一張含有 建築物, 室內 的圖片

自動產生的描述

組別：第 24 組

題目：WK Fighter

組員：游明憲 學號 106590016

龍昱達 學號 106590024

授課老師：陳偉凱

[**簡介** 2](#_Toc11245107)

[1. 動機 2](#_Toc11245108)

[2. 分工 2](#_Toc11245109)

[**遊戲介紹** 3](#_Toc11245110)

[1. 遊戲說明 3](#_Toc11245111)

[一、 遊玩方式 3](#_Toc11245112)

[二、 遊戲規則 4](#_Toc11245113)

[三、 特殊功能 4](#_Toc11245114)

[四、 密技 5](#_Toc11245115)

[2. 遊戲圖形 5](#_Toc11245116)

[3. 遊戲音效 9](#_Toc11245117)

[**程式設計** 10](#_Toc11245118)

[1. 程式架構: 10](#_Toc11245119)

[2. 程式類別: 11](#_Toc11245120)

[3. 程式技術: 11](#_Toc11245121)

[**結語** 13](#_Toc11245122)

[1. 問題與解決方法 13](#_Toc11245123)

[2. 時間表 13](#_Toc11245124)

[3. 貢獻比例 14](#_Toc11245125)

[4. 自我檢核表 15](#_Toc11245126)

[5. 收穫 15](#_Toc11245127)

[6. 心得、感想 16](#_Toc11245128)

[7. 對本課程的建議 18](#_Toc11245129)

[**附錄** 19](#_Toc11245130)

**簡介**

1. 動機:

起初在討論主題時，就發現我們彼此都熱愛遊玩"生死格鬥"，這是一款"格鬥遊戲"，此類型的遊戲因為上手難度偏高，擁有的玩家人數也偏少，所以我們希望的主題是做出一款有別於市面的格鬥遊戲，這款遊戲要包含幾個特點1.減少複雜的指令2.快節奏的戰鬥3.避免老手對新手的虐殺，相當然爾我們不可能去模仿任何已存在的格鬥遊戲，勢必得自己原創，這在難度上就增加了不少，再來就圖形的碰撞、角色自身的狀態變換、招式間的連結等，都是展現物件導向精神的好方法，我們對這堂課的期許除了獲得高分外，也要在大學時期留下屬於自己的作品，最後主題決定製作原創遊戲-"WK Fighter"。

1. 分工:

游明憲:程式撰寫、圖形設計、主體程式與戰鬥角色撰寫。

龍昱達:程式撰寫、圖形設計、副程式與遊戲環境撰寫。

**遊戲介紹**

1. 遊戲說明:
2. 遊玩方式:

這是個不是你死就我活的決鬥舞台，竭盡所能的運用你的智慧和操作來擊敗你的對手，取得最終勝利，而每個招式皆會消耗氣力，所以良好的控制氣力也是遊玩重點，以下為指令表:

(基本動作為角色共同的操作)

|  |  |  |
| --- | --- | --- |
| 基本動作 | P1按鍵 | P2按鍵 |
| 移動 | A、D | ←、→ |
| 普通攻擊 | G | K |
| 跳躍 | F | J |
| 衝刺 | R | I |
| 防禦 | E | P |
| 特技 | T | L |
| 大絕 | Y | O |

|  |  |
| --- | --- |
| 火柴人組合技 | 指令 |
| 練氣 | 下+防禦 |
| 重攻擊 | 下+普攻 |
| 地面三連擊 | 普攻x3 |
| 空中二連擊 | (空中時)普攻x2 |
| 天勾拳 | 上+普攻 |
| 一飛沖天 | 上+特技 |
| 神龍擺尾 | (空中時)下+普攻 |
| 突刺 | 衝刺+普攻 |
| 橫掃千軍 | 衝刺+特技 |
| 遁空 | (空中時)上+普攻 |

|  |  |
| --- | --- |
| Rina組合技 | 指令 |
| 魔力回復 | 下+防禦 |
| 重攻擊 | 下+普攻 |
| 地面三連擊 | 普攻x3 |
| 空中二連擊 | (空中時)普攻x2 |
| 旋風龍捲 | 上+普攻 |
| 迴光 | 上+特技 |
| 空襲腳 | (空中時)下+普攻 |
| 閃瞬 | 衝刺+普攻 |
| 閃光箭雨 | 下+特技 |

1. 遊戲規則:

