

目錄

一. 簡介.....	1
1. 動機：	1
2. 分工：	1
二. 遊戲介紹	1
1. 遊戲說明：	1
2. 遊戲圖形：	2
3. 遊戲音效：	6
三. 程式設計	7
1. 程式架構：	7
2. 程式類別：	8
3. 程式技術：	9
四. 結語.....	10
1. 問題及解決方法：	10
2. 時間表(不含上課時間)：	11
3. 貢獻比例：	12

4. 自我檢核表：	12
5. 收獲：	13
6. 心得、感想：	15
7. 對於本課程的建議：無	16
附錄	17
mygame.h	17
mygame.cpp	17
Character.h	21
Character.cpp	22
Effect.h	45
Effect.cpp	45
Item.h	45
Item.cpp	46
Map.h	48
Map.cpp	49
Monster.h	61
Monster.cpp	62

一. 簡介

1. 動機：

新楓之谷是一款2005年出現於台灣並且營運至今已經15年的老遊戲，在我們小時候成為許多人的回憶，小時候因為不懂如何遊玩，因此等級總是弱弱的，沒有辦法有打倒大boss的快感，而上大學後，剛好課堂上需要製作一款遊戲，而我們想到了年幼時期的回憶，藉由這次修課的大好機會來回憶並且重製屬於我們的楓之谷。

2. 分工：



劉恒育：程式撰寫、程式規劃、遊戲規畫、技術指導

鄒承軒：程式撰寫、素材找尋、修圖訓練

二. 遊戲介紹

1. 遊戲說明：

遊玩方式：

 <p>楓之谷基本操作法</p> <p>↑ 往左, 右移動 ← →</p> <p>↑ ↓ 搖天梯, 繩索</p> <p>↑ 往其他地方移動</p> <p>Z 跳躍 X 道具 拾 Q 技能 Q W 技能 W E 技能 E R 技能 R</p>	<p>上下左右為基本控制，Z為跳，X為撿取物品。</p> <p>技能Q為攻擊怪物</p> <p>技能W為放一區塊雷電電擊</p> <p>技能E為無敵一小段時間</p> <p>技能R為瞬移一段距離</p>
 <p>使用Enter鍵可以使用道具</p>	<p>按I鍵可以打開背包，背包打開的同時不能移動，且上下左右為選取道具，enter為使用道具。</p>

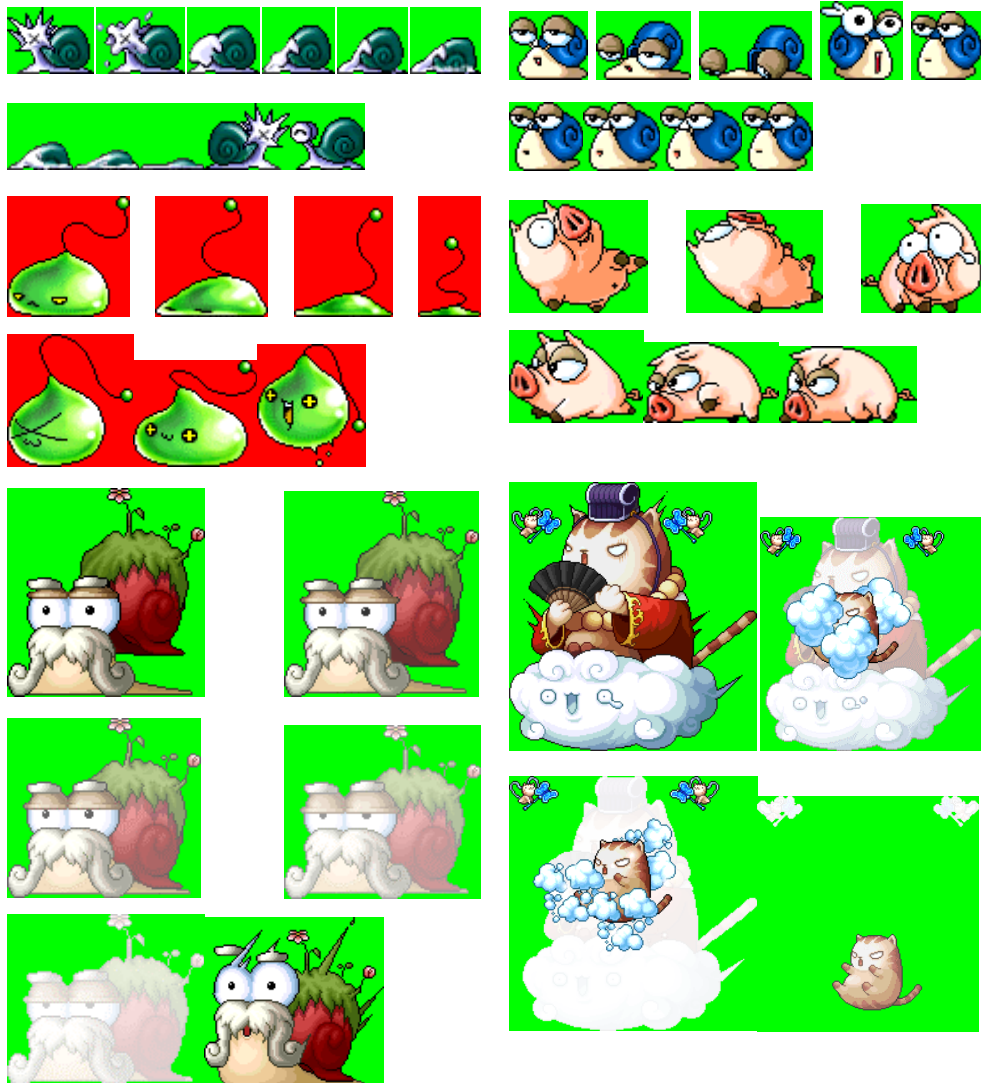
遊戲規則：利用一層一層增強的怪物，鍛鍊自己的等級，並且打倒最終boss，但在路途中須注意自己的血量，以免死亡。

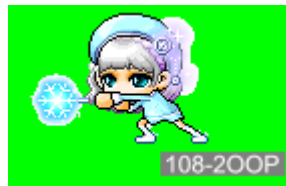
特殊功能：在梯子上按方向鍵+Z(跳)能夠跳離梯子。

密技：

    <p>作者資訊 按鍵說明 任務說明 作弊按鈕</p>	<p>使用按鍵F4可以輕輕鬆鬆升級到1000等，而且擁有10000血魔</p>
--	---

2. 遊戲圖形：





你的角色已經死亡
請按【Enter】後重生於小屋

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4

5 6 7 8 9

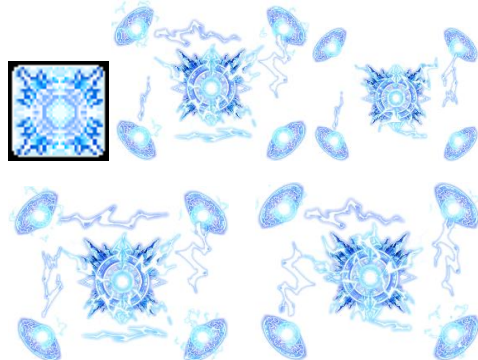


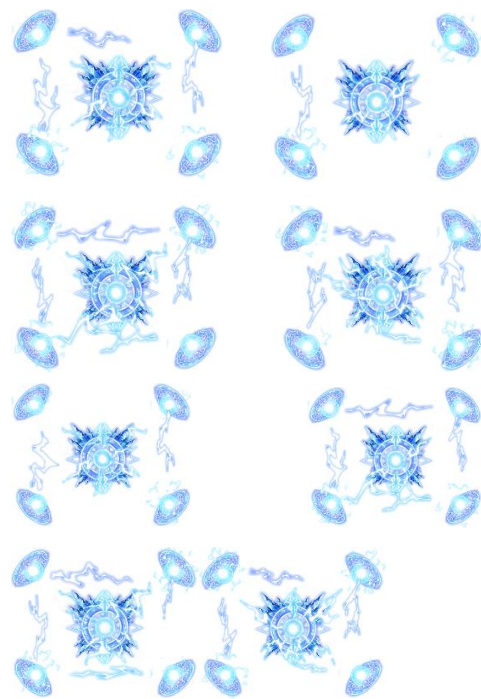
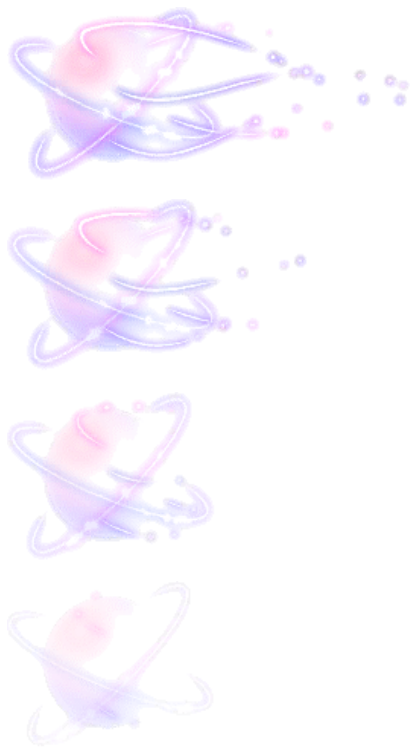
Lv.
HP
MP

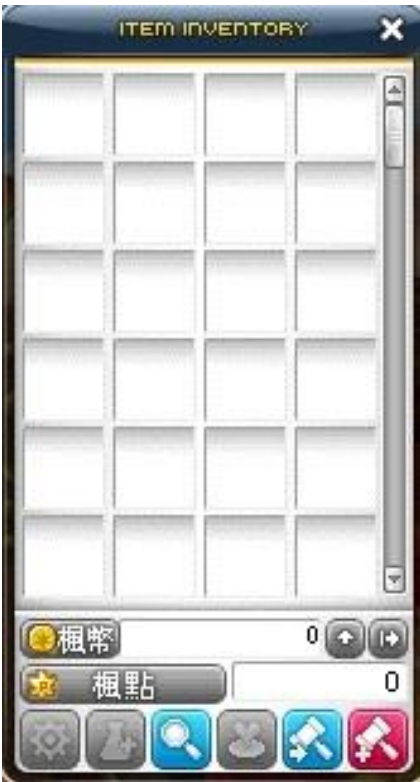
0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9 .

0







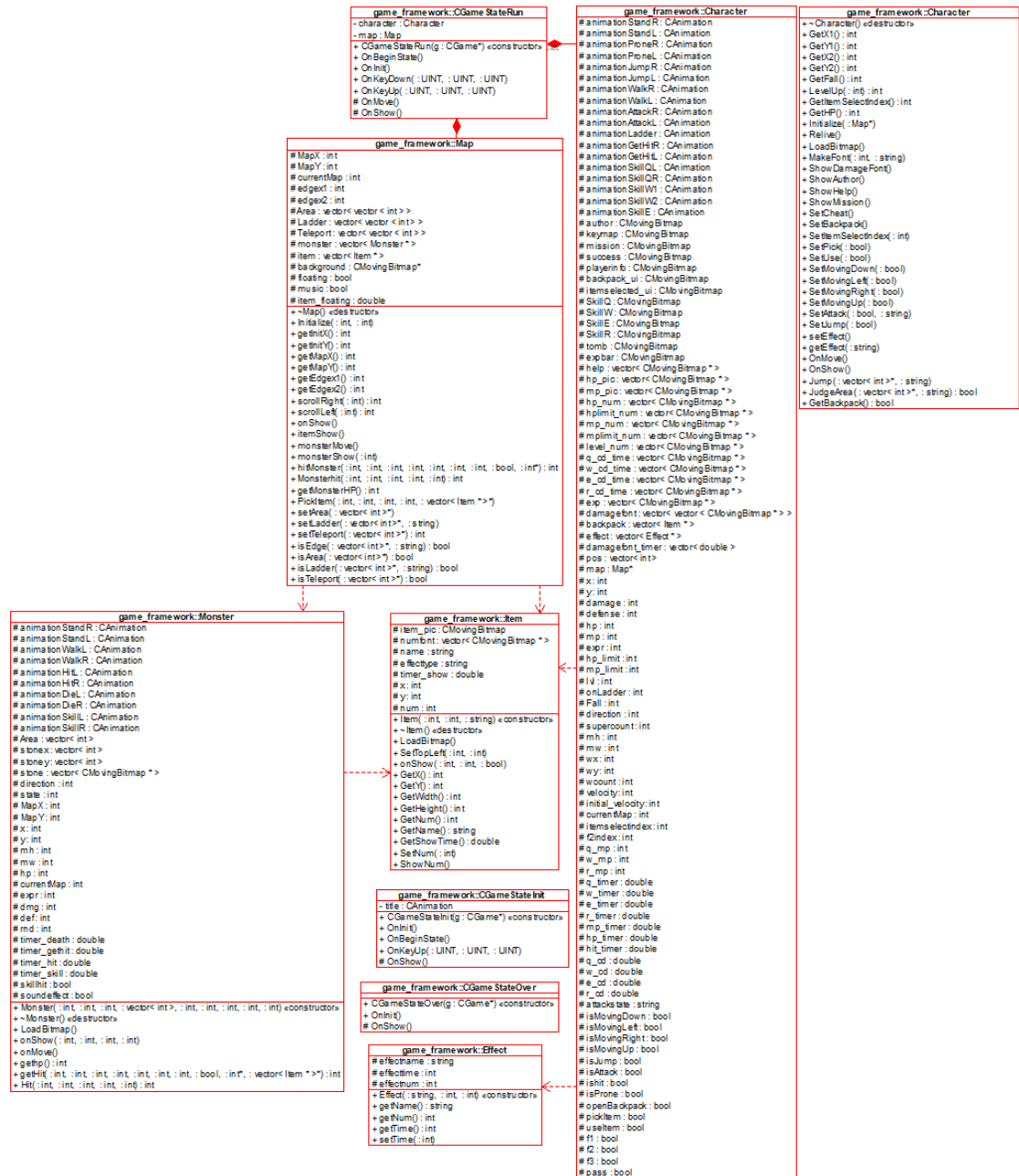
3. 遊戲音效：

音效	音效名稱
登入畫面	bgmtitle.mp3
初始小屋地圖	bgm0_1.mp3
嫩寶地圖	bgm1_1.mp3
藍寶地圖	bgm1_2.mp3
紅寶王地圖	bgm1_3.mp3
肥肥地圖	bgm2_1.mp3
綠水靈地圖	bgm2_2.mp3
喵怪仙人地圖	bgm2_3.mp3
巴洛古地圖	bgm3_1.mp3
等級提升	lvlup.wav
撿起物品	pickitem.wav
技能 Q	SkillQ.wav
技能 W	SkillW.wav
技能 E	SkillE.mp3
技能 R	SkillR.wav
解決最終 boss	success.wav
喵怪仙人攻擊	23attack.wav
巴洛古攻擊	31attack.wav

三. 程式設計

1. 程式架構：

由於Character太長，因此裁切兩部分(最右兩部分為Character)



2. 程式類別：

類別名稱	.h檔行數	.cpp檔行數	說明
Character	63	1607	取得人物基本位置及各項數值，並且判定道具使用、攻擊技能使用、檢取道具是否正在使用、播放技能音效、檢取物品音效、升等音效。
Map	45	895	判定往下一關及上一關是往哪張地圖，並且在這邊判定地圖的邊界、梯子、地圖上的檯面能不能站人、傳送點、怪物生成、物品掉落、地圖捲動、播放地圖BGN。
Monster	24	923	判定怪物的血量、是否被打到並扣血、判定怪物移動、播放技能音

			效、Boss巴洛古死亡後的通關音效。
Effect	15	36	判斷物品效果、藥水持續時間、裝備的各項數值
Item	27	152	來儲存物品的名稱和設定數值、若是在背包內則會有疊加功能可以紀錄該物品有多少個
mygame	61	266	載入遊戲、判定鍵盤按下與否、載入音樂檔案。
總行數	235	3879	

3. 程式技術：

- 1.使用重力加速度的演算法去實作跳躍與掉落的功能。
- 2.障礙物判斷使用點座標系統，以 x1y1,x2y2 的方式形成一條直線形成可站立區域。
- 3.地圖可以橫向捲動。
- 4.使用 clock()與 double 形成 timer 控制各項參數。
- 5.使用了遞迴去計算下次升級所需要的經驗值，等級越高所需經驗值也越高。
- 6.每張地圖有不一樣的音樂，使用地圖編號去撥放音樂，不需要用到 if 去判斷哪張圖該用哪首音樂。
- 7.使用 timer 改變物品在地圖上的高度做出有浮動的感覺。

- 8.每張地圖大小不同，程式會自動去判斷該地圖最左和最右的 x 座標去設定邊界，避免角色掉落。
- 9.背包內同樣的物品有自動疊加功能。
- 10.第二個 Boss 會依照人物的方位來決定向左或向右來攻擊玩家。
- 11.使用 while 與 isFinalBitmap 去完成技能顯示動畫，顯示當下鎖定玩家其他動作，直到動畫完成。

