemyHP = 300;  FullHP = enemyHP;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_Cactus::~CEnemy\_Cactus(){}  int CEnemy\_Cactus::GetX1()  {  return x;  }  int CEnemy\_Cactus::GetY1()  {  return y;  }  int CEnemy\_Cactus::GetX2()  {  return x + animation.Width();  }  int CEnemy\_Cactus::GetY2()  {  return y + animation.Height();  }  int CEnemy\_Cactus::GetWidth()  {  return animation.Width();  }  int CEnemy\_Cactus::GetHeight()  {  return animation.Height();  }  string CEnemy\_Cactus::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_Cactus::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount==0)  {  GetHitAnimation.Reset();  HitAnimation.Reset();  CAudio::Instance()->Play(9, false);  enemyHP -= damage;  GetHit = true;  GetHitDelayCount = 15;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_Cactus::LoadBitmap()  {  animation.AddBitmap(IDB\_CACTUSNOMOVE\_0, RGB(255, 127, 39));  animation.AddBitmap(IDB\_CACTUSNOMOVE\_1, RGB(255, 127, 39));  animation.AddBitmap(IDB\_CACTUSNOMOVE\_2, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_0, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_1, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_2, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_3, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_4, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_5, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_6, RGB(255, 127, 39));  AttackAnimation.AddBitmap(IDB\_CACTUSATTACK\_7, RGB(255, 127, 39));  DeadAnimation.AddBitmap(IDB\_CACTUSDEAD\_0, RGB(255, 127, 39));  DeadAnimation.AddBitmap(IDB\_CACTUSDEAD\_1, RGB(255, 127, 39));  DeadAnimation.AddBitmap(IDB\_CACTUSDEAD\_2, RGB(255, 127, 39));  DeadAnimation.AddBitmap(IDB\_CACTUSDEAD\_3, RGB(255, 127, 39));  DeadAnimation.AddBitmap(IDB\_CACTUSDEAD\_4, RGB(255, 127, 39));  GetHitAnimation.AddBitmap(IDB\_CACTUSGETHIT\_0, RGB(255, 127, 39));  GetHitAnimation.AddBitmap(IDB\_CACTUSGETHIT\_1, RGB(255, 127, 39));  GetHitAnimation.AddBitmap(IDB\_CACTUSGETHIT\_2, RGB(255, 127, 39));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_3, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_4, RGB(63, 72, 204));  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  }  void CEnemy\_Cactus::OnMove()  {  const int STEP\_SIZE = 0;  if (GetHitDelayCount > 0)GetHitDelayCount--;  if (AttackDelayCount > 0) AttackDelayCount--;  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  AttackAnimation.OnMove();  GetHitAnimation.OnMove();  HitAnimation.OnMove();  animation.OnMove();  if (AttackAnimation.IsFinalBitmap()) ReadyToAttack = false;  if (ReadyToAttack && AttackDelayCount==0)  {  AttackAnimation.Reset();  AttackDelayCount = 150;  }  if (enemyHP <= 0 && !DeadAnimation.IsFinalBitmap()) DeadAnimation.OnMove();  }  void CEnemy\_Cactus::OnShow()  {  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  if (enemyHP <= 0)  {  if (!DeadAnimation.IsFinalBitmap())  {  DeadAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  DeadAnimation.OnShow();  }  }  else if (ReadyToAttack)  {  AttackAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  AttackAnimation.OnShow();  if (AttackAnimation.GetCurrentBitmapNumber() == 4)  {  CAudio::Instance()->Play(10, false);  AttackFlag = true; //發出尖刺時攻擊  }  else AttackFlag = false;  }  else if (GetHit)  {  GetHitAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  GetHitAnimation.OnShow();  HitAnimation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  HitAnimation.OnShow();  if (GetHitAnimation.IsFinalBitmap()) GetHit = false;  if (HitAnimation.IsFinalBitmap()) GetHit = false;  }  else  {  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  }  }  void CEnemy\_Cactus::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) &&(AttackDelayCount==0))  {  ReadyToAttack = true;  }  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()) && AttackFlag)  {  \*heroHP -= enemyAttackDamage;  }  }  bool CEnemy\_Cactus::isDead()  {  if (enemyHP <= 0) return true;  else return false;  }  void CEnemy\_Cactus::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // CEnemy\_Statue: Enemy Statue class  /////////////////////////////////////////////////////////////////////////////  CEnemy\_Statue::CEnemy\_Statue(gameMap\* pointer, int x, int y) : CEnemy(pointer, x, y)  {  const int INITIAL\_VELOCITY = 15; // 初始上升速度  const int FLOOR = 100; // 地板座標  floor = FLOOR;  initial\_velocity = INITIAL\_VELOCITY;  velocity = initial\_velocity;  GetHitDelayCount = 0;  HitAnimation.SetDelayCount(3);  ShowLifeBarDelayCount = 0;  enemyHP = 500;  FullHP = enemyHP;  for (int i = 0; i < 100; i++) LifeBar\_1.push\_back(new CMovingBitmap); //100個血條圖片  }  CEnemy\_Statue::~CEnemy\_Statue() {}  int CEnemy\_Statue::GetX1()  {  return x;  }  int CEnemy\_Statue::GetY1()  {  return y;  }  int CEnemy\_Statue::GetX2()  {  return x + Statue.Width();  }  int CEnemy\_Statue::GetY2()  {  return y + Statue.Height();  }  int CEnemy\_Statue::GetWidth()  {  return Statue.Width();  }  int CEnemy\_Statue::GetHeight()  {  return Statue.Height();  }  string CEnemy\_Statue::GetEnemyType()  {  return EnemyType;  }  void CEnemy\_Statue::GetAttack(const int damage)  {  if ((GetX2() >= heroAttackRange["x1"]) && (heroAttackRange["x2"] >= GetX1()) && (GetY2() >= heroAttackRange["y1"]) && (heroAttackRange["y2"] >= GetY1()) && GetHitDelayCount==0)  {  HitAnimation.Reset();  CAudio::Instance()->Play(11, false);  GetHitDelayCount = 15;  GetHit = true;  enemyHP -= FullHP / 3;  ShowLifeBarDelayCount = 150;  }  }  void CEnemy\_Statue::LoadBitmap()  {  Statue.LoadBitmap(IDB\_STATUE, RGB(255, 0, 0));  Statue\_Broken.LoadBitmap(IDB\_STATUE\_BROKEN, RGB(255, 0, 0));  HitAnimation.AddBitmap(IDB\_HIT\_0, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_1, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_2, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_3, RGB(63, 72, 204));  HitAnimation.AddBitmap(IDB\_HIT\_4, RGB(63, 72, 204));  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++) (\*i)->LoadBitmap(IDB\_ENEMYLIFEBAR\_0);  LifeBar\_0.LoadBitmap(IDB\_ENEMYLIFEBAR\_LONG);  }  void CEnemy\_Statue::OnMove()  {  if (GetHitDelayCount > 0) GetHitDelayCount--;  if (ShowLifeBarDelayCount > 0) ShowLifeBarDelayCount--;  HitAnimation.OnMove();  }  void CEnemy\_Statue::OnShow()  {  if (ShowLifeBarDelayCount != 0)  {  LifeBar\_0.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y - 5));  LifeBar\_0.ShowBitmap();  changeLifeBarLength();  }  if (enemyHP <= 0)  {  Statue\_Broken.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  Statue\_Broken.ShowBitmap();  }  else  {  Statue.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  Statue.ShowBitmap();  }  if (GetHit)  {  HitAnimation.SetTopLeft(currentMap->ScreenX(x + 5), currentMap->ScreenY(y + 50));  HitAnimation.OnShow();  if (HitAnimation.IsFinalBitmap()) GetHit = false;  }  }  void CEnemy\_Statue::AttackByEnemy(int \*heroHP, bool \*Poison)  {  if ((GetX2() >= hero["x1"]) && (hero["x2"] >= GetX1()) && (GetY2() >= hero["y1"]) && (hero["y2"] >= GetY1()))  {  \*heroHP -= enemyAttackDamage;  }  }  bool CEnemy\_Statue::isDead()  {  if (enemyHP <= 0) return true;  else return false;  }  void CEnemy\_Statue::changeLifeBarLength()  {  int xMove = currentMap->ScreenX(x);  int yMove = currentMap->ScreenY(y - 3);  int counter = 0;  float lengthOfLifeBar = ((float)enemyHP / (float)FullHP) \* 100; //重新計算血條長度  for (vector<CMovingBitmap\*>::iterator i = LifeBar\_1.begin(); i != LifeBar\_1.end(); i++)  {  if (xMove + counter < xMove + lengthOfLifeBar)  {  (\*i)->SetTopLeft(xMove + counter, yMove);  (\*i)->ShowBitmap();  }  counter += 1;  }  }  /////////////////////////////////////////////////////////////////////////////  // bullet: bullet base class  /////////////////////////////////////////////////////////////////////////////  bullet::bullet(gameMap\* point,int nx, int ny, int step)  {  x = nx;  y = ny;  currentMap = point;  distance = 0;  STEP\_SIZE = step;  animation.SetDelayCount(2);  }  bullet::~bullet() {}  int bullet::GetX1()  {  return x;  }  int bullet::GetY1()  {  return y;  }  int bullet::GetX2()  {  return x + animation.Width();  }  int bullet::GetY2()  {  return y + animation.Height();  }  bool bullet::isDelet()  {  if (distance > 500)  {  return true;  }  if (!currentMap->isSpace(GetX1(), GetY1())) // 當x座標碰到牆  {  return true;  }  return false;  }  void bullet::OnMove()  {  animation.OnMove();  x += STEP\_SIZE;  if (STEP\_SIZE > 0)  {  distance += STEP\_SIZE;  }  else  {  distance -= STEP\_SIZE;  }  }  void bullet::OnShow()  {  animation.SetTopLeft(currentMap->ScreenX(x), currentMap->ScreenY(y));  animation.OnShow();  }  /////////////////////////////////////////////////////////////////////////////  // bullet\_sunFlower : bullet\_sunFlower class  /////////////////////////////////////////////////////////////////////////////  bullet\_sunFlower::bullet\_sunFlower(gameMap\* point, int nx, int ny, int step) : bullet(point, nx, ny, step) {}  bullet\_sunFlower::~bullet\_sunFlower() {}  void bullet\_sunFlower::LoadBitmap()  {  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_0, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_1, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_2, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_3, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_4, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_5, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_6, RGB(63, 72, 204));  animation.AddBitmap(IDB\_SUNFLOWERBULLET\_7, RGB(63, 72, 204));  }  } |