當有一方的血量歸零，就代表那方輸了，而另一方則獲勝。

為了避免過長的戰鬥，在畫面中央上方有個倒數時間，時間歸零血量較多者獲勝，假如血量相同，則會進入驟死時間，雙方血量都會降為1，

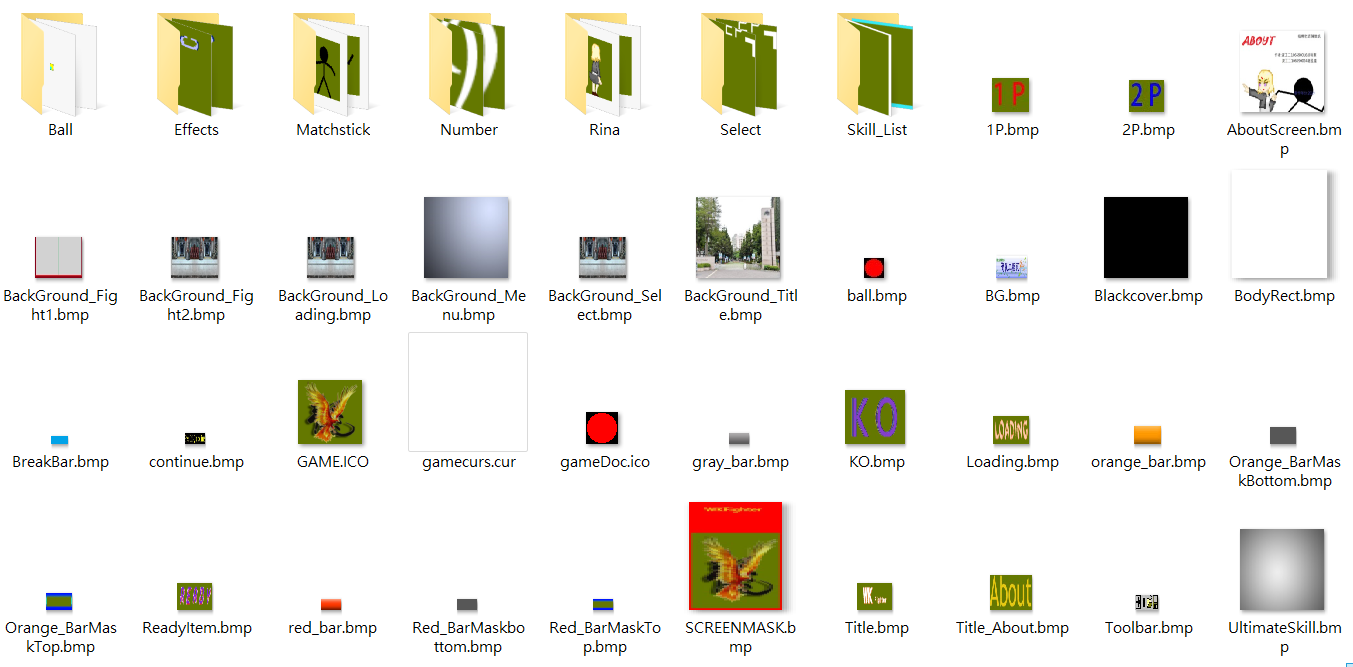
意味著只要被碰到一下即死。

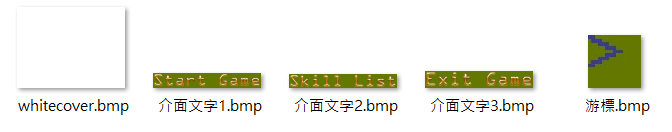
1. 特殊功能:

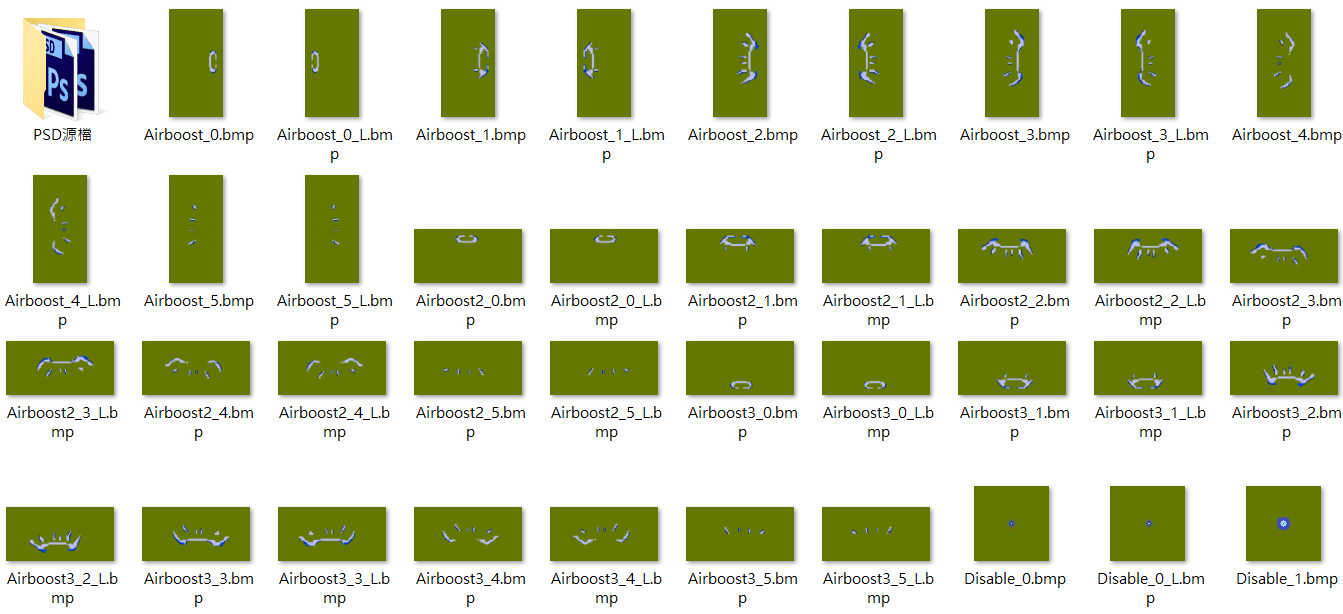
為了避免有人一直按住防禦，降低遊玩體驗和節奏，除了使用防禦會消耗氣力外，也鼓勵玩家在正確的時機按下防禦，我們設計了完美防禦系統，當玩家在被攻擊前很短的一段時間內按下防禦，會產生衝擊波，假如對手在衝擊破範圍內，將會被擊飛而且受到傷害，我們也想到可能會有玩家想用僥倖的心態使用完美防禦，所以也設計成一直連按防禦是不會觸發完美防禦的。