四. 結語

1. 問題及解決方法：

- 1.若在迴圈執行中對 Vector 大小進行會影響到接下來的順序，使用另一個 Vector 去紀錄以大到小需要刪除的位置，再使用迴圈去刪除。
- 2.CAnimation 如果以原本的左上角對齊的話若動畫中有些圖片的長寬不一樣，會很容易歪掉，以中間對齊的方式解決這個問題。
- 3.地圖上怪物的動作有分為站立和走動，若進到怪物的 function 再用 random 去設定動作的話，每隻怪物的動作都會一樣，因為會一直取到 random 的第一個數字，後來直接在 map 內一次 random 完所有怪物的動作再用 function 傳入來解決這個問題。
- 4.Boss 與一般小怪相比了了技能，而每個 Boss 技能也都不同，使用地圖編號去判定看這個地圖是不是 Boss 的地圖，若是 Boss 的話，就會多跑一個 Skill Function 的判斷，一般小怪則不會。
- 5.地圖會依人物位置變動而捲動，怪物的位置也要及時調整，地圖內的 onshow 會呼叫怪物的 onshow，地圖的捲動變數也會傳進怪物的 class 裡面，這樣怪物也會跟著地圖捲動而改變位置。
- 6.onMove 或 onShow 太多程式碼了，經過優化之後把能分開的程式碼分到其他的 function，避免進 onShow 或 onMove 才判斷。

2. 時間表(不含上課時間)：

週次	劉恒育	鄒承軒	說明
一	6	6	遊戲規劃討論以及尋找素材。
二	15	8	程式導入地圖，人物開始會移動，並且能偵測障礙物。
三	12	9	地圖卷軸製作，怪物生成。
四	13	8	人物技能設定，技能攻擊怪物判定，怪物血量歸零會死亡並過一段時間自動在地圖任一地方生成。
五	7	5	地圖切換，傳送點設定與偵測，換地圖會換BGM，地圖有邊界無法超過。
六	12	7	人物血量設定，受傷會出現表情變化，獲得經驗值後等級、魔力會提升，技能有冷卻時間無法無限制施放。
七	8	5	等級、血量魔量顯示在畫面，攻擊怪物時會出現傷害數字及字型。
八	6	4	血量魔量能正確顯示在畫面，有數字也有長條圖能快速辨認，技能小icon能顯示在右上角，並且伴隨著目前的冷卻時間
九	8	5	怪物死亡後會開始掉落物品，背包擁有背包介面，也能撿取掉落物，並自動放入背包排序好。
十	15	10	人物可以下梯子，梯子上按跳躍可以跳，背包內物品使用會有功能，技能魔力需求修正，改w技能Bug。
十一	8	7	新增兩張怪物地圖及怪物。

十二	12	8	新增一張怪物地圖以及兩張boss關卡，boss會有自己獨特的技能。
十三	8	7	Boss技能會判定碰撞傷害，人物技能多加E技能(無敵)，人物吃盾牌時能夠增加防禦力。
十四	15	10	音效補足，裝備掉落會隨等級怪物提升，新增R技能(瞬移)。
十五	12	10	程式優化處理，將memory leak處理掉，並將about、遊戲說明，遊戲作弊按鍵補上。
總行數	157	109	

3. 貢獻比例：

劉恒育：50%

鄒承軒：50%

4. 自我檢核表：

	項目	完成否	無法完成的原因
1	解決Memory leak	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
2	自訂遊戲Icon	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
3	全螢幕啟動	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
4	有About畫面	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
5	初始畫面說明按鍵及滑鼠之用法與密技	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
6	上傳setup/source檔	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
7	Setup檔可正確執行	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
8	報告字型、點數、對齊、行距、頁碼等格式正確	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	

9	報告封面、側邊格式正確	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	
10	報告附錄程式格式正確	<input checked="" type="checkbox"/> 已完成 <input type="checkbox"/> 未完成	

5. 收穫：

劉恒育：

經過這次實作遊戲的經驗，讓我學到許多新的東西及技巧，以前都只是寫一些小小的程式，很多做法在這次的實作遊戲中不一定行的通，其中令我印象深刻的地方有：

1.遊戲座標的判斷:因為我們本身的地圖寬度是大於遊戲畫面的寬度，所以需要常常去換算這兩者的位置，以免人物已經走出畫面外了，地圖還是沒有捲動，或是怪物明明在畫面外，卻跑進畫面裡面的狀況。

2.怪物與人物的碰撞判斷:這跟遊戲座標的判斷有關係，人物會一直顯示在畫面中，所以人物的座標是以畫面座標為主，而怪物則會佈滿的整個地圖，所以是以地圖座標為主，碰撞偵測要先將兩者的座標換算好，另外還有左右之分，人物在怪物的左邊，要對怪物左邊的點進行偵測，若人物在右邊，則要使用怪物右邊的點進行偵測。

3.Clock()的應用:我們實作的遊戲很多都是要靠時間去判斷是否能執行，或是能不能顯示，所以我們藉由clock()這項函數，搭配一個double的變數去做出一個timer，來計算技能的冷卻時間、執行時間或是怪物攻擊的冷卻時間等。

4.Random函數:當我們需要對怪物random他的動作時，怪物有自己的一個class，若地圖上有10隻怪物，每隻的動作都是進到怪物的function內才進行random的動作，random函數會一直取第一個數字，導致所有怪物相同，所以我們必須要地圖的class內一次取完所有的random數字，再傳進怪物的class裡面。

5.重力加速度之演算法運用:原本還在擔心跳躍與落下的速度看起來很不真實,後來參考老師一開始的寫法,讓我認識到重力加速度的演算法,也把他學了起來並運用在遊戲之中。

6.除錯技巧:這次遊戲對我來說是我目前寫過最大型的城市,很複雜,程式碼也很多,一開始在寫的時候都是把當週進度所有東西都一次寫好在測試,但是這樣可能會發生這邊有問題,影響到了另一樣功能,這樣除錯都要跳來跳去,也會導致思緒跟著變得複雜,所以後來都是每完成一樣功能就進行測試,沒問題之後在新增下一樣功能,所有功能完成後再進行一次統一測試,不僅有效率,也可以確定程式的完整性。

鄒承軒：

這次受到寫遊戲的陶冶,練習許多以前不曾遇到過的問題,從前通常只是寫一個程式達到老師目的而已,而這次是寫一個遊戲並且容易遇到許多不同且難以察覺的問題,例如:

1. 「 new 」,每次在new完後,一定要記得delete,不然程式打完並執行後往往會出現memory leak,而且有時候寫得太多,可能還會忘記自己在那曾經new過,然後突然出現一大堆memory leak到時可能又得熬夜爆肝修bug,非常不值得。

2. 「 onshow順序 », onshow 雖然是個很簡單的問題,不過剛開始寫的時候想說,為什麼loadbitmap後卻沒有東西顯示出來,後來找了程式碼的bug找了很久,才發現是在這個簡單的環節出了錯,load的圖被大的地圖給蓋掉了。

3. 「 音樂,音效的應用 »,有許多音效非常難找,但背景音樂因為原廠楓之谷就有許多source可以取得,但是那種類似怪物的音效就只能夠自己上網慢慢找類似的,並且東拼西湊才能為一個完整的動作配完音。

4. 「為圖片或音樂設定enum編號」，設定好的enum編號可以大大地增進執行效率，才不會因為忘記圖片或者音樂的編號而因為當初亂設定，而要跑回去看，浪費了許多找的時間。
5. 「修圖技巧」，有些特定的圖片並不是都很完善，有時候載下來後，可能某一張因為怪物放技能而造成圖片破圖或顯示不出來，這時候修圖技巧就很重要了，利用上一張與下一張圖片的共通點，再加上一點點的變化，便能夠成功地一份動圖給修好了，放上遊戲裡簡直完美無缺。

6. 心得、感想：

劉恒育：

這次的物件導向設計實習這門課程非常的充實，從每個星期都要花掉一天的假日去完成每個星期的進度，雖然很累，有時候也覺得很煩，但是當完成該週的進度時，又會覺得非常的有成就感，而且能看到每個星期一點一滴的把遊戲建立起來，到最後的成品，也是一件非常有趣的事情，以前想踏入資訊領域的關鍵原因就是因為我很喜歡玩遊戲，我也很嚮往有一天能寫出一個遊戲，而這門課程也算是完成我一個小小的夢想，能看到一個遊戲從無到有，也深刻的體會到，可能遊戲中一個小小的判斷，就要花費許多的程式碼與想法去實現，但是我覺得其中最痛苦的就是圖片沒有透明通道和新增圖片進專案了，由於 gameframework 已經是十幾年前的東西了，我覺得如果用更好的框架去做遊戲，會輕鬆許多，也可以把遊戲做的更好，寫程式碼的過程中，也讓我了解到遊戲的架構，還有每個物件的關係，如果關係沒有處理好，Coding 起來就會非常困難，常常沒辦法對應到你想要的物件，也因為這門課，我覺得我的程式能力進步了許多。

鄒承軒：

二上時修過老師的「物件導向程式設計」，在那時因為有做過買衣服的程式，因此而幫我奠定了許多關於繼承、class分割的觀念，為我下學期的課程奠定了基礎。

下學期的「物件導向程式設計實習」這門課程雖然讓我起初在做的時候感覺到許多挫折感，從剛開始的Eracer程式開始看如何撰寫一套遊戲程式，那時有些程式碼看都不知道是在做什麼用的，經過一段時間的鑽研後，開始修改到能夠由使用者獨立控制的一個角色，再來變成一個完整的、有開始、有過程、有結束、有過關、有音效的遊戲，不過這中間的心路歷程也是十分痛苦，每週都有每週程式的進度，需要週週添加小功能才能讓這個程式完整、完善，不過卻也因為每週都得添加功能而讓放學後的生活充滿著OOP，有時甚至會因為Bug解不出來而熬夜，犧牲了我們資工系珍貴的睡眠時間，不過卻也因為這些努力，而能將這款從期初定下的遊戲，從剛開始的有橡皮擦又有球在跳的範例程式變成一款小型的楓之谷，看到這個成果便覺得自己的努力似乎沒有白費了。

7. 對於本課程的建議：無

附錄

mygame.h
<pre>#include "Map.h" #include "Character.h" enum AUDIO_ID { bgmtitle,//0 bgm01,//1 bgm11,//2 bgm12,//3 bgm13,//4 bgm21,//5 bgm22,//6 bgm23,//7 bgm31,//8 skillq,//9 skillw,//10 skille,//11 skillr,//12 pickitem,//13 levelup,//14 attack23,//15 attack31,//16 success,//17 }; namespace game_framework { class CGameStateInit : public CGameState { public: CGameStateInit(CGame* g); void OnInit(); void OnBeginState(); void OnKeyUp(UINT, UINT, UINT); protected: void OnShow(); private: CAnimation title; }; class CGameStateRun : public CGameState { public: CGameStateRun(CGame* g); void OnBeginState(); void OnInit(); void OnKeyDown(UINT, UINT, UINT); void OnKeyUp(UINT, UINT, UINT); protected: void OnMove(); void OnShow(); private: Character character; Map map; }; class CGameStateOver : public CGameState { public: CGameStateOver(CGame* g); void OnInit(); protected: void OnShow(); }; }</pre>
mygame.cpp
<pre>#include "stdafx.h" #include "Resource.h" #include <mmsystem.h> #include <draw.h> #include <vector></pre>

```

#include <string>
#include <sstream>
#include <iomanip>
#include "audio.h"
#include "gamelib.h"
#include "mygame.h"
namespace game_framework
{
    CGameStateInit::CGameStateInit(CGame* g)
        : CGameState(g)
    {
    }
    void CGameStateInit::OnInit()
    {
        ShowInitProgress(0);
        CAudio::Instance()->Load(bgmtitle, "sounds\\bgmtitle.mp3");
        CAudio::Instance()->Load(bgm01, "sounds\\bgm0_1.mp3");
        CAudio::Instance()->Load(bgm11, "sounds\\bgm1_1.mp3");
        CAudio::Instance()->Load(bgm12, "sounds\\bgm1_2.mp3");
        CAudio::Instance()->Load(bgm13, "sounds\\bgm1_3.mp3");
        CAudio::Instance()->Load(bgm21, "sounds\\bgm2_1.mp3");
        CAudio::Instance()->Load(bgm22, "sounds\\bgm2_2.mp3");
        CAudio::Instance()->Load(bgm23, "sounds\\bgm2_3.mp3");
        CAudio::Instance()->Load(bgm31, "sounds\\bgm3_1.mp3");
        CAudio::Instance()->Load(skillq, "sounds\\SkillQ.wav");
        CAudio::Instance()->Load(skillw, "sounds\\SkillW.wav");
        CAudio::Instance()->Load(skillr, "sounds\\SkillE.mp3");
        CAudio::Instance()->Load(skillr, "sounds\\SkillR.wav");
        CAudio::Instance()->Load(pickitem, "sounds\\pickitem.wav");
        CAudio::Instance()->Load(levelup, "sounds\\lvlup.wav");
        CAudio::Instance()->Load(attack23, "sounds\\23attack.wav");
        CAudio::Instance()->Load(attack31, "sounds\\31attack.wav");
        CAudio::Instance()->Load(success, "sounds\\success.wav");
        CAudio::Instance()->Play(bgmtitle, true);
        title.AddBitmap(IDB_title_0);
        title.AddBitmap(IDB_title_1);
        title.SetTopLeft(0 + title.Width() / 2, 0 + title.Height() / 2);
        title.SetDelayCount(10);
        title.Reset();
    }
    void CGameStateInit::OnBeginState()
    {
    }
    void CGameStateInit::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
    {
        const char KEY_SPACE = ' ';

        if (nChar == KEY_SPACE)
            GotoGameState(GAME_STATE_RUN);
    }
    void CGameStateInit::OnShow()
    {
        title.OnShow();
        title.OnMove();
    }
    CGameStateOver::CGameStateOver(CGame* g)
        : CGameState(g)
    {
    }
    void CGameStateOver::OnInit()
    {
        ShowInitProgress(66);
        ShowInitProgress(100);
    }
    void CGameStateOver::OnShow()
    {
        CDC* pDC = CDDraw::GetBackCDC();
        CFont f, *fp;
        f.CreatePointFont(160, "Times New Roman");
        fp = pDC->SelectObject(&f);
        pDC->SetBkColor(RGB(0, 0, 0));
        pDC->SetTextColor(RGB(255, 255, 0));
    }
}

```



```

        char str[80];
        pDC->TextOut(240, 210, str);
        pDC->SelectObject(fp);
        CDDraw::ReleaseBackCDC();
    }
    CGameStateRun::CGameStateRun(CGame* g)
        : CGameState(g) {}
    void CGameStateRun::OnBeginState()
    {
    }
    void CGameStateRun::OnMove()
    {
        map.monsterMove();
        character.OnMove();
    }
    void CGameStateRun::OnInit()
    {
        ShowInitProgress(33);
        character.LoadBitmap();
        character.Initialize(&map);
        ShowInitProgress(50);
    }
    void CGameStateRun::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
    {
        const char KEY_ENTER = 0x0D;
        const char KEY_BACKPACK = 0x49;
        const char KEY_PICK = 0x58;
        const char KEY_LEFT = 0x25;
        const char KEY_UP = 0x26;
        const char KEY_RIGHT = 0x27;
        const char KEY_DOWN = 0x28;
        const char KEY_JUMP = 0x5A;
        const char KEY_SKILLQ = 0x51;
        const char KEY_SKILLW = 0x57;
        const char KEY_SKILLE = 0x45;
        const char KEY_SKILLR = 0x52;
        const char KEY_F1 = 0x70;
        const char KEY_F2 = 0x71;
        const char KEY_F3 = 0x72;
        const char KEY_F4 = 0x73;
        if(nChar==KEY_F1){
            character.ShowAuthor();
        }
        if (nChar == KEY_F2) {
            character.ShowHelp();
        }
        if (nChar == KEY_F3) {
            character.ShowMission();
        }
        if (nChar == KEY_F4) {
            character.SetCheat();
        }
        if (nChar == KEY_ENTER)
        {
            character.SetUse(true);
        }
        if (character.GetHP() <= 0) {
            return;
        }
        if (nChar == KEY_BACKPACK)
        {
            character.SetBackpack();
        }
        if (nChar == KEY_PICK)
        {
            character.SetPick(true);
        }
        if (nChar == KEY_SKILLQ)
        {
            character.SetAttack(true, "Q");
        }
        if (nChar == KEY_SKILLW)

```

```

{
    character.SetAttack(true, "W");
}
if (nChar == KEY_SKILLE)
{
    character.SetAttack(true, "E");
}
if (nChar == KEY_SKILLR) {
    character.SetAttack(true, "R");
}
if (nChar == KEY_LEFT)
{
    if (character.GetBackpack())
    {
        if (character.GetItemSelectIndex() > 0)
        {
            character.SetItemSelectIndex(character.GetItemSelectIndex() - 1);
        }
    }
    else
    {
        character.SetMovingLeft(true);
        character.SetMovingRight(false);
    }
}
if (nChar == KEY_RIGHT)
{
    if (character.GetBackpack())
    {
        if (character.GetItemSelectIndex() < 23)
        {
            character.SetItemSelectIndex(character.GetItemSelectIndex() + 1);
        }
    }
    else
    {
        character.SetMovingRight(true);
        character.SetMovingLeft(false);
    }
}
if (nChar == KEY_UP)
{
    if (character.GetBackpack())
    {
        if (character.GetItemSelectIndex() >= 4)
        {
            character.SetItemSelectIndex(character.GetItemSelectIndex() - 4);
        }
    }
    else
    {
        character.SetMovingUp(true);
        character.SetMovingDown(false);
    }
}
if (nChar == KEY_DOWN)
{
    if (character.GetBackpack())
    {
        if (character.GetItemSelectIndex() <= 19)
        {
            character.SetItemSelectIndex(character.GetItemSelectIndex() + 4);
        }
    }
    else
    {
        character.SetMovingDown(true);
        character.SetMovingUp(false);
    }
}
if (nChar == KEY_JUMP)
{