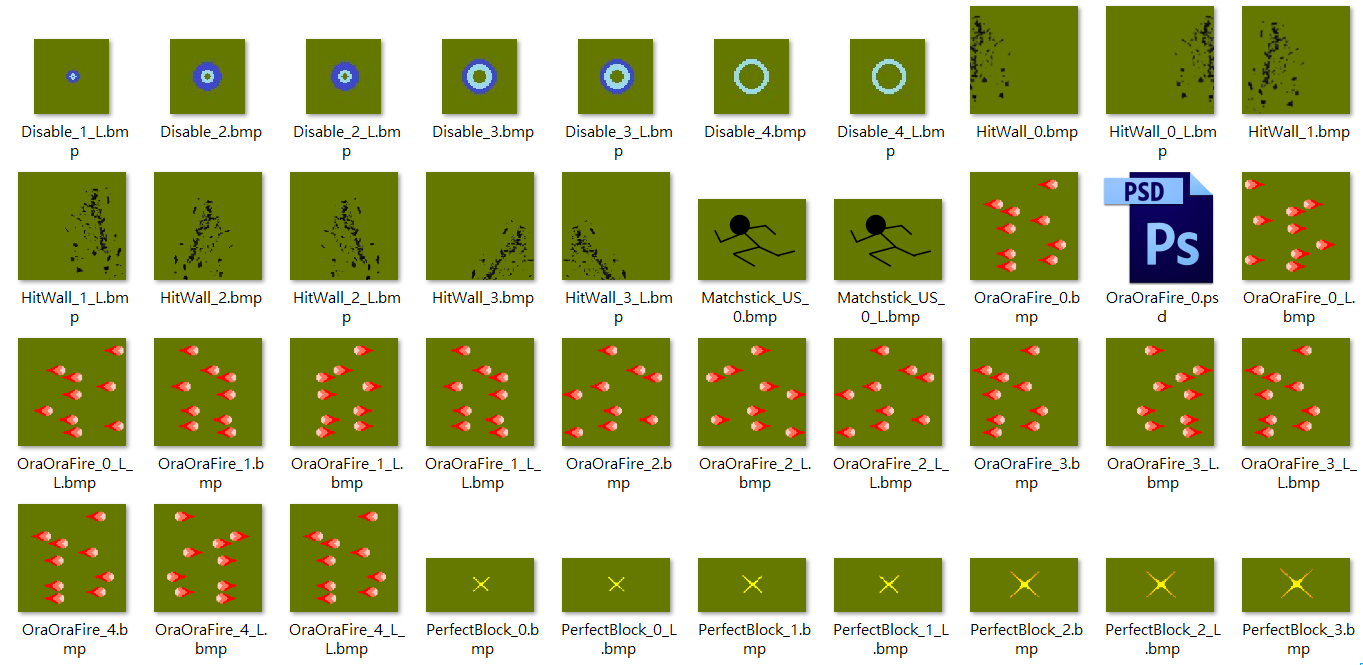
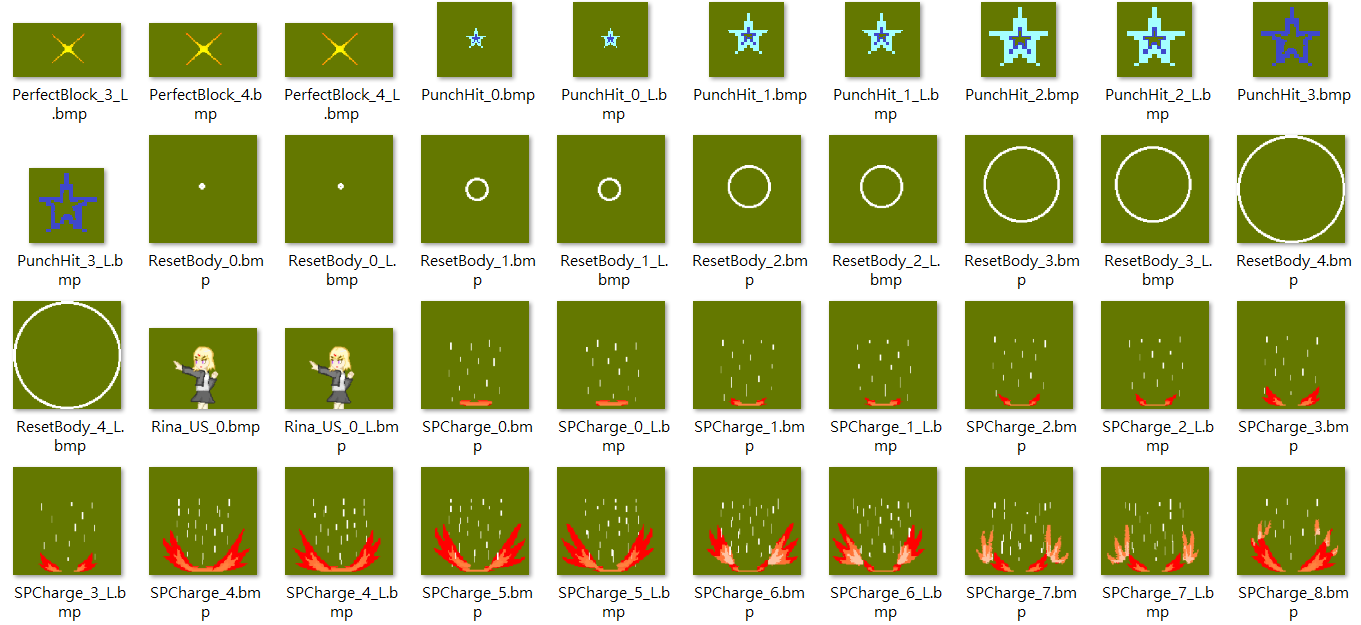