```

<pre> character.SetJump(true); } } void CGameStateRun::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags) { const char KEY_ENTER = 0x0D; const char KEY_PICK = 0x58; const char KEY_LEFT = 0x25; const char KEY_UP = 0x26; const char KEY_RIGHT = 0x27; const char KEY_DOWN = 0x28; if (nChar == KEY_ENTER) { character.SetUse(false); } if (nChar == KEY_PICK) { character.SetPick(false); } if (nChar == KEY_LEFT) { character.SetMovingLeft(false); } if (nChar == KEY_RIGHT) { character.SetMovingRight(false); } if (nChar == KEY_UP) { character.SetMovingUp(false); } if (nChar == KEY_DOWN) { character.SetMovingDown(false); } } void CGameStateRun::OnShow() { map.onShow(); map.itemShow(); map.monsterShow(character.GetX1()); character.OnShow(); } } </pre>	
	<div>Character.h</div> <pre> #pragma once #include "Map.h" #include "Effect.h" #include "Item.h" namespace game_framework { class Character { public: ~Character(); int GetX1(); int GetY1(); int GetX2(); int GetY2(); int GetFall(); int LevelUp(int); int GetItemSelectIndex(); int GetHP(); void Initialize(Map*); void Relive(); void LoadBitmap(); void MakeFont(int, string); void ShowDamageFont(); void ShowAuthor(); void ShowHelp(); }; } </pre>

```

void ShowMission();
void SetCheat();
void SetBackpack();
void SetItemSelectedIndex(int);
void SetPick(bool);
void SetUse(bool);
void SetMovingDown(bool);
void SetMovingLeft(bool);
void SetMovingRight(bool);
void SetMovingUp(bool);
void SetAttack(bool, string);
void SetJump(bool);
void setEffect();
void getEffect(string);
void OnMove();
void OnShow();
void Jump(vector<int>*, string);
bool JudgeArea(vector<int>*, string);
bool GetBackpack();

protected:
    CAnimation animationStandR, animationStandL, animationProneR, animationProneL, animationJumpR,
animationJumpL, animationWalkR, animationWalkL, animationAttackR, animationAttackL, animationLadder,
animationGetHitR, animationGetHitL;
    CAnimation animationSkillQL, animationSkillQR, animationSkillW1, animationSkillW2, animationSkillE;
    CMovingBitmap author, keymap, mission, success, playerinfo, backpack_ui, itemselected_ui, SkillQ, SkillW, SkillE,
SkillR, tomb, expbar;
    vector<CMovingBitmap*> help, hp_pic, mp_pic, hp_num, hplimit_num, mp_num, mplimit_num, level_num,
q_cd_time, w_cd_time, e_cd_time, r_cd_time, exp;
    vector<vector<CMovingBitmap*>> damagefont;
    vector<Item*> backpack;
    vector<Effect*> effect;
    vector<double> damagefont_timer;
    vector<int> pos;
    Map* map;
    int x, y, damage, defense, hp, mp, expr, hp_limit, mp_limit, lvl, onLadder, Fall, direction, supercount;
    int mh, mw, wx, wy, wcount, velocity, initial_velocity, currentMap, itemselectindex, f2index;
    int q_mp, w_mp, r_mp;
    double q_timer, w_timer, e_timer, r_timer, mp_timer, hp_timer, hit_timer, q_cd, w_cd, e_cd, r_cd;
    string attackstate;
    bool isMovingDown, isMovingLeft, isMovingRight, isMovingUp, isJump, isAttack, ishit, isProne, openBackpack,
pickItem, useItem, f1, f2, f3, pass;
};
}

```

Character.cpp

```

#include "stdafx.h"
#include "Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "audio.h"
#include "gamelib.h"
#include "Character.h"
#include <vector>
#include <string>
#include <sstream>
#include <iomanip>
namespace game_framework
{
    Character::~Character()
    {
        for (int i = 0; i < (int)help.size(); i++)
        {
            delete help[i];
        }
        for (int i = 0; i < (int)backpack.size(); i++)
        {
            delete backpack[i];
        }
        for (int i = 0; i < (int)effect.size(); i++)
        {
            delete effect[i];
        }
    }
}

```

```

        for (int i = 0; i < (int)hp_pic.size(); i++)
        {
            delete hp_pic[i];
        }
        for (int i = 0; i < (int)mp_pic.size(); i++)
        {
            delete mp_pic[i];
        }
        for (int i = 0; i < (int)level_num.size(); i++)
        {
            delete level_num[i];
        }
        for (int i = 0; i < (int)hp_num.size(); i++)
        {
            delete hp_num[i];
        }
        for (int i = 0; i < (int)hplimit_num.size(); i++)
        {
            delete hplimit_num[i];
        }
        for (int i = 0; i < (int)mp_num.size(); i++)
        {
            delete mp_num[i];
        }
        for (int i = 0; i < (int)mplimit_num.size(); i++)
        {
            delete mplimit_num[i];
        }
        for (int i = 0; i < (int)q_cd_time.size(); i++)
        {
            delete q_cd_time[i];
        }
        for (int i = 0; i < (int)w_cd_time.size(); i++)
        {
            delete w_cd_time[i];
        }
        for (int i = 0; i < (int)e_cd_time.size(); i++)
        {
            delete e_cd_time[i];
        }
        for (int i = 0; i < (int)r_cd_time.size(); i++)
        {
            delete r_cd_time[i];
        }
        for (int i = 0; i < (int)damagefont.size(); i++)
        {
            for (int j = 0; j < (int)damagefont[i].size(); j++)
            {
                delete damagefont[i][j];
            }
        }
        for (int i = 0; i < (int)exp.size(); i++)
        {
            delete exp[i];
        }
    }
    int Character::GetX1()
    {
        return x;
    }
    int Character::GetY1()
    {
        return y;
    }
    int Character::GetX2()
    {
        return x + animationStandR.Width();
    }
    int Character::GetY2()
    {
        return y + animationStandR.Height();
    }
}

```

```

int Character::GetFall()
{
    return Fall;
}
int Character::LevelUp(int lvl)
{
    if (lvl == 1)
    {
        return 100;
    }
    else
    {
        return LevelUp(lvl - 1) + 200;
    }
}
int Character::GetItemSelectIndex()
{
    return itemselectindex;
}
int Character::GetHP()
{
    return hp;
}
void Character::Initialize(Map* MAP)
{
    map = MAP;
    damage = defense = 0;
    lvl = 1;
    hp = hp_limit = 750;
    mp = mp_limit = 450;
    q_mp = 30;
    w_mp = 150;
    r_mp = 30;
    q_cd = 300;
    w_cd = 2000;
    e_cd = 15000;
    r_cd = 500;
    velocity = initial_velocity = 12;
    hp_timer = mp_timer = q_timer = w_timer = e_timer = r_timer = -1;
    supercount = wcount = -1;
    Fall = onLadder = direction = currentMap = expr = itemselectindex = f2index = 0;
    f1 = f2 = f3 = pass = openBackpack = pickItem = useItem = isJump = isMovingLeft = isMovingRight = isMovingUp
= isMovingDown = isAttack = isProne = false;
    map->Initialize(currentMap, 1);
    mh = animationStandR.Height() / 2;
    mw = animationStandR.Width() / 2;
    x = map->getInitX() + map->getMapX();
    y = map->getInitY() + map->getMapY() - 70;
}
void Character::Relive()
{
    hp = hp_limit;
    mp = mp_limit;
    hp_timer = mp_timer = q_timer = w_timer = e_timer = -1;
    supercount = wcount = -1;
    velocity = initial_velocity = 12;
    Fall = onLadder = currentMap = direction = 0;
    openBackpack = pickItem = useItem = isJump = isMovingLeft = isMovingRight = isMovingUp = isMovingDown =
isProne = isAttack = false;
    map->Initialize(currentMap, 1);
    x = map->getInitX() + map->getMapX();
    y = map->getInitY() + map->getMapY() - 70;
}
void Character::LoadBitmap()
{
    author.LoadBitmap(IDB_author, RGB(0, 255, 0));
    keymap.LoadBitmap(IDB_keymap, RGB(0, 255, 0));
    mission.LoadBitmap(IDB_help_4, RGB(0, 255, 0));
    success.LoadBitmap(IDB_success, RGB(0, 255, 0));
    playerinfo.LoadBitmap(IDB_playerinfo, RGB(0, 255, 0));
    for (int i = 0; i < 4; i++) {
        help.push_back(new CMovingBitmap);
    }
}

```



```

        help[i]->LoadBitmap(652 + i, RGB(0, 255, 0));
    }
    for (int i = 0; i < 100; i++)
    {
        hp_pic.push_back(new CMovingBitmap);
        mp_pic.push_back(new CMovingBitmap);
        hp_pic[i]->LoadBitmap(IDB_hp, RGB(0, 255, 0));
        mp_pic[i]->LoadBitmap(IDB_mp, RGB(0, 255, 0));
    }
    backpack_ui.LoadBitmap(IDB_backpack_ui);
    itemselected_ui.LoadBitmap(IDB_ItemSelectedUI, RGB(255, 255, 255));
    tomb.LoadBitmap(IDB_tomb, RGB(0, 255, 0));
    expbar.LoadBitmap(IDB_expbar);
    animationStandR.AddBitmap(IDB_standright_0, RGB(0, 255, 0));
    animationStandR.AddBitmap(IDB_standright_1, RGB(0, 255, 0));
    animationStandR.AddBitmap(IDB_standright_2, RGB(0, 255, 0));
    animationStandR.AddBitmap(IDB_standright_3, RGB(0, 255, 0));
    animationStandL.AddBitmap(IDB_standleft_0, RGB(0, 255, 0));
    animationStandL.AddBitmap(IDB_standleft_1, RGB(0, 255, 0));
    animationStandL.AddBitmap(IDB_standleft_2, RGB(0, 255, 0));
    animationStandL.AddBitmap(IDB_standleft_3, RGB(0, 255, 0));
    animationProneR.AddBitmap(IDB_proneright_0, RGB(0, 255, 0));
    animationProneL.AddBitmap(IDB_proneleft_0, RGB(0, 255, 0));
    animationJumpR.AddBitmap(IDB_jumpright_0, RGB(0, 255, 0));
    animationJumpL.AddBitmap(IDB_jumpleft_0, RGB(0, 255, 0));
    animationWalkR.AddBitmap(IDB_walkright_0, RGB(0, 255, 0));
    animationWalkR.AddBitmap(IDB_walkright_1, RGB(0, 255, 0));
    animationWalkR.AddBitmap(IDB_walkright_2, RGB(0, 255, 0));
    animationWalkR.AddBitmap(IDB_walkright_3, RGB(0, 255, 0));
    animationWalkL.AddBitmap(IDB_walkleft_0, RGB(0, 255, 0));
    animationWalkL.AddBitmap(IDB_walkleft_1, RGB(0, 255, 0));
    animationWalkL.AddBitmap(IDB_walkleft_2, RGB(0, 255, 0));
    animationWalkL.AddBitmap(IDB_walkleft_3, RGB(0, 255, 0));
    animationAttackR.AddBitmap(IDB_attackright_0, RGB(0, 255, 0));
    animationAttackR.AddBitmap(IDB_attackright_1, RGB(0, 255, 0));
    animationAttackR.AddBitmap(IDB_attackright_2, RGB(0, 255, 0));
    animationAttackL.AddBitmap(IDB_attackleft_0, RGB(0, 255, 0));
    animationAttackL.AddBitmap(IDB_attackleft_1, RGB(0, 255, 0));
    animationAttackL.AddBitmap(IDB_attackleft_2, RGB(0, 255, 0));
    animationLadder.AddBitmap(IDB_ladder_0, RGB(0, 255, 0));
    animationLadder.AddBitmap(IDB_ladder_1, RGB(0, 255, 0));
    animationGetHitR.AddBitmap(IDB_hitright_0, RGB(0, 255, 0));
    animationGetHitL.AddBitmap(IDB_hitleft_0, RGB(0, 255, 0));
    SkillQ.LoadBitmap(IDB_SkillQ);
    SkillW.LoadBitmap(IDB_SkillW);
    SkillE.LoadBitmap(IDB_SkillE);
    SkillR.LoadBitmap(IDB_SkillR);
    animationSkillQL.AddBitmap(IDB_SkillQleft_0, RGB(255, 255, 255));
    animationSkillQL.AddBitmap(IDB_SkillQleft_1, RGB(255, 255, 255));
    animationSkillQL.AddBitmap(IDB_SkillQleft_2, RGB(255, 255, 255));
    animationSkillQL.AddBitmap(IDB_SkillQleft_3, RGB(255, 255, 255));
    animationSkillQL.AddBitmap(IDB_SkillQleft_4, RGB(255, 255, 255));
    animationSkillQR.AddBitmap(IDB_SkillQright_0, RGB(255, 255, 255));
    animationSkillQR.AddBitmap(IDB_SkillQright_1, RGB(255, 255, 255));
    animationSkillQR.AddBitmap(IDB_SkillQright_2, RGB(255, 255, 255));
    animationSkillQR.AddBitmap(IDB_SkillQright_3, RGB(255, 255, 255));
    animationSkillQR.AddBitmap(IDB_SkillQright_4, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_0, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_1, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_2, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_3, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_4, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_5, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_6, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_7, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_8, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_9, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_10, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_11, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_12, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_13, RGB(255, 255, 255));
    animationSkillW1.AddBitmap(IDB_SkillW1_14, RGB(255, 255, 255));

```

```

animationSkillW1.AddBitmap(IDB_SkillW1_15, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_0, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_1, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_2, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_3, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_4, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_5, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_6, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_7, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_8, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_9, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_10, RGB(255, 255, 255));
animationSkillW2.AddBitmap(IDB_SkillW2_11, RGB(255, 255, 255));
animationSkillE.AddBitmap(IDB_SkillE1_0, RGB(0, 255, 0));
animationSkillE.AddBitmap(IDB_SkillE1_1, RGB(0, 255, 0));
animationSkillE.AddBitmap(IDB_SkillE1_2, RGB(0, 255, 0));
animationSkillE.AddBitmap(IDB_SkillE1_3, RGB(0, 255, 0));
animationAttackR.SetDelayCount(3);
animationAttackL.SetDelayCount(3);
animationSkillQR.SetDelayCount(3);
animationSkillQL.SetDelayCount(3);
animationSkillW1.SetDelayCount(3);
animationSkillW2.SetDelayCount(3);
animationSkillE.SetDelayCount(4);
animationAttackR.Reset();
animationAttackL.Reset();
animationSkillQR.Reset();
animationSkillQL.Reset();
animationSkillW1.Reset();
animationSkillW2.Reset();
animationSkillE.Reset();
}
void Character::MakeFont(int num, string type)
{
    string mkfont;
    if (type == "HP")
    {
        mkfont = to_string(num);

        for (int i = 0; i < (int)hp_num.size(); i++)
        {
            delete hp_num[i];
        }
        hp_num.clear();
        for (int i = 0; i < (int)mkfont.size(); i++)
        {
            hp_num.push_back(new CMovingBitmap);
            hp_num[i]->LoadBitmap(335 + mkfont[i] - 48, RGB(0, 255, 0));
        }
    }
    else if (type == "MP")
    {
        mkfont = to_string(num);

        for (int i = 0; i < (int)mp_num.size(); i++)
        {
            delete mp_num[i];
        }
        mp_num.clear();
        for (int i = 0; i < (int)mkfont.size(); i++)
        {
            mp_num.push_back(new CMovingBitmap);
            mp_num[i]->LoadBitmap(335 + mkfont[i] - 48, RGB(0, 255, 0));
        }
    }
    else if (type == "HPLimit")
    {
        mkfont = to_string(num);
        for (int i = 0; i < (int)hplimit_num.size(); i++)
        {
            delete hplimit_num[i];
        }
    }
}