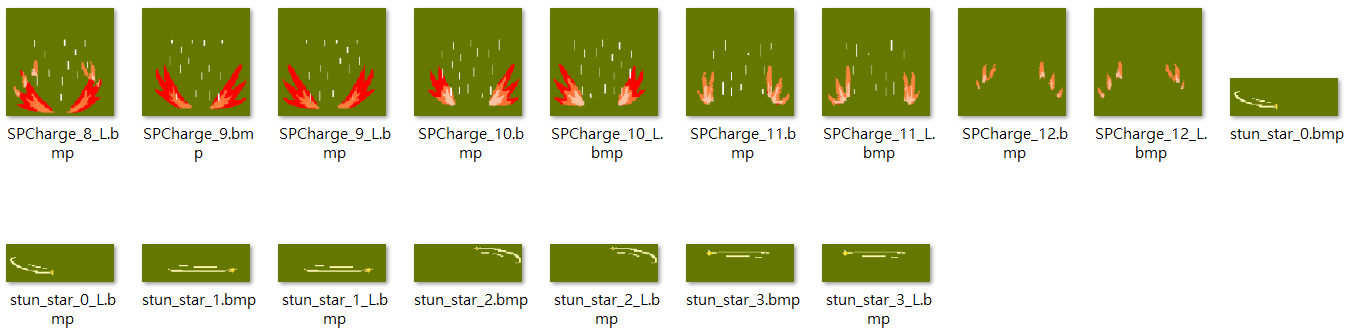
1. 密技:

戰鬥中連按空白鍵十五下，會將倒數時間設為1秒

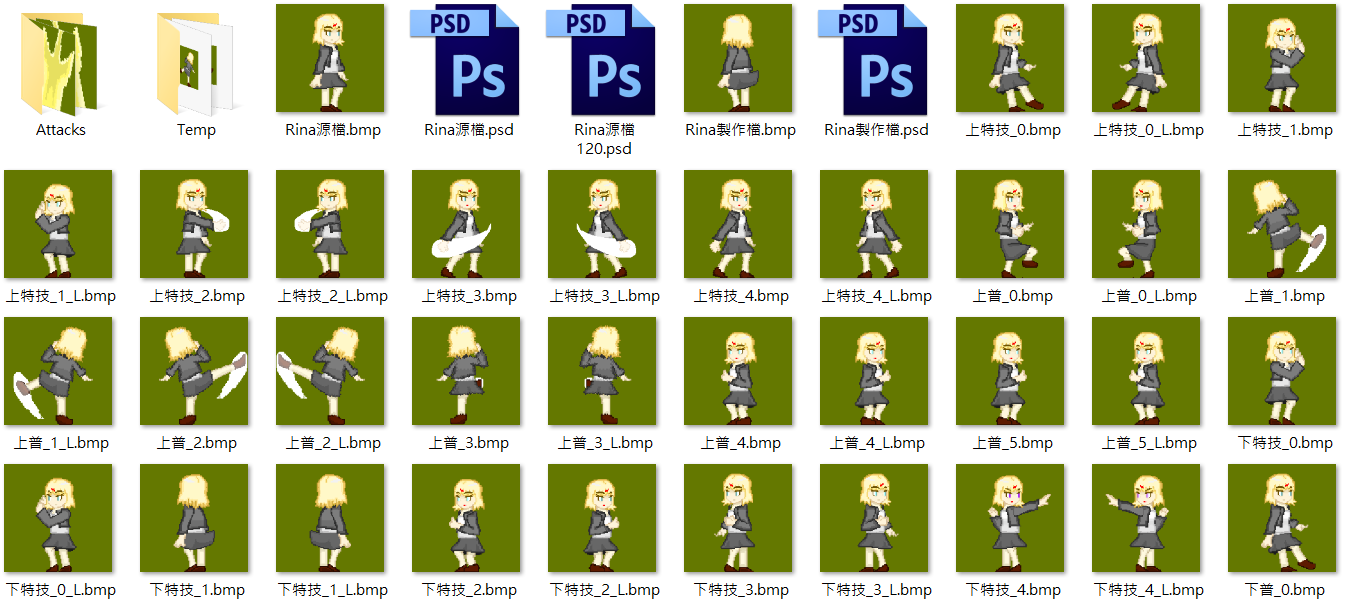
1. 遊戲圖形: (由於圖片數量過多，而且大小不盡相同，故以截圖呈現)











1. 遊戲音效:

|  |  |
| --- | --- |
| 名稱 | 用途 |
| Beep | 游標移動的音效 |
| Choose | 游標選擇確認的音效 |
| CutIn | 人物放大絕的音效 |
| Ding | 遊戲載入完成的音效 |
| Disable | 取消音效 |
| DoubleHelix-Xi | 戰鬥背景音樂(非商業用途) |
| Fire1 | 火焰特效音效 |
| HitWall | 擊中牆壁的音效 |
| jump | 跳躍音效 |
| light1 | 光屬性攻擊音效 |
| light2 | 光屬性攻擊音效 |
| NormalHit | 普通攻擊音效 |
| NormalHit2 | 普通攻擊音效 |
| rush | 衝刺音效 |
| SbDown | KO音效 |
| SliceHit | 斬擊屬性音效 |
| SPCharge | 充能音效 |
| Stoned | 暈眩音效 |
| Title | 開頭畫面音樂 |

**程式設計**

1. 一張含有 螢幕擷取畫面 的圖片

   自動產生的描述程式架構:

一張含有 螢幕擷取畫面 的圖片

自動產生的描述一張含有 螢幕擷取畫面 的圖片

自動產生的描述

1. 程式類別:

|  |  |  |  |
| --- | --- | --- | --- |
| 類別名稱 | .h檔行數 | .cpp檔行數 | 說明 |
| WKAudio | 34 | 29 | 音效介面 |
| WKBitmap | 124 | 546 | 圖形介面控制 |
| AttackObj | 99 | 608 | 攻擊物件類別 |
| Characters | 251 | 2880 | 角色類別(繼承自BattlePlayer) |
| Bar | 19 | 74 | 控制類比血條 |
| BattlePlayer | 251 | 987 | 基本戰鬥者類別 |
| EffectSprite | 24 | 84 | 特效類別 |
| SelectionBitmap | 16 | 85 | 選項圖形類別 |
| KeyBoardState | 83 | 164 | 鍵盤現狀保存類別 |
| Keycode | 80 | 12 | 鍵盤Key碼直覺化 |
| CollisionSensor | 14 | 143 | 圖形碰撞類別 |
| FunctionUser | 16 | 60 | 複雜的整合涵式 |
| TypeConverter | 7 | 22 | 提供基礎C++資料型別轉換的類別 |

1. 程式技術:

|  |  |
| --- | --- |
| 項目 | 應用原理 |
| 圖片像素碰撞 | 以Cbitmap讀取時預先載入圖型的像素詳值，把遊戲設定的透明色設做碰撞無效區、之後並以該部林陣列範圍做And排定。 |
| 戰鬥角色類別 | 以BattlePlayer為基礎類別，使用眾多虛擬函數去表現接下來繼承的子類別各自的不同，在外部使用指標指向子類別。 |
| 閃電讀取 | 本身架構以Thread呈現讀取時畫面，為了加速讀取速度，可以預先設定那些需要使用像素碰撞以節省時間，並且把角色實體先設定好，不選擇使用指標的new作法，讓在選擇一樣的組合時，可以繞過角色建置。 |
| 鍵盤組件物件化 | 以預設變數開關方式來常駐鍵盤現狀變數，而不是使用觸發，如此可以更加精準紀錄曾經所按的狀況，適合格鬥遊戲 |
| 標頭互相導入 | AttackOBJ跟Battleplayer互相引入，其中AttackOBJ有個指標變數是Belone跟Target，可以反向操作所屬類別以及所屬類別的敵人之變數。 |
| 物理引擎與視窗引擎 | 常駐變數每次會隨著時間在物件裡操作，其中位移以int視窗座標、double實際座標、速度、加速度來呈現。 |