```

```

        hplimit_num.clear();
        hplimit_num.push_back(new CMovingBitmap);
        hplimit_num[0]->LoadBitmap(IDB_mphp_line, RGB(0, 255, 0));
        for (int i = 1; i < (int)mkfont.size() + 1; i++)
        {
            hplimit_num.push_back(new CMovingBitmap);
            hplimit_num[i]->LoadBitmap(335 + mkfont[i - 1] - 48, RGB(0, 255, 0));
        }
    }
    else if (type == "MPLimit")
    {
        mkfont = to_string(num);
        for (int i = 0; i < (int)mplimit_num.size(); i++)
        {
            delete mplimit_num[i];
        }
        mplimit_num.clear();
        mplimit_num.push_back(new CMovingBitmap);
        mplimit_num[0]->LoadBitmap(IDB_mphp_line, RGB(0, 255, 0));
        for (int i = 1; i < (int)mkfont.size() + 1; i++)
        {
            mplimit_num.push_back(new CMovingBitmap);
            mplimit_num[i]->LoadBitmap(335 + mkfont[i - 1] - 48, RGB(0, 255, 0));
        }
    }
    else if (type == "Damage")
    {
        mkfont = to_string(num);
        vector<CMovingBitmap*> dmg_vector;
        int init_x, init_y;
        init_x = x + (animationStandR.Width() / 2) - ((mkfont.length() * 38) / 2);
        init_y = y - 15 - 59;
        for (int i = 0; i < (int)mkfont.size(); i++)
        {
            dmg_vector.push_back(new CMovingBitmap);
            dmg_vector[i]->LoadBitmap(302 + mkfont[i] - 48, RGB(0, 255, 0));
            dmg_vector[i]->SetTopLeft(init_x + i * 38, init_y);
        }
        damagefont.push_back(dmg_vector);
        damagefont_timer.push_back(clock());
    }
    else if (type == "level")
    {
        mkfont = to_string(num);
        for (int i = 0; i < (int)level_num.size(); i++)
        {
            delete level_num[i];
        }
        level_num.clear();
        for (int i = 0; i < (int)mkfont.size(); i++)
        {
            level_num.push_back(new CMovingBitmap);
            level_num[i]->LoadBitmap(312 + mkfont[i] - 48, RGB(0, 255, 0));
        }
    }
    else if (type == "Q_cd")
    {
        ostringstream stream;
        stream << fixed << setprecision(1);
        stream << (double)num / 1000;
        mkfont = stream.str();
        for (int i = 0; i < (int)q_cd_time.size(); i++)
        {
            delete q_cd_time[i];
        }
        q_cd_time.clear();
        for (int i = 0; i < (int)mkfont.size(); i++)
        {
            q_cd_time.push_back(new CMovingBitmap);

            if (mkfont[i] == '.')
            {

```

```

        q_cd_time[i]->LoadBitmap(IDB_skill_dot, RGB(0, 255, 0));
    }
    else
    {
        q_cd_time[i]->LoadBitmap(324 + mkfont[i] - 48, RGB(0, 255, 0));
    }
}
else if (type == "W_cd")
{
    ostringstream stream;
    stream << fixed << setprecision(1);
    stream << (double)num / 1000;
    mkfont = stream.str();
    for (int i = 0; i < (int)w_cd_time.size(); i++)
    {
        delete w_cd_time[i];
    }
    w_cd_time.clear();
    for (int i = 0; i < (int)mkfont.size(); i++)
    {
        w_cd_time.push_back(new CMovingBitmap);
        if (mkfont[i] == '.')
        {
            w_cd_time[i]->LoadBitmap(IDB_skill_dot, RGB(0, 255, 0));
        }
        else
        {
            w_cd_time[i]->LoadBitmap(324 + mkfont[i] - 48, RGB(0, 255, 0));
        }
    }
}
else if (type == "E_cd")
{
    ostringstream stream;
    stream << fixed << setprecision(1);
    stream << (double)num / 1000;
    mkfont = stream.str();
    for (int i = 0; i < (int)e_cd_time.size(); i++)
    {
        delete e_cd_time[i];
    }
    e_cd_time.clear();
    for (int i = 0; i < (int)mkfont.size(); i++)
    {
        if (mkfont[i] == '.')
        {
            break;
        }
        else
        {
            e_cd_time.push_back(new CMovingBitmap);
            e_cd_time[i]->LoadBitmap(324 + mkfont[i] - 48, RGB(0, 255, 0));
        }
    }
}
else if (type == "R_cd")
{
    ostringstream stream;
    stream << fixed << setprecision(1);
    stream << (double)num / 1000;
    mkfont = stream.str();
    for (int i = 0; i < (int)r_cd_time.size(); i++)
    {
        delete r_cd_time[i];
    }
    r_cd_time.clear();
    for (int i = 0; i < (int)mkfont.size(); i++)
    {
        r_cd_time.push_back(new CMovingBitmap);
        if (mkfont[i] == '.')
        {

```

```

        r_cd_time[i]->LoadBitmap(IDB_skill_dot, RGB(0, 255, 0));
    }
    else
    {
        r_cd_time[i]->LoadBitmap(324 + mkfont[i] - 48, RGB(0, 255, 0));
    }
}
}

void Character::ShowDamageFont()
{
    int del = -1;
    for (int i = 0; i < (int)damagefont.size(); i++)
    {
        for (int j = 0; j < (int)damagefont[i].size(); j++)
        {
            if (clock() - damagefont_timer[i] <= 400)
            {
                damagefont[i][j]->SetTopLeft(damagefont[i][j]->Left(), damagefont[i][j]->Top() - (int)((clock() - damagefont_timer[i]) / 100) * 2));
                damagefont[i][j]->ShowBitmap();
            }
        }
        if (clock() - damagefont_timer[i] >= 1500)
        {
            del = i;
        }
    }
    if (del != -1)
    {
        for (int i = 0; i < del + 1; i++)
        {
            for (int j = 0; j < (int)damagefont[i].size(); j++)
            {
                delete damagefont[i][j];
            }
            damagefont_timer.erase(damagefont_timer.begin(), damagefont_timer.begin() + del + 1);
            damagefont.erase(damagefont.begin(), damagefont.begin() + del + 1);
        }
    }
}

void Character::ShowAuthor()
{
    f1 = !f1;
    if (f1) {
        f2 = false;
        f3 = false;
    }
}

void Character::ShowHelp()
{
    f2 = !f2;
    if (f2) {
        f2index = 0;
        f1 = false;
        f3 = false;
    }
}

void Character::ShowMission()
{
    f3 = !f3;
    if (f3) {
        f1 = false;
        f2 = false;
    }
}

void Character::SetCheat()
{
    lv1 = 1000;
    damage = 10000;
    defense = 10000;
}

```

```

        hp = 10000;
        hp_limit = 10000;
        mp = 10000;
        mp_limit = 10000;
    }
    void Character::SetBackpack()
    {
        openBackpack = !openBackpack;
    }
    void Character::SetItemSelectIndex(int index)
    {
        itemselectindex = index;
    }
    void Character::SetPick(bool flag)
    {
        pickItem = flag;
    }
    void Character::SetUse(bool flag)
    {
        useItem = flag;
    }
    void Character::SetMovingDown(bool flag)
    {
        isMovingDown = flag;
        if (!flag) {
            isProne = flag;
        }
    }
    void Character::SetMovingLeft(bool flag)
    {
        if (isAttack && attackstate == "Q")
        {
            isMovingLeft = false;
        }
        else
        {
            isMovingLeft = flag;
        }
    }
    void Character::SetMovingRight(bool flag)
    {
        if (isAttack && attackstate == "Q")
        {
            isMovingRight = false;
        }
        else
        {
            isMovingRight = flag;
        }
    }
    void Character::SetMovingUp(bool flag)
    {
        isMovingUp = flag;
    }
    void Character::SetAttack(bool flag, string s)
    {
        isAttack = false;
        attackstate = s;
        if (attackstate == "Q" && mp >= q_mp && (clock() - q_timer >= q_cd || q_timer == -1) && q_timer != 1 &&
onLadder == 0)
        {
            isAttack = flag;
            q_timer = 1;
            isMovingLeft = false;
            isMovingRight = false;
            isJump = false;
            CAudio::Instance()->Play(9);
        }
        if (attackstate == "W" && mp >= w_mp && (clock() - w_timer >= w_cd || w_timer == -1) && w_timer != 1)
        {
            isAttack = flag;
            wcount = 0;

```



```

        mp -= w_mp;
        w_timer = 1;
        wx = x + mw - map->getMapX();
        wy = y - 125 - map->getMapY();
        animationSkillW1.SetTopLeft(wx, wy);
        animationSkillW2.SetTopLeft(wx, wy);
        animationSkillW1.Reset();
        animationSkillW2.Reset();
        CAudio::Instance()->Play(10);
    }
    if (attackstate == "E" && supercount == -1 && (clock() - e_timer >= e_cd || e_timer == -1) && e_timer != 1) {
        e_timer = 1;
        supercount = 0;
        isAttack = flag;
        CAudio::Instance()->Play(11);
    }
    if (attackstate == "R" && mp >= r_mp && (clock() - r_timer >= r_cd || r_timer == -1) && r_timer != 1 && onLadder
== 0 && !isProne)
    {
        isAttack = flag;
        mp -= r_mp;
        r_timer = 1;
        CAudio::Instance()->Play(12);
    }
}
void Character::SetJump(bool flag)
{
    if (isAttack && attackstate == "Q")
    {
        isJump = false;
    }
    else
    {
        isJump = flag;
    }
}
void Character::setEffect()
{
    bool flag = false;
    backpack[itemselectindex]->SetNum(backpack[itemselectindex]->GetNum() - 1);
    useItem = false;

    if (backpack[itemselectindex]->GetName() == "紅色藥水")
    {
        for (int i = 0; i < (int)effect.size(); i++)
        {
            if (effect[i]->getName() == backpack[itemselectindex]->GetName())
            {
                effect[i]->setTime(10);
                flag = true;
            }
        }
        if (!flag)
        {
            effect.push_back(new Effect(backpack[itemselectindex]->GetName(), 10, 40));
        }
    }
    if (backpack[itemselectindex]->GetName() == "超級紅色藥水")
    {
        for (int i = 0; i < (int)effect.size(); i++)
        {
            if (effect[i]->getName() == backpack[itemselectindex]->GetName())
            {
                effect[i]->setTime(10);
                flag = true;
            }
        }
        if (!flag)
        {
            effect.push_back(new Effect(backpack[itemselectindex]->GetName(), 10, 80));
        }
    }
}

```

```

else if (backpack[itemselectindex]->GetName() == "藍色藥水")
{
    for (int i = 0; i < (int)effect.size(); i++)
    {
        if (effect[i]->getName() == backpack[itemselectindex]->GetName())
        {
            effect[i]->setTime(10);
            flag = true;
        }
    }
    if (!flag)
    {
        effect.push_back(new Effect(backpack[itemselectindex]->GetName(), 10, 50));
    }
}
else if (backpack[itemselectindex]->GetName() == "超級藍色藥水")
{
    for (int i = 0; i < (int)effect.size(); i++)
    {
        if (effect[i]->getName() == backpack[itemselectindex]->GetName())
        {
            effect[i]->setTime(10);
            flag = true;
        }
    }
    if (!flag)
    {
        effect.push_back(new Effect(backpack[itemselectindex]->GetName(), 10, 100));
    }
}
else if (backpack[itemselectindex]->GetName() == "長劍 0")
{
    damage += 20;
}
else if (backpack[itemselectindex]->GetName() == "長劍 1")
{
    damage += 50;
}
else if (backpack[itemselectindex]->GetName() == "長劍 2")
{
    damage += 75;
}
else if (backpack[itemselectindex]->GetName() == "長劍 3")
{
    damage += 200;
}
else if (backpack[itemselectindex]->GetName() == "盾牌 0")
{
    defense += 10;
}
else if (backpack[itemselectindex]->GetName() == "盾牌 1")
{
    defense += 30;
}
else if (backpack[itemselectindex]->GetName() == "上衣 0")
{
    hp_limit += 20;
}
else if (backpack[itemselectindex]->GetName() == "上衣 1")
{
    hp_limit += 50;
}
else if (backpack[itemselectindex]->GetName() == "上衣 2")
{
    hp_limit += 100;
}
if (backpack[itemselectindex]->GetNum() == 0)
{
    delete backpack[itemselectindex];
    backpack.erase(backpack.begin() + itemselectindex);
}
}

```

```

void Character::getEffect(string type)
{
    for (int i = 0; i < (int)effect.size(); i++)
    {
        if (type == "hp")
        {
            if (effect[i]->getName() == "紅色藥水")
            {
                hp += effect[i]->getNum();
                effect[i]->setTime(-1);
            }
            if (effect[i]->getName() == "超級紅色藥水")
            {
                hp += effect[i]->getNum();
                effect[i]->setTime(-1);
            }
        }
        else if (type == "mp")
        {
            if (effect[i]->getName() == "藍色藥水")
            {
                mp += effect[i]->getNum();
                effect[i]->setTime(-1);
            }
            if (effect[i]->getName() == "超級藍色藥水")
            {
                mp += effect[i]->getNum();
                effect[i]->setTime(-1);
            }
        }
    }
    for (int i = (int)effect.size() - 1; i >= 0; i--)
    {
        if (effect[i]->getTime() == 0)
        {
            delete effect[i];
            effect.erase(effect.begin() + i);
        }
    }
}

void Character::OnShow()
{
    vector<int> del;
    playerinfo.SetTopLeft(338, 677);
    playerinfo.ShowBitmap();
    SkillQ.SetTopLeft(824, 0);
    SkillQ.ShowBitmap();
    SkillW.SetTopLeft(874, 0);
    SkillW.ShowBitmap();
    SkillE.SetTopLeft(924, 0);
    SkillE.ShowBitmap();
    SkillR.SetTopLeft(974, 0);
    SkillR.ShowBitmap();
    keymap.SetTopLeft(0, 768 - keymap.Height());
    keymap.ShowBitmap();
    MakeFont(lvl, "level");
    if (useItem) {
        if (f1) {
            f1 = false;
            useItem = false;
        }
        else if (f2) {
            f2index++;
            if (f2index > 3) {
                f2 = false;
                f2index = 0;
            }
            useItem = false;
        }
        else if (f3) {
            f3 = false;
            useItem = false;
        }
    }
}

```

```

    }
    if (currentMap == 7 && map->getMonsterHP() <= 0) {
        pass = true;
        useItem = false;
    }
}
for (int i = 0; i < (int)level_num.size(); i++)
{
    level_num[i]->SetTopLeft(405 + 16 * i, 685);
    level_num[i]->ShowBitmap();
}
int tmp = (int)ceil((double)hp / hp_limit * 100);
if (tmp > 100) {
    tmp = 100;
}
for (int i = 0; i < tmp; i++)
{
    hp_pic[i]->SetTopLeft(379 + 3 * i, 717);
    hp_pic[i]->ShowBitmap();
}
tmp = (int)ceil((double)mp / mp_limit * 100);
if (tmp > 100) {
    tmp = 100;
}
for (int i = 0; i < tmp; i++)
{
    mp_pic[i]->SetTopLeft(379 + 3 * i, 741);
    mp_pic[i]->ShowBitmap();
}
MakeFont(hp, "HP");
for (int i = (int)hp_num.size() - 1; i >= 0; i--)
{
    hp_num[i]->SetTopLeft(519 - ((int)hp_num.size() - 1 - i) * 11, 720);
    hp_num[i]->ShowBitmap();
}
MakeFont(mp, "MP");
for (int i = (int)mp_num.size() - 1; i >= 0; i--)
{
    mp_num[i]->SetTopLeft(519 - ((int)mp_num.size() - 1 - i) * 11, 744);
    mp_num[i]->ShowBitmap();
}
MakeFont(hp_limit, "HPLimit");
for (int i = (int)hplimit_num.size() - 1; i >= 0; i--)
{
    hplimit_num[i]->SetTopLeft(529 + i * 11, 720);
    hplimit_num[i]->ShowBitmap();
}
MakeFont(mp_limit, "MPLimit");
for (int i = (int)mplimit_num.size() - 1; i >= 0; i--)
{
    mplimit_num[i]->SetTopLeft(529 + i * 11, 744);
    mplimit_num[i]->ShowBitmap();
}
expbar.SetTopLeft(0, 763);
expbar.ShowBitmap();
for (int i = 0; i < (int)exp.size(); i++) {
    delete exp[i];
}
exp.clear();
int temp = (int)ceil((double)expr / LevelUp(lvl) * 128);
if (temp > 128) {
    temp = 128;
}
for (int i = 0; i < temp; i++) {
    exp.push_back(new CMovingBitmap);
    exp[i]->LoadBitmap(IDB_exp);
    exp[i]->SetTopLeft(8 * i, 763);
    exp[i]->ShowBitmap();
}
if (hp <= 0) {
    hp = 0;
    if (tomb.Top() < y - 29) {