**結語**

1. 問題與解決方法:

問題1:圖片讀取過多bug

解決方法:讀取時會用到的HBitmap，在讀取圖檔中並沒有被刪除，刪除之後這個大Bug便消失了

問題2:像素碰撞效能不足會lag

解決方法:我們使用混和的方式，矩形碰撞跟像素碰撞互相使用，在效能和精確上取得平衡

P.S.所遇到的問題真的就這些

1. 時間表:

組員A:106590016游明憲

組員B:106590024龍昱達

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 回數 | 日期 | A時數 | B時數 | A工作項目 | B工作項目 |
| 0 | 2019-03-08 ~  2019-03-14 | 26小時 | 8小時 | 重整遊戲框架、框架排版 | 研究Framework練習作業 |
| 1 | 2019-03-15 ~  2019-03-21 | 16小時 | 11小時 | 完成開頭畫面測試 | Memory leaks概念測試、素材初步編碼 |
| 2 | 2019-03-22 ~  2019-03-28 | 21小時 | 9小時 | 完成初始角色”火柴人”程式碼及圖片 | 素材整理及背景繪圖、少部分特效 |
| 3 | 2019-03-29 ~  2019-04-04 | 16小時 | 6小時 | 火柴人普攻完成 | 繪圖輔助、練習新增人物攻擊、暈眩特效繪製 |
| 4 | 2019-04-08 ~  2019-04-14 | 11小時 | 5小時 | 場地互動系統建置(擊牆判定等等) | 繪圖輔助、擊牆特效繪製 |
| 5 | 2019-04-12 ~  2019-04-18 | 17小時 | 15小時 | 補足火柴人剩餘招式(空中連段與大絕) | 遊戲平衡優化、血條製作 |
| 6 | 2019-04-19 ~  2019-04-25 | 15小時 | 7小時 | 製作自己的讀取畫面(閃電讀取) | 火柴人動作BugFix與完美防禦程式 |
| 7 | 2019-04-26 ~  2019-05-02 | 12小時 | 10小時 | 第二支角色(Rina)建置  閃電讀取第二代 | 遊戲初步流程設計、倒數時間製作 |
| 8 | 2019-05-03 ~  2019-05-09 | 15小時 | 10小時 | 遊戲框架資源超載BugFix | 選角畫面程式與圖片 |
| 9 | 2019-05-10 ~  2019-05-16 | 6小時 | 10小時 | 簡化程式以利更新 | 重整遊戲選角畫面與遊戲美工概念 |
| 10 | 2019-05-17 ~  2019-05-23 | 7.5小時 | 5小時 | 新增Rina的空工連段 | 招式表規劃與圖片設計 |
| 11 | 2019-05-24 ~  2019-05-30 | 5.5小時 | 9小時 | 新增Rina的暈眩攻擊與大絕、閃電讀取第三代 | 招式表實作 |
| 12 | 2019-05-31 ~  2019-06-06 | 5小時 | 5小時 | 遊戲收尾Bug修復、安裝檔建置、製作密技 | 補足About畫面、剩餘遊戲要求 |
| 合計 |  | 173小時 | 110小時 |  |  |

1. 貢獻比例

106590016游明憲:50%

106590024龍昱達:50%

1. 自我檢核表:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 項目 | 完成否 | 無法完成的原因 |
| 1 | 解決Memory leak | ■已完成 □未完成 |  |
| 2 | 自定遊戲Icon | ■已完成 □未完成 |  |
| 3 | 全螢幕啟動 | ■已完成 □未完成 |  |
| 4 | 有About畫面 | ■已完成 □未完成 |  |
| 5 | 初始畫面說明按鍵及滑鼠之用法與密技 | ■已完成 □未完成 |  |
| 6 | 上傳setup/apk/source檔 | ■已完成 □未完成 |  |
| 7 | setup檔可正確執行 | ■已完成 □未完成 |  |
| 8 | 報告字型、點數、對齊、行距、頁碼等格式正確 | ■已完成 □未完成 |  |
| 9 | 報告封面、側邊格式正確 | □已完成 □未完成 |  |
| 10 | 報告附錄程式格式正確 | ■已完成 □未完成 |  |

1. 收穫:

|  |  |
| --- | --- |
| 組員 | 收穫項目 |
| 106590016游明憲 | C++基礎語法與整理  C++物件建置詳細流程  C++程式引入規範及訣竅  C++中的多執行續使用  C++繼承  C++Friendclass  Visual Studio排版及#Define Region  C++memory leak原理  C++指標建立與刪除  C++指標容器與子類別用法  C++DirectDraw表層應用 |
| 106590016龍昱達 | C++基本語法  C++auto和iterator運用  C++命名空間原理  C++call by reference運用  C++static功能與用法  Visual Studio中斷點與逐步執行  C++virtual function實作  C++STL容器原理和用法  C++類別繼承 |

1. 心得、感想:

游明憲:

我想我們碰到最大的問題，肯定是撰寫期的心得報告，30%比例怎麼想都不太恰當，連老師自己都說不恰當了，我實在不理解這個分量比例要怎麼拿捏；

如果去掉這個，那應該就是在第二次DEMO以前的史詩級Bug，我們因為圖片讀取多於其他組很多，所以率先碰到了這個問題，就是在圖片讀取過多的情況下，即便圖片資源並沒有再次讀取，但是仍會有某個資料爆表的問題，那個資料就是在讀取時會用到的HBitmap，在讀取圖檔中並沒有被刪除，刪除之後這個大Bug便消失了，如果不是特別去翻老師底層的框架找到這個錯誤，不然沒有學生能夠拿這個框架做夠大的遊戲吧。

遊戲需要讀取像素碰撞卻效能不夠的問題，最後我們使用混和的方式，舉行碰撞跟像素碰撞互相使用，效能有顯著提升。

在製作讀取畫面下了一些功夫，因為我們的遊戲圖片相比其他組是超過10倍之多，如果把所有圖片都擠在一開始，會讀取太久，所以我們自己做了讀取畫面，但為了讓畫面有活動性，所以使用了以前自己根本不曾會去使用到的Thread語法。

不過這堂課是上大學以來與自己的興趣最為接近的課程，製作遊戲，雖然製作遊戲常常被其他學者或資訊業界人士說成是資工中的小菜一疊，也許製作遊戲與設計遊戲並不是在程式上的一個鑽研與突破，但是能夠讓程式產生美感、靈活運用，我到覺得也是一門學問，老師這次採取的方式主要就是讓學生自由發揮並給予協助，雖然我們這組幾乎沒有使用到任何協助，不過也因為這種教法適用於我，讓我自己成功的把上學習偷懶沒學好的C++語法一口氣學了80%以上，算是收穫良多。

龍昱達:

辛苦了一學期，終於要結束了，第一堂老師的說明仿佛只是前幾天而已，還記得那時剛聽完帶著一顆不安又興奮的心離開教室，不安的是以我現在的程度真的寫得出甚麼東西嗎，興奮的是夢想好像實現了一半，我猜大部分的同學一開始會想走資訊這條路應該都是想要寫遊戲吧，但是一路上的科目、實作好像都讓我們離初衷越來越遠，而且也漸漸消磨了我們對寫程式的熱情，不少同學因此轉系、休學，這門課大概就是像黑暗中的一盞明燈，在我們徬徨不安、猶豫不前時遇見了它，幫助我從新燃起希望，也讓我知道當初的選擇不是錯誤。回想起一開始做練習的時候，真的很廢，每行程式不僅不知道功能是甚麼連照打還會打錯，然後跟隊友做git的練習只要衝突都解決不了，只能重開一個資料夾從頭來過，然後越做越順，而且之後隊友還把全部整合新的類別，變成像api一樣只要按照方法做就好，輕鬆載入跟顯示圖片，記得第二個禮拜隊友就把像素碰撞寫出來，那時候我們冒出了惡作劇的念頭，去找了一張小精靈的地圖將道路塗上透明色，手動塞幾顆圓球上去，裝作我們好像要做小精靈一樣，而且因為真的可以移動、吃豆子、加分數，騙了不少人，很多人到第一次demo前都還以為我們要做小精靈，這邊不得不吐槽一下，班上真的有人的題目是小精靈，但是那組第一次demo的進度比我們惡作劇寫的還少。再來就是因為這門課學了不少東西，像其實我原本連PS都不會用，更別說是畫圖了，還好隊友會一點而且非常有耐心教我，現在我已經將常用工具的快捷鍵全部背起來了，然後因為要做遊戲的關係常常去隊友家，認識了他的家人，也在那吃過不少次晚餐，總覺得很不好意思，這門課帶給我的不只是知識跟友情，還有那種看著自己作品越來越完整的成就感，這些都是難以言喻的，多年後應該時不時就會回想起這段熱血的大學時光。

1. 對本課程的建議:

無

**附錄**

mygame.h

using namespace std;

namespace game\_framework {

/////////////////////////////////////////////////////////////////////////////

// 這個class為遊戲的遊戲開頭畫面物件

// 每個Member function的Implementation都要弄懂

/////////////////////////////////////////////////////////////////////////////

class CGameStateInit : public CGameState

{

public:

CGameStateInit(CGame \*g);