```

```

        tomb.SetTopLeft(x, tomb.Top() + 15);
    }
    else {
        vector<int> pos;
        pos.push_back(tomb.Left());
        pos.push_back(tomb.Top() + 29);
        JudgeArea(&pos, "tomb");
        tomb.SetTopLeft(x, pos[1] - 29);
        if (useItem) {
            Relive();
        }
    }
    tomb.ShowBitmap();
    CMovingBitmap die_dialog;
    die_dialog.LoadBitmap(IDB_die_dialog);
    die_dialog.SetTopLeft(256, 304);
    die_dialog.ShowBitmap();
    return;
}
if (openBackpack)
{
    backpack_ui.SetTopLeft(858, 75);
    backpack_ui.ShowBitmap();

    for (int i = 0; i < (int)backpack.size(); i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (backpack[i]->GetName() == backpack[j]->GetName())
            {
                backpack[j]->SetNum(backpack[j]->GetNum() + 1);
                del.push_back(i);
                break;
            }
        }
    }

    for (int i = (int)del.size() - 1; i >= 0; i--)
    {
        delete backpack[del[i]];
        backpack.erase(backpack.begin() + del[i]);
    }
    for (int i = 0; i < (int)backpack.size(); i++)
    {
        backpack[i]->SetTopLeft(867 + (i % 4) * 36, 103 + (i / 4) * 35);
        backpack[i]->onShow(0, 0, false);
        backpack[i]->ShowNum();
    }
    itemselected_ui.SetTopLeft(865 + (itemselectindex % 4) * 36, 101 + (itemselectindex / 4) * 35);
    itemselected_ui.ShowBitmap();
}
if (useItem && openBackpack && itemselectindex < (int)backpack.size())
{
    setEffect();
}
ShowDamageFont();
if (q_cd - (clock() - q_timer) > 0 && q_timer != -1 && q_timer != 1)
{
    MakeFont(int(q_cd - (clock() - q_timer)), "Q_cd");

    for (int i = 0; i < (int)q_cd_time.size(); i++)
    {
        q_cd_time[i]->SetTopLeft(830 + i * 12, 50);
        q_cd_time[i]->ShowBitmap();
    }
}
if (w_cd - (clock() - w_timer) > 0 && w_timer != -1 && w_timer != 1)
{
    MakeFont(int(w_cd - (clock() - w_timer)), "W_cd");

    for (int i = 0; i < (int)w_cd_time.size(); i++)
    {
        w_cd_time[i]->SetTopLeft(880 + i * 12, 50);
    }
}

```

```

        w_cd_time[i]->ShowBitmap();
    }
}
if (e_cd - (clock() - e_timer) > 0 && e_timer != -1 && e_timer != 1)
{
    MakeFont(int(e_cd - (clock() - e_timer)), "E_cd");

    for (int i = 0; i < (int)e_cd_time.size(); i++)
    {
        e_cd_time[i]->SetTopLeft(935 + i * 12, 50);
        e_cd_time[i]->ShowBitmap();
    }
}
if (r_cd - (clock() - r_timer) > 0 && r_timer != -1 && r_timer != 1)
{
    MakeFont(int(r_cd - (clock() - r_timer)), "R_cd");

    for (int i = 0; i < (int)r_cd_time.size(); i++)
    {
        r_cd_time[i]->SetTopLeft(985 + i * 12, 50);
        r_cd_time[i]->ShowBitmap();
    }
}
if (w_timer == 1)
{
    if (wcount == 3)
    {
        animationSkillW2.SetTopLeft(wx + map->getMapX(), wy + map->getMapY());
        animationSkillW2.OnShow();
    }
    else
    {
        animationSkillW1.SetTopLeft(wx + map->getMapX(), wy + map->getMapY());
        animationSkillW1.OnShow();
    }
}
if (e_timer == 1) {
    animationSkillE.SetTopLeft(x + mw, y - mh * 2);
    animationSkillE.OnShow();
}
if (isJump)
{
    if (direction == 0)
    {
        animationJumpR.SetTopLeft(x + mw, y + mh);
        animationJumpR.OnShow();
    }
    else
    {
        animationJumpL.SetTopLeft(x + mw, y + mh);
        animationJumpL.OnShow();
    }
}
else if (q_timer == 1)
{
    if (direction == 0)
    {
        animationAttackR.SetTopLeft(x + mw, y + mh);
        animationAttackR.OnShow();

        if (animationAttackR.IsFinalBitmap())
        {
            animationSkillQR.SetTopLeft(x + animationStandR.Width() + (animationSkillQL.Width() / 2), y +
(animationSkillQL.Height() / 2));
            animationSkillQR.OnShow();
        }
    }
    else
    {
        animationAttackL.SetTopLeft(x + mw, y + mh);
        animationAttackL.OnShow();
    }
}

```

```

        if (animationAttackL.IsFinalBitmap())
        {
            animationSkillQL.SetTopLeft(x - (animationSkillQL.Width() / 2), y + (animationSkillQL.Height() /
2));
            animationSkillQL.OnShow();
        }
    }
}
else if (isMovingRight && onLadder == 0)
{
    animationWalkR.SetTopLeft(x + mw, y + mh);
    animationWalkR.OnShow();
    direction = 0;
}
else if (isMovingLeft && onLadder == 0)
{
    animationWalkL.SetTopLeft(x + mw, y + mh);
    animationWalkL.OnShow();
    direction = 1;
}
else if (isMovingDown && onLadder == 0 && !isJump && Fall == 0)
{
    if (direction == 0)
    {
        animationProneR.SetTopLeft(x + mw, y + mh);
        animationProneR.OnShow();
    }
    else
    {
        animationProneL.SetTopLeft(x + mw, y + mh);
        animationProneL.OnShow();
    }
}
else if (onLadder == 1)
{
    animationLadder.SetTopLeft(x + mw, y + mh);
    animationLadder.OnShow();
}
else if (direction == 0 && onLadder == 0)
{
    if (ishit && clock() - hit_timer <= 1000)
    {
        animationGetHitR.SetTopLeft(x + mw, y + mh);
        animationGetHitR.OnShow();
    }
    else
    {
        animationStandR.SetTopLeft(x + mw, y + mh);
        animationStandR.OnShow();
    }
}
else if (direction == 1 && onLadder == 0)
{
    if (ishit && clock() - hit_timer <= 1000)
    {
        animationGetHitL.SetTopLeft(x + mw, y + mh);
        animationGetHitL.OnShow();
    }
    else
    {
        animationStandL.SetTopLeft(x + mw, y + mh);
        animationStandL.OnShow();
    }
}
}
if (currentMap == 7 && map->getMonsterHP() <= 0 && !pass) {
    success.SetTopLeft((1024 - success.Width()) / 2, (768 - success.Height()) / 2);
    success.ShowBitmap();
}
if (f1) {
    author.SetTopLeft(252, 256);
    author.ShowBitmap();
}
}

```

```

        if (f2) {
            help[f2index]->SetTopLeft((1024 - help[f2index]->Width()) / 2, (768 - help[f2index]->Height()) / 2);
            help[f2index]->ShowBitmap();
        }
        if (f3) {
            mission.SetTopLeft((1024 - mission.Width()) / 2, (768 - mission.Height()) / 2);
            mission.ShowBitmap();
        }
    }
    void Character::OnMove()
    {
        if (hp <= 0) {
            return;
        }
        int totaldmg = 0, getdmg = map->Monsterhit(x, y, x + animationStandR.Width(), y + animationStandR.Height(),
defense);
        if (supercount != -1) {
            getdmg = 0;
        }
        if (getdmg != 0)
        {
            hp -= getdmg;
            getdmg = 0;
            ishit = true;
            hit_timer = clock();
            if (hp <= 0) {
                tomb.SetTopLeft(x, 0);
                tomb.ShowBitmap();
            }
        }
        if (clock() - mp_timer >= 1000 && hp != 0)
        {
            getEffect("mp");
            mp += lvl;
            mp_timer = clock();

            if (mp > mp_limit)
            {
                mp = mp_limit;
            }
        }
        if (clock() - hp_timer >= 1000 && hp != 0)
        {
            getEffect("hp");
            hp += 3 * lvl;
            hp_timer = clock();

            if (hp > hp_limit)
            {
                hp = hp_limit;
            }
        }
        if (expr >= LevelUp(lvl))
        {
            CAudio::Instance()->Play(14);
            expr -= LevelUp(lvl);
            lvl++;
            hp_limit += 50;
            mp_limit += 30;
            hp = hp_limit;
            mp = mp_limit;
            q_mp += 15;
            w_mp += 30;
        }
        if (pickItem)
        {
            map->PickItem(x - map->getMapX(), y - map->getMapY(), animationStandR.Width(),
animationStandR.Height(), &backpack);
        }
        pos.clear();
        pos.push_back(x);
        pos.push_back(y);
    }

```



```

JudgeArea(&pos, "null");
if (w_timer == 1)
{
    if (wcount == 3)
    {
        animationSkillW2.OnMove();

        if (animationSkillW2.IsFinalBitmap())
        {
            expr += map->hitMonster(x, y, animationSkillW1.Left(), animationSkillW1.Top(),
animationSkillW1.Left() + animationSkillW1.Width(), animationSkillW1.Top() + animationSkillW1.Height(), 150 + damage
+ 25 * (lvl - 1), animationSkillW2.IsFinalBitmap(), &totaldmg);
            isAttack = false;
            w_timer = clock();

            if (totaldmg > 0)
            {
                MakeFont(totaldmg, "Damage");
            }
            animationSkillW1.Reset();
            animationSkillW2.Reset();
            wcount = -1;
        }
    }
    else if (animationSkillW1.IsFinalBitmap())
    {
        expr += map->hitMonster(x, y, animationSkillW1.Left(), animationSkillW1.Top(),
animationSkillW1.Left() + animationSkillW1.Width(), animationSkillW1.Top() + animationSkillW1.Height(), 50 + damage +
25 * (lvl - 1), animationSkillW1.IsFinalBitmap(), &totaldmg);
        if (totaldmg > 0)
        {
            MakeFont(totaldmg, "Damage");
        }
        if (wcount < 3)
        {
            wcount++;
            animationSkillW1.Reset();
        }
    }
    else
    {
        animationSkillW1.OnMove();
    }
}
if (e_timer == 1) {
    animationSkillE.OnMove();
    if (animationSkillE.IsFinalBitmap()) {
        supercount++;
        animationSkillE.Reset();
    }
    if (supercount == 5) {
        supercount = -1;
        animationSkillE.Reset();
        isAttack = false;
        e_timer = clock();
    }
}
if (q_timer == 1)
{
    if (direction == 0)
    {
        if (animationAttackR.IsFinalBitmap())
        {
            animationSkillQR.OnMove();
            expr += map->hitMonster(x, y, animationSkillQR.Left(), animationSkillQR.Top(),
animationSkillQR.Left() + animationSkillQR.Width(), animationSkillQR.Top() + (animationSkillQR.Height() / 2), 100 +
damage + 25 * (lvl - 1), animationSkillQR.IsFinalBitmap(), &totaldmg);

            if (animationSkillQR.IsFinalBitmap())
            {
                isAttack = false;
                animationAttackR.Reset();
            }
        }
    }
}

```

```

        animationSkillQR.Reset();
        q_timer = clock();
        mp -= q_mp;

        if (totaldmg > 0)
        {
            MakeFont(totaldmg, "Damage");
        }
    }
}
else
{
    animationAttackR.OnMove();
}
}
else
{
    if (animationAttackL.IsFinalBitmap())
    {
        animationSkillQL.OnMove();
        expr += map->hitMonster(x, y, animationSkillQL.Left(), animationSkillQL.Top(),
animationSkillQL.Left() + animationSkillQL.Width(), animationSkillQL.Top() + (animationSkillQL.Height() / 2), 100 +
damage + 25 * (lvl - 1), animationSkillQL.IsFinalBitmap(), &totaldmg);

        if (animationSkillQL.IsFinalBitmap())
        {
            isAttack = false;
            animationAttackL.Reset();
            animationSkillQL.Reset();
            q_timer = clock();
            mp -= q_mp;

            if (totaldmg > 0)
            {
                MakeFont(totaldmg, "Damage");
            }
        }
    }
    else
    {
        animationAttackL.OnMove();
    }
}
}
if (isAttack&&attackstate == "R") {
    if (direction == 0) {
        if (JudgeArea(&pos, "RR"))
        {
            direction = 0;
            x = pos[0];
            y = pos[1];
        }
    }
    else {
        if (JudgeArea(&pos, "RL"))
        {
            direction = 1;
            x = pos[0];
            y = pos[1];
        }
    }
}
}
if (isMovingLeft)
{
    if (JudgeArea(&pos, "left"))
    {
        direction = 1;
        x = pos[0];
        y = pos[1];
    }
}
}

```

```

else if (isMovingRight)
{
    if (JudgeArea(&pos, "right"))
    {
        direction = 0;
        x = pos[0];
        y = pos[1];
    }
}
else if (isMovingUp)
{
    if (JudgeArea(&pos, "up"))
    {
        x = pos[0];
        y = pos[1];
    }
}
else if (isMovingDown)
{
    if (JudgeArea(&pos, "down"))
    {
        x = pos[0];
        y = pos[1];
    }
}
else if (direction == 0)
{
    if (ishit && clock() - hit_timer <= 1000)
    {
        animationGetHitR.OnMove();
    }
    else
    {
        ishit = false;
        animationStandR.OnMove();
    }
}
else if (direction == 1)
{
    if (ishit && clock() - hit_timer <= 1000)
    {
        animationGetHitL.OnMove();
    }
    else
    {
        ishit = false;
        animationStandL.OnMove();
    }
}
if (isJump)
{
    if (onLadder == 1)
    {
        onLadder = 0;
        velocity = 6;
    }

    if (Fall == 1)
    {
        isJump = false;
    }
    else if (JudgeArea(&pos, "jump"))
    {
        y = pos[1];
    }
}
if (Fall == 1)
{
    if (JudgeArea(&pos, "fall"))
    {
        if (map->isArea(&pos))
        {

```

```

        Fall = 0;
        velocity = initial_velocity;
        isJump = false;
    }

    x = pos[0];
    y = pos[1];
}
}
}
void Character::Jump(vector<int>* pos, string move)
{
    if (move == "jump")
    {
        if (velocity > initial_velocity)
        {
            velocity = initial_velocity;
        }

        if (velocity > 0)
        {
            (*pos)[1] -= velocity;
            velocity--;
        }
        else
        {
            Fall = 1;
            velocity = 1;
        }
    }
    else
    {
        if (velocity < 18)
        {
            velocity++;
        }

        (*pos)[1] += velocity;
    }
}
}
bool Character::JudgeArea(vector<int>* pos, string dir)
{
    if (!isJump && onLadder == 0)
    {
        if (!map->isArea(pos))
        {
            if (Fall == 1)
            {
                isJump = false;
            }
            else
            {
                Fall = 1;
                velocity = 1;
            }
        }
    }

    if (dir == "tomb") {
        if (!map->isArea(pos))
        {
            (*pos)[1] += 15;
        }
    }

    if (dir == "RR"&&onLadder == 0) {
        if (!map->isEdge(pos, "RR"))
        {
            if ((*pos)[0] + 250 <= 600)
            {
                (*pos)[0] += 250;
            }
            else {
                (*pos)[0] += map->scrollRight(250);
            }
        }
    }
}

```

```

        }
        if ((*pos)[0] - map->getMapX() > map->getEdgex2()) {
            (*pos)[0] = map->getEdgex2() + map->getMapX() - 34;
        }
    }
    isAttack = false;
    r_timer = clock();
}
else if (dir == "RL" && onLadder == 0) {
    if (!map->isEdge(pos, "RL"))
    {
        if ((*pos)[0] - 250 >= 350)
        {
            (*pos)[0] -= 250;
        }
        else
        {
            (*pos)[0] -= map->scrollLeft(250);
        }
        if ((*pos)[0] - map->getMapX() < map->getEdgex1()) {
            (*pos)[0] = map->getEdgex1() + map->getMapX() - 34;
        }
    }
    isAttack = false;
    r_timer = clock();
}
if (dir == "left" && onLadder == 0)
{
    if (!map->isEdge(pos, "left"))
    {
        if ((*pos)[0] - 5 >= 350)
        {
            (*pos)[0] -= 5;
        }
        else
        {
            (*pos)[0] -= map->scrollLeft(5);
        }
    }
}

    animationWalkL.OnMove();
}
if (dir == "right" && onLadder == 0)
{
    if (!map->isEdge(pos, "right"))
    {
        if ((*pos)[0] + 5 <= 600)
        {
            (*pos)[0] += 5;
        }
        else
        {
            (*pos)[0] += map->scrollRight(5);
        }
    }
    animationWalkR.OnMove();
}
if (dir == "up")
{
    if (onLadder == 0)
    {
        if (map->isTeleport(pos))
        {
            int tmp = currentMap;
            currentMap = map->setTeleport(pos);
            isMovingUp = false;
            if (tmp != currentMap) {
                if (q_timer == 1) {
                    q_timer = clock();
                    isAttack = false;
                }
            }
            if (w_timer == 1) {

```

```

        w_timer = clock();
        isAttack = false;
        wcount = -1;
    }
}
else if (map->isLadder(pos, dir))
{
    map->setLadder(pos, dir);
    onLadder = 1;
    velocity = initial_velocity;
    animationLadder.OnShow();
    isMovingLeft = false;
    isMovingRight = false;
    isJump = false;
    Fall = 0;
}
}
else
{
    (*pos)[1] -= 5;
    isJump = false;
    animationLadder.OnMove();

    if (map->isArea(pos))
    {
        onLadder = 0;
    }
}
}
if (dir == "down")
{
    if (map->isLadder(pos, dir) && onLadder == 0)
    {
        map->setLadder(pos, dir);
        onLadder = 1;
        velocity = initial_velocity;
        animationLadder.OnShow();
        isMovingLeft = false;
        isMovingRight = false;
        isJump = false;
        Fall = 0;
    }
    else if (onLadder == 0 && !isJump && Fall == 0)
    {
        isProne = true;
        if (direction == 0)
        {
            animationProneR.OnMove();
        }
        else
        {
            animationProneL.OnMove();
        }
    }
    else if (onLadder == 1)
    {
        (*pos)[1] += 5;
        animationLadder.OnMove();

        if (map->isArea(pos))
        {
            onLadder = 0;
        }
    }
}
}
if (dir == "jump" || dir == "fall")
{
    Jump(pos, dir);
    if (direction == 0)
    {
        animationJumpR.OnMove();
    }
}
}

```

<pre> } else { animationJumpL.OnMove(); } } return true; } bool Character::GetBackpack() { return openBackpack; } }</pre>
Effect.h
<pre>#pragma once namespace game_framework { class Effect { public: Effect(string, int, int); string getName(); int getNum(); int getTime(); void setTime(int); protected: string effectname; int effecttime, effectnum; }; }</pre>
Effect.cpp
<pre>#include "stdafx.h" #include "Resource.h" #include <mmsystem.h> #include <draw.h> #include "audio.h" #include "gamelib.h" #include "Effect.h" #include <vector> #include <string> #include <sstream> #include <iomanip> namespace game_framework { Effect::Effect(string name, int time, int num) { effectname = name; effecttime = time; effectnum = num; } string Effect::getName() { return effectname; } int Effect::getNum() { return effectnum; } int Effect::getTime() { return effecttime; } void Effect::setTime(int time) { effecttime += time; } }</pre>
Item.h
<pre>#pragma once namespace game_framework { class Item {</pre>

<pre> public: Item(int, int, string); ~Item(); void LoadBitmap(); void SetTopLeft(int, int); void onShow(int, int, bool); int GetX(); int GetY(); int GetWidth(); int GetHeight(); int GetNum(); string GetName(); double GetShowTime(); void SetNum(int); void ShowNum(); protected: CMovingBitmap item_pic; vector<CMovingBitmap*> numfont; string name, effecttype; double timer_show; int x, y, num; }; </pre>	
<p style="text-align: center;">Item.cpp</p>	<pre> #include "stdafx.h" #include "Resource.h" #include <mmsystem.h> #include <ddraw.h> #include "audio.h" #include "gamelib.h" #include "Item.h" #include <vector> #include <string> #include <sstream> #include <iomanip> namespace game_framework { Item::Item(int px, int py, string s) { timer_show = clock(); x = px; y = py; name = s; num = 1; LoadBitmap(); } Item::~Item() { for (int i = 0; i < (int)numfont.size(); i++) { delete numfont[i]; } } void Item::LoadBitmap() { if (name == "紅色藥水") { item_pic.LoadBitmap(IDB_potion_red, RGB(0, 255, 0)); } else if (name == "超級紅色藥水") { item_pic.LoadBitmap(IDB_superred, RGB(0, 255, 0)); } else if (name == "藍色藥水") { item_pic.LoadBitmap(IDB_potion_blue, RGB(0, 255, 0)); } else if (name == "超級藍色藥水") { item_pic.LoadBitmap(IDB_superblue, RGB(0, 255, 0)); } } } </pre>


```

else if (name == "長劍 0")
{
    item_pic.LoadBitmap(IDB_sword_0, RGB(0, 255, 0));
}
else if (name == "長劍 1")
{
    item_pic.LoadBitmap(IDB_sword_1, RGB(0, 255, 0));
}
else if (name == "長劍 2")
{
    item_pic.LoadBitmap(IDB_sword_2, RGB(0, 255, 0));
}
else if (name == "長劍 3")
{
    item_pic.LoadBitmap(IDB_sword_3, RGB(0, 255, 0));
}
else if (name == "盾牌 0")
{
    item_pic.LoadBitmap(IDB_shield_0, RGB(0, 255, 0));
}
else if (name == "盾牌 1")
{
    item_pic.LoadBitmap(IDB_shield_1, RGB(0, 255, 0));
}
else if (name == "上衣 0")
{
    item_pic.LoadBitmap(IDB_cloth_0, RGB(0, 255, 0));
}
else if (name == "上衣 1")
{
    item_pic.LoadBitmap(IDB_cloth_1, RGB(0, 255, 0));
}
else if (name == "上衣 2")
{
    item_pic.LoadBitmap(IDB_cloth_2, RGB(0, 255, 0));
}
}
void Item::SetTopLeft(int x1, int y1)
{
    x = x1;
    y = y1;
}
void Item::onShow(int MapX, int MapY, bool floating)
{
    if (floating)
    {
        item_pic.SetTopLeft(x + MapX, y + MapY - 5);
    }
    else
    {
        item_pic.SetTopLeft(x + MapX, y + MapY);
    }

    item_pic.ShowBitmap();
}
int Item::GetX()
{
    return x;
}
int Item::GetY()
{
    return y;
}
int Item::GetWidth()
{
    return item_pic.Width();
}
int Item::GetHeight()
{
    return item_pic.Height();
}
int Item::GetNum()

```

<pre> { return num; } string Item::GetName() { return name; } double Item::GetShowTime() { return timer_show; } void Item::SetNum(int n) { num = n; } void Item::ShowNum() { string s = to_string(num); for (int i = 0; i < (int)numfont.size(); i++) { delete numfont[i]; } numfont.clear(); for (int i = 0; i < (int)s.size(); i++) { numfont.push_back(new CMovingBitmap); numfont[i]->LoadBitmap(335 + s[i] - 48, RGB(0, 255, 0)); numfont[i]->SetTopLeft(x + 30 - (s.length() - i)*numfont[i]->Width(), y + 30 - numfont[i]->Height()); numfont[i]->ShowBitmap(); } } } </pre>	
	<div>Map.h</div> <pre> #pragma once #include "Monster.h" #include "Item.h" namespace game_framework { class Map { public: ~Map(); void Initialize(int, int); int getInitX(); int getInitY(); int getMapX(); int getMapY(); int getEdgex1(); int getEdgex2(); int scrollRight(int); int scrollLeft(int); void onShow(); void itemShow(); void monsterMove(); void monsterShow(int); int hitMonster(int, int, int, int, int, int, int, bool, int*); int Monsterhit(int, int, int, int, int); int getMonsterHP(); void PickItem(int, int, int, int, vector<Item*>); void setArea(vector<int*>); void setLadder(vector<int*>, string); int setTeleport(vector<int*>); bool isEdge(vector<int*>, string); bool isArea(vector<int*>); bool isLadder(vector<int*>, string); bool isTeleport(vector<int*>); protected: int MapX, MapY, currentMap, edgex1, edgex2; vector<vector<int*>> Area; vector<vector<int*>> Ladder; } } </pre>

<pre> vector<vector<int>> Teleport; vector<Monster*> monster; vector<Item*> item; CMovingBitmap *background = NULL; bool floating, music; double item_floating; }; } </pre>	
	Map.cpp
<pre> #include "stdafx.h" #include "Resource.h" #include <mmsystem.h> #include <draw.h> #include "audio.h" #include "gamelib.h" #include "Map.h" #include <vector> #include <string> #include <sstream> #include <iomanip> namespace game_framework { Map::~Map() { for (int i = 0; i < (int)monster.size(); i++) { delete monster[i]; } for (int i = 0; i < (int)item.size(); i++) { delete item[i]; } delete background; } void Map::Initialize(int map, int tmp) { vector<int> Areaxy; vector<int> Ladderxy; vector<int> Teleportxy; int random; currentMap = map; Area.clear(); Ladder.clear(); Teleport.clear(); music = floating = false; item_floating = -1; for (int i = 0; i < int(monster.size()); i++) { delete monster[i]; } monster.clear(); if (background != NULL) { delete background; } for (int i = 0; i < int(item.size()); i++) { delete item[i]; } item.clear(); if (currentMap == 0) { Areaxy.clear(); Areaxy.push_back(80); Areaxy.push_back(440); Areaxy.push_back(715); Areaxy.push_back(440); Area.push_back(Areaxy); Teleportxy.clear(); </pre>	

```

Teleportxy.push_back(650);
Teleportxy.push_back(440);
Teleportxy.push_back(715);
Teleportxy.push_back(440);
Teleportxy.push_back(1);
Teleport.push_back(Teleportxy);
background = new CMovingBitmap;
background->LoadBitmap(IDB_map01);
edgex1 = Area[0][0];
edgex2 = Area[0][2];
}
else if (currentMap == 1)
{
Areaxy.clear();
Areaxy.push_back(0);
Areaxy.push_back(610);
Areaxy.push_back(1985);
Areaxy.push_back(610);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(0);
Areaxy.push_back(313);
Areaxy.push_back(610);
Areaxy.push_back(313);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(630);
Areaxy.push_back(363);
Areaxy.push_back(1130);
Areaxy.push_back(363);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(480);
Areaxy.push_back(116);
Areaxy.push_back(1057);
Areaxy.push_back(116);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(1075);
Areaxy.push_back(165);
Areaxy.push_back(1796);
Areaxy.push_back(165);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(1667);
Areaxy.push_back(313);
Areaxy.push_back(1985);
Areaxy.push_back(313);
Area.push_back(Areaxy);
Ladderxy.clear();
Ladderxy.push_back(260);
Ladderxy.push_back(313);
Ladderxy.push_back(292);
Ladderxy.push_back(495);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(575);
Ladderxy.push_back(116);
Ladderxy.push_back(607);
Ladderxy.push_back(220);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(994);
Ladderxy.push_back(116);
Ladderxy.push_back(1026);
Ladderxy.push_back(250);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(769);
Ladderxy.push_back(363);
Ladderxy.push_back(801);
Ladderxy.push_back(495);

```

```

Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(1684);
Ladderxy.push_back(165);
Ladderxy.push_back(1716);
Ladderxy.push_back(222);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(1772);
Ladderxy.push_back(313);
Ladderxy.push_back(1804);
Ladderxy.push_back(495);
Ladder.push_back(Ladderxy);
Teleportxy.clear();
Teleportxy.push_back(25);
Teleportxy.push_back(610);
Teleportxy.push_back(95);
Teleportxy.push_back(610);
Teleportxy.push_back(-1);
Teleport.push_back(Teleportxy);
Teleportxy.clear();
Teleportxy.push_back(1895);
Teleportxy.push_back(610);
Teleportxy.push_back(1965);
Teleportxy.push_back(610);
Teleportxy.push_back(1);
Teleport.push_back(Teleportxy);
background = new CMovingBitmap;
background->LoadBitmap(IDB_map11);
edgex1 = Area[0][0];
edgex2 = Area[0][2];
for (int i = 0; i < int(Area.size()); i++)
{
    srand((unsigned)time(NULL));
    for (int j = Area[i][0]; j < Area[i][2]; j += random)
    {
        random = rand() % 2;
        monster.push_back(new Monster(j, Area[i][1], random, Area[i], currentMap, 10, 150, 25, 5));
        random = rand() % 50 + 70;
    }
}
}
else if (currentMap == 2)
{
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(628);
    Areaxy.push_back(1500);
    Areaxy.push_back(628);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(32);
    Areaxy.push_back(363);
    Areaxy.push_back(412);
    Areaxy.push_back(363);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(428);
    Areaxy.push_back(408);
    Areaxy.push_back(940);
    Areaxy.push_back(408);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(1023);
    Areaxy.push_back(408);
    Areaxy.push_back(1500);
    Areaxy.push_back(408);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(33);
    Areaxy.push_back(145);
    Areaxy.push_back(413);

```

```
Areaxy.push_back(145);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(429);
Areaxy.push_back(189);
Areaxy.push_back(940);
Areaxy.push_back(189);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(956);
Areaxy.push_back(144);
Areaxy.push_back(1335);
Areaxy.push_back(144);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(1285);
Areaxy.push_back(57);
Areaxy.push_back(1467);
Areaxy.push_back(57);
Area.push_back(Areaxy);
Ladderxy.clear();
Ladderxy.push_back(299);
Ladderxy.push_back(363);
Ladderxy.push_back(327);
Ladderxy.push_back(525);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(602);
Ladderxy.push_back(408);
Ladderxy.push_back(630);
Ladderxy.push_back(525);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(1141);
Ladderxy.push_back(408);
Ladderxy.push_back(1169);
Ladderxy.push_back(525);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(225);
Ladderxy.push_back(145);
Ladderxy.push_back(253);
Ladderxy.push_back(264);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(743);
Ladderxy.push_back(189);
Ladderxy.push_back(771);
Ladderxy.push_back(307);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(1244);
Ladderxy.push_back(144);
Ladderxy.push_back(1272);
Ladderxy.push_back(319);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(1337);
Ladderxy.push_back(57);
Ladderxy.push_back(1365);
Ladderxy.push_back(93);
Ladder.push_back(Ladderxy);
Teleportxy.clear();
Teleportxy.push_back(25);
Teleportxy.push_back(628);
Teleportxy.push_back(95);
Teleportxy.push_back(628);
Teleportxy.push_back(-1);
Teleport.push_back(Teleportxy);
Teleportxy.clear();
Teleportxy.push_back(1420);
Teleportxy.push_back(628);
```

```

Teleportxy.push_back(1490);
Teleportxy.push_back(628);
Teleportxy.push_back(1);
Teleport.push_back(Teleportxy);
background = new CMovingBitmap;
background->LoadBitmap(IDB_map12);
edgex1 = Area[0][0];
edgex2 = Area[0][2];
for (int i = 0; i < int(Area.size()); i++)
{
    srand((unsigned)time(NULL));
    for (int j = Area[i][0]; j < Area[i][2]; j += random)
    {
        random = rand() % 2;
        monster.push_back(new Monster(j, Area[i][1], random, Area[i], currentMap, 25, 300, 40, 15));
        random = rand() % 50 + 70;
    }
}
}
else if (currentMap == 3)
{
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(508);
    Areaxy.push_back(1420);
    Areaxy.push_back(508);
    Area.push_back(Areaxy);
    Teleportxy.clear();
    Teleportxy.push_back(20);
    Teleportxy.push_back(508);
    Teleportxy.push_back(95);
    Teleportxy.push_back(508);
    Teleportxy.push_back(-1);
    Teleport.push_back(Teleportxy);
    Teleportxy.clear();
    Teleportxy.push_back(1345);
    Teleportxy.push_back(508);
    Teleportxy.push_back(1420);
    Teleportxy.push_back(508);
    Teleportxy.push_back(1);
    Teleport.push_back(Teleportxy);
    background = new CMovingBitmap;
    background->LoadBitmap(IDB_map14);
    edgex1 = Area[0][0];
    edgex2 = Area[0][2];
    monster.push_back(new Monster(750, Area[0][1], 0, Area[0], currentMap, 300, 500, 150, 100));
}
else if (currentMap == 4)
{
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(595);
    Areaxy.push_back(1805);
    Areaxy.push_back(595);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(240);
    Areaxy.push_back(355);
    Areaxy.push_back(655);
    Areaxy.push_back(355);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(690);
    Areaxy.push_back(295);
    Areaxy.push_back(1195);
    Areaxy.push_back(295);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(175);
    Areaxy.push_back(295);
    Areaxy.push_back(175);
}

```

```

Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(330);
Areaxy.push_back(115);
Areaxy.push_back(835);
Areaxy.push_back(115);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(1230);
Areaxy.push_back(355);
Areaxy.push_back(1555);
Areaxy.push_back(355);
Area.push_back(Areaxy);
Ladderxy.clear();
Ladderxy.push_back(435);
Ladderxy.push_back(355);
Ladderxy.push_back(480);
Ladderxy.push_back(515);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(240);
Ladderxy.push_back(175);
Ladderxy.push_back(285);
Ladderxy.push_back(275);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(705);
Ladderxy.push_back(115);
Ladderxy.push_back(750);
Ladderxy.push_back(215);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(1025);
Ladderxy.push_back(295);
Ladderxy.push_back(1070);
Ladderxy.push_back(500);
Ladder.push_back(Ladderxy);
Teleportxy.clear();
Teleportxy.push_back(25);
Teleportxy.push_back(595);
Teleportxy.push_back(100);
Teleportxy.push_back(595);
Teleportxy.push_back(-1);
Teleport.push_back(Teleportxy);
Teleportxy.clear();
Teleportxy.push_back(1715);
Teleportxy.push_back(595);
Teleportxy.push_back(1790);
Teleportxy.push_back(595);
Teleportxy.push_back(1);
Teleport.push_back(Teleportxy);
background = new CMovingBitmap;
background->LoadBitmap(IDB_map23);
edgex1 = Area[0][0];
edgex2 = Area[0][2];
for (int i = 0; i < int(Area.size()); i++)
{
    srand((unsigned)time(NULL));
    for (int j = Area[i][0]; j < Area[i][2]; j += random)
    {
        random = rand() % 2;
        monster.push_back(new Monster(j, Area[i][1], random, Area[i], currentMap, 200, 750, 100, 30));
        random = rand() % 80 + 90;
    }
}
}
else if (currentMap == 5)
{
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(622);
    Areaxy.push_back(1199);

```



```

Areaxy.push_back(622);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(0);
Areaxy.push_back(385);
Areaxy.push_back(1119);
Areaxy.push_back(385);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(0);
Areaxy.push_back(145);
Areaxy.push_back(849);
Areaxy.push_back(145);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(885);
Areaxy.push_back(145);
Areaxy.push_back(939);
Areaxy.push_back(145);
Area.push_back(Areaxy);
Areaxy.clear();
Areaxy.push_back(975);
Areaxy.push_back(145);
Areaxy.push_back(1029);
Areaxy.push_back(145);
Area.push_back(Areaxy);
Ladderxy.clear();
Ladderxy.push_back(378);
Ladderxy.push_back(385);
Ladderxy.push_back(410);
Ladderxy.push_back(515);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(800);
Ladderxy.push_back(385);
Ladderxy.push_back(832);
Ladderxy.push_back(515);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(261);
Ladderxy.push_back(145);
Ladderxy.push_back(293);
Ladderxy.push_back(272);
Ladder.push_back(Ladderxy);
Ladderxy.clear();
Ladderxy.push_back(581);
Ladderxy.push_back(145);
Ladderxy.push_back(613);
Ladderxy.push_back(272);
Ladder.push_back(Ladderxy);
Teleportxy.clear();
Teleportxy.push_back(15);
Teleportxy.push_back(145);
Teleportxy.push_back(86);
Teleportxy.push_back(145);
Teleportxy.push_back(-1);
Teleport.push_back(Teleportxy);
Teleportxy.clear();
Teleportxy.push_back(1127);
Teleportxy.push_back(622);
Teleportxy.push_back(1198);
Teleportxy.push_back(622);
Teleportxy.push_back(1);
Teleport.push_back(Teleportxy);
background = new CMovingBitmap;
background->LoadBitmap(IDB_map22);
edgex1 = Area[0][0];
edgex2 = Area[0][2];
for (int i = 0; i < int(Area.size()); i++)
{
    srand((unsigned)time(NULL));
    for (int j = Area[i][0]; j < Area[i][2]; j += random)

```

```

        {
            random = rand() % 2;
            monster.push_back(new Monster(j, Area[i][1], random, Area[i], currentMap, 300, 900, 125, 50));
            random = rand() % 80 + 90;
        }
    }
}
else if (currentMap == 6)
{
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(660);
    Areaxy.push_back(1190);
    Areaxy.push_back(660);
    Area.push_back(Areaxy);
    Teleportxy.clear();
    Teleportxy.push_back(10);
    Teleportxy.push_back(660);
    Teleportxy.push_back(85);
    Teleportxy.push_back(660);
    Teleportxy.push_back(-1);
    Teleport.push_back(Teleportxy);
    Teleportxy.clear();
    Teleportxy.push_back(1113);
    Teleportxy.push_back(660);
    Teleportxy.push_back(1188);
    Teleportxy.push_back(660);
    Teleportxy.push_back(1);
    Teleport.push_back(Teleportxy);
    background = new CMovingBitmap;
    background->LoadBitmap(IDB_map24);
    edgex1 = Area[0][0];
    edgex2 = Area[0][2];
    monster.push_back(new Monster((Area[0][3] - Area[0][0]) / 2, Area[0][1], 1, Area[0], currentMap, 5000, 2000,
600, 100));
}
else if (currentMap == 7)
{
    Areaxy.clear();
    Areaxy.push_back(0);
    Areaxy.push_back(625);
    Areaxy.push_back(1326);
    Areaxy.push_back(626);
    Area.push_back(Areaxy);
    Areaxy.clear();
    Areaxy.push_back(390);
    Areaxy.push_back(565);
    Areaxy.push_back(1020);
    Areaxy.push_back(565);
    Area.push_back(Areaxy);
    Teleportxy.clear();
    Teleportxy.push_back(15);
    Teleportxy.push_back(625);
    Teleportxy.push_back(80);
    Teleportxy.push_back(625);
    Teleportxy.push_back(-1);
    Teleport.push_back(Teleportxy);
    background = new CMovingBitmap;
    background->LoadBitmap(IDB_map31);
    edgex1 = Area[0][0];
    edgex2 = Area[0][2];
    monster.push_back(new Monster(528, 525, 1, Area[0], currentMap, 10000, 5000, 0, 1000));
}
if (background->Width() < 1024)
{
    MapX = (1024 - background->Width()) / 2;
}
else if (tmp > 0)
{
    MapX = 0;
}
else

```

```

    {
        MapX = 1024 - background->Width();
    }

    if (background->Height() < 768)
    {
        MapY = (768 - background->Height()) / 2;
    }
    else
    {
        MapY = 768 - background->Height();
    }
    background->SetTopLeft(MapX, MapY);
}
int Map::getInitX()
{
    return 350;
}
int Map::getInitY()
{
    return Area[0][1];
}
int Map::getMapX()
{
    return MapX;
}
int Map::getMapY()
{
    return MapY;
}
int Map::getEdgex1()
{
    return edgex1;
}
int Map::getEdgex2()
{
    return edgex2;
}
int Map::scrollRight(int step)
{
    if (background->Width() <= 1024) {
        return step;
    }
    else if (background->Width() + MapX - step >= 1024)
    {
        MapX -= step;
        background->SetTopLeft(MapX, MapY);
        background->ShowBitmap();
        step = 0;
    }
    else if (background->Width() + MapX > 1024)
    {
        step -= ((background->Width() + MapX) - 1024);
        MapX = 1024 - background->Width();
        background->SetTopLeft(MapX, MapY);
        background->ShowBitmap();
    }
    return step;
}
int Map::scrollLeft(int step)
{
    if (background->Width() <= 1024) {
        return step;
    }
    else if (MapX + step <= 0)
    {
        MapX += step;
        background->SetTopLeft(MapX, MapY);
        background->ShowBitmap();
        step = 0;
    }
    else

```

```

    {
        step += MapX;
        MapX = 0;
        background->SetTopLeft(MapX, MapY);
        background->ShowBitmap();
    }
    return step;
}
void Map::onShow()
{
    if (!music) {
        for (int i = 0; i < 18; i++) {
            CAudio::Instance()->Stop(i);
        }
        CAudio::Instance()->Play(currentMap + 1, true);
        music = true;
    }
    background->ShowBitmap();
}
void Map::itemShow()
{
    int del = -1;
    if (clock() - item_floating >= 500)
    {
        floating = !floating;
        item_floating = clock();
    }
    for (int i = 0; i < (int)item.size(); i++)
    {
        if (clock() - item[i]->GetShowTime() >= 10000)
        {
            del = i;
        }
    }
    if (del != -1)
    {
        for (int i = 0; i < del; i++)
        {
            delete item[i];
        }
        item.erase(item.begin(), item.begin() + del);
    }
    for (int i = 0; i < (int)item.size(); i++)
    {
        item[i]->onShow(MapX, MapY, floating);
    }
}
void Map::monsterMove()
{
    for (int i = 0; i < int(monster.size()); i++)
    {
        monster[i]->onMove();
    }
}
void Map::monsterShow(int px)
{
    srand((unsigned)time(NULL));

    for (int i = 0; i < int(monster.size()); i++)
    {
        monster[i]->onShow(MapX, MapY, rand() % 5, px);
    }
}
int Map::hitMonster(int cx, int cy, int skillx1, int skilly1, int skillx2, int skilly2, int damage, bool finalbitmap, int*
totaldamage)
{
    int expr = 0;

    for (int i = 0; i < int(monster.size()); i++)
    {
        expr += monster[i]->getHit(cx, cy, skillx1, skilly1, skillx2, skilly2, damage, finalbitmap, totaldamage, &item);
    }
}

```

```

        return expr;
    }
}
int Map::Monsterhit(int x1, int y1, int x2, int y2, int def)
{
    int damage = 0;
    for (int i = 0; i < int(monster.size()); i++)
    {
        damage += monster[i]->Hit(x1, y1, x2, y2, def);
    }
    return damage;
}
int Map::getMonsterHP()
{
    return monster[0]->gethp();
}
void Map::PickItem(int x, int y, int w, int h, vector<Item*>* backpack)
{
    vector<int> del;
    for (int i = 0; i < (int)item.size(); i++)
    {
        if (item[i]->GetX() + item[i]->GetWidth() / 2 >= x && item[i]->GetX() + item[i]->GetWidth() / 2 <= x + w &&
item[i]->GetY() + item[i]->GetHeight() / 2 >= y && item[i]->GetY() + item[i]->GetHeight() / 2 <= y + h)
        {
            CAudio::Instance()->Play(13);
            backpack->push_back(item[i]);
            del.push_back(i);
        }
    }
    for (int i = (int)del.size() - 1; i >= 0; i--)
    {
        item.erase(item.begin() + del[i]);
    }
}
void Map::setArea(vector<int*>* pos)
{
    double y = 0;
    for (int i = 0; i < int(Area.size()); i++)
    {
        if ((*pos)[0] - MapX + 34 >= Area[i][0] && (*pos)[0] - MapX + 34 <= Area[i][2])
        {
            if (Area[i][1] != Area[i][3])
            {
                y = double((*pos)[0] - MapX + 34 - Area[i][0]) * double(Area[i][3] - Area[i][1]) / double(Area[i][2] -
Area[i][0]);
                y = (int)y + Area[i][1];
            }
            else
            {
                y = Area[i][1];
            }
            if ((abs((*pos)[1] - MapY - (y - 70)) <= 15))
            {
                (*pos)[1] = (int)y + MapY - 70;
            }
        }
    }
}
void Map::setLadder(vector<int*>* pos, string position)
{
    if (position == "up")
    {
        for (int i = 0; i < int(Ladder.size()); i++)
        {
            if ((*pos)[0] - MapX + 34 >= Ladder[i][0] && (*pos)[0] - MapX + 34 <= Ladder[i][2] && (*pos)[1] -
MapY >= Ladder[i][1] && (*pos)[1] - MapY <= Ladder[i][3])
            {
                (*pos)[0] = Ladder[i][0] + MapX - (62 - (Ladder[i][2] - Ladder[i][0])) / 2 - 7;
            }
        }
    }
    else
    {

```

```

        for (int i = 0; i < int(Ladder.size()); i++)
        {
            if ((*pos)[0] - MapX + 34 >= Ladder[i][0] && (*pos)[0] - MapX + 34 <= Ladder[i][2] && (*pos)[1] -
MapY + 70 >= Ladder[i][1] && (*pos)[1] - MapY + 70 <= Ladder[i][3])
            {
                (*pos)[0] = Ladder[i][0] + MapX - (62 - (Ladder[i][2] - Ladder[i][0])) / 2 - 7;
                (*pos)[1] += 15;
            }
        }
    }
}

int Map::setTeleport(vector<int>* pos)
{
    int tmp;
    if (currentMap == 3 || currentMap == 6) {
        if (monster[0]->gethp() > 0) {
            return currentMap;
        }
    }
    for (int i = 0; i < int(Teleport.size()); i++)
    {
        if ((*pos)[0] - MapX + 34 >= Teleport[i][0] && (*pos)[0] - MapX + 34 <= Teleport[i][2] && (abs((*pos)[1] -
MapY - (Teleport[i][1] - 70)) <= 10))
        {

            tmp = Teleport[i][4];
            CAudio::Instance()->Stop(currentMap);
            currentMap += tmp;
            Initialize(currentMap, tmp);

            if (tmp > 0 || currentMap == 0)
            {
                (*pos)[0] = Teleport[0][0] + MapX;
                (*pos)[1] = Teleport[0][1] + MapY - 70;
            }
            else
            {
                (*pos)[0] = Teleport[(int)Teleport.size() - 1][0] + MapX;
                (*pos)[1] = Teleport[(int)Teleport.size() - 1][1] + MapY - 70;
            }
            break;
        }
    }
    return currentMap;
}

bool Map::isEdge(vector<int>* pos, string direction)
{
    if ((*pos)[0] - MapX + 34 <= edgex1 && direction == "left")
    {
        (*pos)[0] = edgex1 + MapX - 34;
        return true;
    }
    else if ((*pos)[0] - MapX + 34 >= edgex2 && direction == "right")
    {
        (*pos)[0] = edgex2 + MapX - 34;
        return true;
    }
    return false;
}

bool Map::isArea(vector<int>* pos)
{
    double y = 0;
    for (int i = 0; i < int(Area.size()); i++)
    {
        if ((*pos)[0] - MapX + 34 >= Area[i][0] && (*pos)[0] - MapX + 34 <= Area[i][2])
        {
            if (Area[i][1] != Area[i][3])
            {
                y = double((*pos)[0] - MapX + 34 - Area[i][0]) * double(Area[i][3] - Area[i][1]) / double(Area[i][2] -
Area[i][0]);
                y = (int)y + Area[i][1];
            }
        }
    }
}

```

```

        else
        {
            y = Area[i][1];
        }
        if ((abs((*pos)[1] - MapY - (y - 70)) <= 15))
        {
            setArea(pos);
            return true;
        }
    }
}
return false;
}
bool Map::isLadder(vector<int>* pos, string position)
{
    if (position == "up")
    {
        for (int i = 0; i < int(Ladder.size()); i++)
        {
            if ((*pos)[0] - MapX + 34 >= Ladder[i][0] && (*pos)[0] - MapX + 34 <= Ladder[i][2] && (*pos)[1] -
MapY >= Ladder[i][1] && (*pos)[1] - MapY <= Ladder[i][3])
            {
                return true;
            }
        }
    }
    else
    {
        for (int i = 0; i < int(Ladder.size()); i++)
        {
            if ((*pos)[0] - MapX + 34 >= Ladder[i][0] && (*pos)[0] - MapX + 34 <= Ladder[i][2] && (*pos)[1] -
MapY + 70 >= Ladder[i][1] && (*pos)[1] - MapY + 70 <= Ladder[i][3])
            {
                return true;
            }
        }
    }
    return false;
}
bool Map::isTeleport(vector<int>* pos)
{
    for (int i = 0; i < int(Teleport.size()); i++)
    {
        if ((*pos)[0] - MapX + 34 >= Teleport[i][0] && (*pos)[0] - MapX + 34 <= Teleport[i][2] && (abs((*pos)[1] -
MapY - (Teleport[i][1] - 70)) <= 10))
        {
            return true;
        }
    }
    return false;
}
}

```

Monster.h

```

#pragma once
#include "Item.h"
namespace game_framework
{
    class Monster
    {
    public:
        Monster(int, int, int, vector<int>, int, int, int, int, int);
        ~Monster();
        void LoadBitmap();
        void onShow(int, int, int, int);
        void onMove();
        int gethp();
        int getHit(int, int, int, int, int, int, int, bool, int*, vector<Item*>);
        int Hit(int, int, int, int, int);
    protected:
        CAnimation animationStandR, animationStandL, animationWalkL, animationWalkR, animationHitL,
animationHitR, animationDieL, animationDieR, animationSkillL, animationSkillR;
    }
}

```

<pre> vector<int> Area, stonex, stoney; vector<CMovingBitmap*> stone; int direction, state, MapX, MapY, x, y, mh, mw, hp, currentMap, expr, dmg, def, rnd; double timer_death, timer_gethit, timer_hit, timer_skill; bool skillhit, soundeffect; }; } </pre>	
	Monster.cpp
<pre> #include "stdafx.h" #include "Resource.h" #include <mmsystem.h> #include <ddraw.h> #include "audio.h" #include "gamelib.h" #include "Monster.h" #include <vector> #include <string> #include <sstream> #include <iomanip> namespace game_framework { Monster::Monster(int MapX, int MapY, int dir, vector<int> area, int map, int mexp, int hpnum, int dmgnum, int defnum) { expr = mexp; dmg = dmgnum; def = defnum; hp = hpnum; timer_death = timer_gethit = timer_hit = timer_skill = -1; currentMap = map; Area = area; LoadBitmap(); x = MapX; y = MapY - animationStandR.Height(); direction = dir; stone.clear(); soundeffect = false; mh = animationStandR.Height() / 2; mw = animationStandR.Width() / 2; if (currentMap == 6) { timer_skill = clock(); rnd = (rand() % 4 + 1) * 1000; } else if (currentMap == 7) { timer_skill = clock(); rnd = 3000; } } Monster::~Monster() { for (int i = 0; i < (int)stone.size(); i++) { delete stone[i]; } } void Monster::LoadBitmap() { if (currentMap == 1) { animationStandR.AddBitmap(IDB_11STANDRIGHT_0, RGB(0, 255, 0)); animationStandL.AddBitmap(IDB_11STANDLEFT_0, RGB(0, 255, 0)); animationWalkR.AddBitmap(IDB_11MOVERIGHT_0, RGB(0, 255, 0)); animationWalkR.AddBitmap(IDB_11MOVERIGHT_1, RGB(0, 255, 0)); animationWalkR.AddBitmap(IDB_11MOVERIGHT_2, RGB(0, 255, 0)); animationWalkR.AddBitmap(IDB_11MOVERIGHT_3, RGB(0, 255, 0)); animationWalkR.AddBitmap(IDB_11MOVERIGHT_4, RGB(0, 255, 0)); animationWalkL.AddBitmap(IDB_11MOVELEFT_0, RGB(0, 255, 0)); animationWalkL.AddBitmap(IDB_11MOVELEFT_1, RGB(0, 255, 0)); animationWalkL.AddBitmap(IDB_11MOVELEFT_2, RGB(0, 255, 0)); animationWalkL.AddBitmap(IDB_11MOVELEFT_3, RGB(0, 255, 0)); animationWalkL.AddBitmap(IDB_11MOVELEFT_4, RGB(0, 255, 0)); </pre>	

[illegible]

[illegible]

[illegible]

```

        animationDieR.AddBitmap(IDB_24dieright_10, RGB(0, 255, 0));
        animationDieL.SetDelayCount(5);
        animationDieR.SetDelayCount(5);
        animationDieL.Reset();
        animationDieR.Reset();
    }
    else if (currentMap == 7)
    {
        animationStandR.AddBitmap(IDB_31standleft_0, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_0, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_1, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_2, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_3, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_4, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_5, RGB(0, 255, 0));
        animationStandL.AddBitmap(IDB_31standleft_6, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_0, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_1, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_2, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_3, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_4, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_5, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_6, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_7, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_8, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_9, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_10, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_11, RGB(0, 255, 0));
        animationSkillL.AddBitmap(IDB_31skillleft_12, RGB(0, 255, 0));
        animationDieL.AddBitmap(IDB_31dieleft_0, RGB(0, 255, 0));
        animationHitL.AddBitmap(IDB_31dieleft_0, RGB(0, 255, 0));
        animationStandL.SetDelayCount(5);
        animationSkillL.SetDelayCount(5);
        animationStandL.Reset();
        animationSkillL.Reset();
    }
}

void Monster::onShow(int Map_X, int Map_Y, int random, int px)
{
    MapX = Map_X;
    MapY = Map_Y;

    if (clock() - timer_skill == rnd && currentMap == 6)
    {
        skillhit = false;
    }
    if (currentMap == 6 && ((direction == 0 && x + MapX > px) || (direction == 1 && x + MapX < px)))
    {
        soundeffect = false;
        timer_skill = clock();
        rnd = (rand() % 4 + 1) * 1000;
    }
    if ((px < x + MapX && currentMap == 6 && hp > 0) || currentMap == 7)
    {
        direction = 1;
    }
    else if (px > x + MapX && currentMap == 6 && hp > 0)
    {
        direction = 0;
    }
    if (clock() - timer_skill >= rnd && currentMap == 6)
    {
        state = 1;
    }
    else if ((clock() - timer_skill < rnd && currentMap == 6) || currentMap == 7)
    {
        state = 0;
    }
    else
    {
        state = random;
    }
}

```

```

        if ((clock() - timer_skill) / CLOCKS_PER_SEC > 15 && timer_skill != -1 && currentMap == 3)
        {
            timer_skill = -1;
            def /= 2;
        }
        if (hp <= 0)
        {
            if (direction == 0)
            {
                if (!animationDieR.IsFinalBitmap())
                {
                    animationDieR.SetTopLeft(x + MapX + mw, y + MapY + mh);
                    animationDieR.OnShow();
                }
                else if (int(timer_death) == -1)
                {
                    timer_death = clock();
                }
                else if ((clock() - timer_death) / CLOCKS_PER_SEC >= 3 && (currentMap == 1 || currentMap == 2 ||
currentMap == 4 || currentMap == 5))
                {
                    srand(x);
                    direction = rand() % 2;
                    state = rand() % 2;
                    x = rand() % (Area[2] - Area[0] - (animationStandR.Width() / 2)) + Area[0];
                    timer_death = -1;
                    hp = 500;
                    animationDieR.Reset();
                }
            }
            else
            {
                if (!animationDieL.IsFinalBitmap())
                {
                    animationDieL.SetTopLeft(x + MapX + mw, y + MapY + mh);
                    animationDieL.OnShow();
                }
                else if (int(timer_death) == -1)
                {
                    timer_death = clock();
                }
                else if ((clock() - timer_death) / CLOCKS_PER_SEC >= 3 && (currentMap == 1 || currentMap == 2 ||
currentMap == 4 || currentMap == 5))
                {
                    srand(x);
                    direction = rand() % 2;
                    state = rand() % 2;
                    x = rand() % (Area[2] - Area[0] - (animationStandR.Width() / 2)) + Area[0];
                    timer_death = -1;
                    hp = 500;
                    animationDieL.Reset();
                }
            }
        }
        else
        {
            if (direction == 0)
            {
                if (timer_gethit != -1 && clock() - timer_gethit <= 350)
                {
                    animationHitR.SetTopLeft(x + MapX + mw, y + MapY + mh);
                    animationHitR.OnShow();
                }
                else
                {
                    timer_gethit = -1;

                    if (state == 0)
                    {
                        if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
3)
                        {

```

```

        animationSkillR.SetTopLeft(x + MapX + mw, y + MapY + mh);
        animationSkillR.OnShow();
    }
    else
    {
        animationStandR.SetTopLeft(x + MapX + mw, y + MapY + mh);
        animationStandR.OnShow();
    }
}
else if (currentMap != 7)
{
    if (!soundeffect && clock() - timer_skill >= double(rnd - 4) && currentMap == 6) {
        CAudio::Instance()->Play(15);
        soundeffect = true;
    }
    if (clock() - timer_skill >= rnd && currentMap == 6)
    {
        if (x + (animationStandR.Width() / 2) + 10 < Area[2])
        {
            x += 10;
        }
        else
        {
            direction = 1;
            x -= 10;
        }
    }
    else if (x + (animationStandR.Width() / 2) + 1 < Area[2])
    {
        x += 1;
    }
    else
    {
        direction = 1;
        x -= 1;
    }
}

if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
3)
{
    animationSkillR.SetTopLeft(x + MapX + mw, y + MapY + mh);
    animationSkillR.OnShow();
}
else
{
    animationWalkR.SetTopLeft(x + MapX + mw, y + MapY + mh);
    animationWalkR.OnShow();
}
}
}
else
{
    if (timer_gethit != -1 && clock() - timer_gethit <= 350)
    {
        animationHitL.SetTopLeft(x + MapX + mw, y + MapY + mh);
        animationHitL.OnShow();
    }
    else
    {
        timer_gethit = -1;
    }

    if (state == 0)
    {
        if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
3)
        {
            animationSkillL.SetTopLeft(x + MapX + mw, y + MapY + mh);
            animationSkillL.OnShow();
        }
        else if (!soundeffect && clock() - timer_skill >= double(rnd - 3) && currentMap == 7) {
            CAudio::Instance()->Play(16);

```

```

        soundeffect = true;
    }
    else if (clock() - timer_skill >= rnd && currentMap == 7)
    {

        if (stone.empty())
        {
            animationSkillL.SetTopLeft(x + MapX + mw, y + MapY + mh);
            animationSkillL.OnShow();
        }
        else
        {
            animationStandL.SetTopLeft(x + MapX + mw, y + MapY + mh);
            animationStandL.OnShow();

            if (stone[0]->Top() + stone[0]->Height() >= Area[1])
            {
                for (int i = 0; i < (int)stone.size(); i++)
                {
                    delete stone[i];
                }

                stone.clear();
                stonex.clear();
                stoney.clear();
                timer_skill = clock();
                soundeffect = false;
            }
            else
            {
                for (int i = 0; i < (int)stone.size(); i++)
                {
                    stoney[i] += 20;
                    stone[i]->SetTopLeft(stonex[i] + MapX, stoney[i] + MapY);
                    stone[i]->ShowBitmap();
                }
            }
        }
    }
    else
    {
        animationStandL.SetTopLeft(x + MapX + mw, y + MapY + mh);
        animationStandL.OnShow();
    }
}
else if (currentMap != 7)
{
    if (!soundeffect && clock() - timer_skill >= double(rnd - 4) && currentMap == 6) {
        CAudio::Instance()->Play(15);
        soundeffect = true;
    }
    if (clock() - timer_skill >= rnd && currentMap == 6)
    {
        if (x + (animationStandR.Width() / 2) - 10 > Area[0])
        {
            x -= 10;
        }
        else
        {
            direction = 0;
            x += 10;
        }
    }
    else if (x + (animationStandR.Width() / 2) - 1 > Area[0])
    {
        x -= 1;
    }
    else
    {
        direction = 0;
        x += 1;
    }
}

```

```

3)
        if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
            {
                animationSkillL.SetTopLeft(x + MapX + mw, y + MapY + mh);
                animationSkillL.OnShow();
            }
            else
            {
                animationWalkL.SetTopLeft(x + MapX + mw, y + MapY + mh);
                animationWalkL.OnShow();
            }
        }
    }
}
void Monster::onMove()
{
    if (hp <= 0)
    {
        if (direction == 0)
        {
            if (!animationDieR.IsFinalBitmap())
            {
                animationDieR.OnMove();
            }
        }
        else
        {
            if (!animationDieL.IsFinalBitmap())
            {
                animationDieL.OnMove();
            }
        }
    }
}
else
{
    if (direction == 0)
    {
        if (timer_gethit != -1 && clock() - timer_gethit <= 350)
        {
            animationHitR.OnMove();
        }
        else
        {
            timer_gethit = -1;
        }
        if (state == 0)
        {
            if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
3)
                {
                    animationSkillR.OnMove();
                }
                else
                {
                    animationStandR.OnMove();
                }
            }
            else
            {
3)
                if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
                    {
                        animationSkillR.OnMove();
                    }
                    else
                    {
                        animationWalkR.OnMove();
                    }
                }
            }
        }
    }
}

```



```

    }
    else
    {
        if (timer_gethit != -1 && clock() - timer_gethit <= 350)
        {
            animationHitL.OnMove();
        }
        else
        {
            timer_gethit = -1;

            if (state == 0)
            {
                if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
3)
                {
                    animationSkillL.OnMove();
                }
                else if (clock() - timer_skill >= rnd && currentMap == 7)
                {
                    if (stone.empty())
                    {
                        animationSkillL.OnMove();

                        if (animationSkillL.IsFinalBitmap())
                        {
                            srand((unsigned)time(NULL));

                            for (int i = rand() % 89; i <= Area[2]; i += 85)
                            {
                                stone.push_back(new CMovingBitmap);
                                stonex.push_back(i);
                                stoney.push_back(0);
                                stone[(int)stone.size() - 1]->LoadBitmap(IDB_stone, RGB(0, 255, 0));
                                stone[(int)stone.size() - 1]->SetTopLeft(stonex[(int)stonex.size() - 1] +
MapX, stoney[(int)stoney.size() - 1] + MapY);
                                stone[(int)stone.size() - 1]->ShowBitmap();
                            }
                            skillhit = false;
                            animationSkillL.Reset();
                        }
                    }
                    else
                    {
                        animationStandL.OnMove();
                    }
                }
                else
                {
                    animationStandL.OnMove();
                }
            }
            else
            {
                if (timer_skill != -1 && (clock() - timer_skill) / CLOCKS_PER_SEC <= 15 && currentMap ==
3)
                {
                    animationSkillL.OnMove();
                }
                else
                {
                    animationWalkL.OnMove();
                }
            }
        }
    }
}

int Monster::gethp()
{
    return hp;
}

```

```

    }
    int Monster::getHit(int cx, int cy, int skillx1, int skilly1, int skillx2, int skilly2, int damage, bool finalbitmap, int*
totaldamage, vector<Item*>* item)
    {
        bool flag = false;
        int srcx = x + MapX + (animationStandR.Width() / 2);
        for (int k = y + MapY; k <= y + MapY + animationStandR.Height(); k++)
        {
            if (srcx >= skillx1 && srcx <= skillx2 && k >= skilly1 && k <= skilly2)
            {
                flag = true;
                break;
            }
        }
        if (flag)
        {
            if (timer_gethit == -1)
            {
                timer_gethit = clock();
                if (currentMap != 3 && currentMap != 6 && currentMap != 7) {
                    if (srcx > cx)
                    {
                        x += 25;

                        if (x > Area[2])
                        {
                            x = Area[2];
                        }
                    }
                    else
                    {
                        x -= 25;

                        if (x < Area[0])
                        {
                            x = Area[0];
                        }
                    }
                }
            }
        }
        if (finalbitmap)
        {
            if (hp > 0 && damage > def)
            {
                (*totaldamage) = (*totaldamage) + (damage - def);
                hp = hp - (damage - def);

                if (damage - def > 0 && timer_skill == -1 && currentMap == 3)
                {
                    timer_skill = clock();
                    def *= 2;
                }
            }
            if (hp <= 0)
            {
                int itemnum, itemtype, potiontype;
                srand(x);

                if (currentMap == 1 || currentMap == 2)
                {
                    itemnum = rand() % 2 + 1;

                    for (int i = 0; i < itemnum; i++)
                    {
                        itemtype = rand() % 100;

                        if (itemtype >= 0 && itemtype < 70)
                        {
                            potiontype = rand() % 2;

                            if (potiontype == 0)
                            {
                                item->push_back(new Item(x + 35 * i, Area[1] - 30, "紅色藥水"));

```

```

    }
    else
    {
        item->push_back(new Item(x + 35 * i, Area[1] - 30, "藍色藥水"));
    }
}
else if (itemtype >= 70 && itemtype < 80)
{
    item->push_back(new Item(x + 35 * i, Area[1] - 30, "長劍 0"));
}
else if (itemtype >= 80 && itemtype < 90)
{
    item->push_back(new Item(x + 35 * i, Area[1] - 30, "盾牌 0"));
}
else if (itemtype >= 90 && itemtype < 100)
{
    item->push_back(new Item(x + 35 * i, Area[1] - 30, "上衣 0"));
}
}
}
else if (currentMap == 3) {
    item->push_back(new Item(x + 35, Area[1] - 30, "超級紅色藥水"));
    item->push_back(new Item(x + 70, Area[1] - 30, "超級藍色藥水"));
    item->push_back(new Item(x + 105, Area[1] - 30, "盾牌 1"));
    item->push_back(new Item(x + 140, Area[1] - 30, "長劍 1"));
    item->push_back(new Item(x + 175, Area[1] - 30, "上衣 1"));
}
else if (currentMap == 4 || currentMap == 5)
{
    itemnum = rand() % 2 + 1;

    for (int i = 0; i < itemnum; i++)
    {
        itemtype = rand() % 100;

        if (itemtype >= 0 && itemtype < 70)
        {
            potiontype = rand() % 4;

            if (potiontype == 0)
            {
                item->push_back(new Item(x + 35 * i, Area[1] - 30, "紅色藥水"));
            }
            else if (potiontype == 1)
            {
                item->push_back(new Item(x + 35 * i, Area[1] - 30, "藍色藥水"));
            }
            else if (potiontype == 2)
            {
                item->push_back(new Item(x + 35 * i, Area[1] - 30, "超級紅色藥水"));
            }
            else if (potiontype == 3)
            {
                item->push_back(new Item(x + 35 * i, Area[1] - 30, "超級藍色藥水"));
            }
        }
        else if (itemtype == 70)
        {
            item->push_back(new Item(x + 35 * i, Area[1] - 30, "長劍 2"));
        }
        else if (itemtype > 70 && itemtype < 80)
        {
            item->push_back(new Item(x + 35 * i, Area[1] - 30, "長劍 1"));
        }
        else if (itemtype >= 80 && itemtype < 90)
        {
            item->push_back(new Item(x + 35 * i, Area[1] - 30, "盾牌 1"));
        }
        else if (itemtype >= 90 && itemtype < 100)
        {
            item->push_back(new Item(x + 35 * i, Area[1] - 30, "上衣 1"));
        }
    }
}
}

```

```

    }
    }
    else if (currentMap == 6) {
        item->push_back(new Item(x + 35, Area[1] - 30, "超級紅色藥水"));
        item->push_back(new Item(x + 70, Area[1] - 30, "超級藍色藥水"));
        item->push_back(new Item(x + 105, Area[1] - 30, "盾牌 1"));
        item->push_back(new Item(x + 140, Area[1] - 30, "長劍 3"));
        item->push_back(new Item(x + 175, Area[1] - 30, "上衣 2"));
    }
    else if (currentMap == 7) {
        for (int i = 0; i < 17; i++) {
            CAudio::Instance()->Stop(i);
        }
        CAudio::Instance()->Play(17);
    }
    return expr;
}
}
}
}
return 0;
}
int Monster::Hit(int x1, int y1, int x2, int y2, int def)
{
    bool flag = false;
    int srcx;

    if (direction == 0)
    {
        srcx = x + animationStandR.Width() + MapX;
    }
    else
    {
        srcx = x + MapX;
    }

    for (int i = y + MapY; i <= y + MapY + animationStandR.Height(); i++)
    {
        if (srcx >= x1 && srcx <= x2 && i >= y1 && i <= y2)
        {
            flag = true;
            break;
        }
    }
    if (currentMap == 6 && hp > 0 && flag && !skillhit)
    {
        skillhit = true;
        return dmg - def;
    }
    else if (currentMap == 7)
    {
        for (int i = 0; i < (int)stone.size(); i++)
        {
            flag = false;
            srcx = stone[i]->Left() + stone[i]->Width() / 2 + MapX;

            for (int j = stone[i]->Top() + MapY; j <= stone[i]->Top() + stone[i]->Height() + MapY; j++)
            {
                if (srcx >= x1 && srcx <= x2 && j >= y1 && j <= y2)
                {
                    flag = true;
                    break;
                }
            }
            if (hp > 0 && flag && !skillhit)
            {
                skillhit = true;
                return 900 - def;
            }
        }
    }
}
}

```

```
        else if (hp > 0 && clock() - timer_hit >= 3000 && flag)
        {
            timer_hit = clock();
            return dmg - def;
        }
        return 0;
    }
}
```