~CGameStateInit();

void OnInit(); // 遊戲的初值及圖形設定

void OnBeginState(); // 設定每次重玩所需的變數

void OnKeyDown(UINT, UINT, UINT); // 處理鍵盤Up的動作

void OnKeyUp(UINT, UINT, UINT);

void GameLoading();

protected:

void OnShow(); // 顯示這個狀態的遊戲畫面

void OnMove(); // 邏輯更新

private:

};

/////////////////////////////////////////////////////////////////////////////

// 這個class為遊戲的遊戲執行物件，主要的遊戲程式都在這裡

// 每個Member function的Implementation都要弄懂

/////////////////////////////////////////////////////////////////////////////

class CGameStateRun : public CGameState {

public:

CGameStateRun(CGame \*g);

~CGameStateRun();

void OnBeginState(); // 設定每次重玩所需的變數

void OnInit(); // 遊戲的初值及圖形設定

void OnKeyDown(UINT, UINT, UINT);

void OnKeyUp(UINT, UINT, UINT);

void OnLButtonDown(UINT nFlags, CPoint point); // 處理滑鼠的動作

void OnLButtonUp(UINT nFlags, CPoint point); // 處理滑鼠的動作

void OnMouseMove(UINT nFlags, CPoint point); // 處理滑鼠的動作

void OnRButtonDown(UINT nFlags, CPoint point); // 處理滑鼠的動作

void OnRButtonUp(UINT nFlags, CPoint point); // 處理滑鼠的動作

protected:

void OnMove(); // 移動遊戲元素

void OnShow(); // 顯示這個狀態的遊戲畫面

private:

};

/////////////////////////////////////////////////////////////////////////////

// 這個class為遊戲的結束狀態(Game Over)

// 每個Member function的Implementation都要弄懂

/////////////////////////////////////////////////////////////////////////////

class CGameStateOver : public CGameState {

public:

CGameStateOver(CGame \*g);

void OnBeginState(); // 設定每次重玩所需的變數

void OnInit();

protected:

void OnMove(); // 移動遊戲元素

void OnShow(); // 顯示這個狀態的遊戲畫面

private:

};

}

mygame.cpp

#pragma region LibraryImport

#include "stdafx.h"

#include "Resource.h"

#include <mmsystem.h>

#include <ddraw.h>

#include <windows.h>

#include <sstream>

#include <list>

#include <vector>

#include <thread>

#include "audio.h"

#include "gamelib.h"

#include "mygame.h"

#include "MainFrm.h"

#include "Keycode.h"

#include "KeyBoardState.h"

#include "WKBitmap.h"

#include "BattlePlayer.h"

#include "CollisionSensor.h"

#include "TypeConverter.h"

#include "WKAudio.h"

#include "Bar.h"

#include "Characters.h"

#include "FunctionUser.h"

#include "SelectionBitmap.h"

#pragma endregion

//命名空間引入

#pragma region NamespaceImport

using namespace std;

using namespace CollisionSensor\_namespace;

using namespace TypeConverter\_namespace;

using namespace WKAudio\_namespace;

using namespace FunctionUser\_namespace;

#pragma endregion

namespace game\_framework

{

#pragma region 變數

//本遊戲全域變數

#pragma region Global

//邏輯

int GameAction = 0;//遊戲場景

const bool DebugMode = false;//是否啟用Debug模式

const bool LoaddingBoost = false;//使否啟用讀取加速

bool CloseingDebug = false;

//顯示

CameraPosition Camera;//遊戲鏡頭

const COLORREF TransparentColor = RGB(100, 120, 0);//透明色設定

const int left = 1;

const int right = 2;

//各腳色預載圖

Matchstick Matchstick\_1 = Matchstick(1);

Matchstick Matchstick\_2 = Matchstick(2);

Rina Rina\_1 = Rina(1);

Rina Rina\_2 = Rina(2);

//戰鬥

#pragma region 戰鬥畫面變數

double ReadyTimer;

BattlePlayer \*Player1;//1P戰鬥者k

BattlePlayer \*Player2;//2P戰鬥者

int Player1Character = 1;//Player1選擇的角色ID

int Player2Character = 0;//Player1選擇的角色ID

BitmapPicture BK;//戰鬥背景

BitmapPicture BlackCover;//戰鬥黑幕

BitmapPicture WhiteCover;//戰鬥白幕

BitmapPicture ReadyBmp;//ReadyBitmap

BitmapPicture KoBmp;//ReadyBitmap

int 密技 = 0;

double BlackCoverfactor;//戰鬥黑幕大小

Bar Bar\_HP1;//玩家1血量

Bar Bar\_HP2;//玩家2血量

Bar Bar\_SP1;//玩家1體力

Bar Bar\_SP2;//玩家2體力

Bar Bar\_RE1;//玩家1殘餘回復量

Bar Bar\_RE2;//玩家2殘餘回復量

Bar Bar\_Player1Break;//玩家1失衡值

Bar Bar\_Player2Break;//玩家2失衡值

BitmapPicture Bar\_HP1\_MaskTop;

BitmapPicture Bar\_HP1\_MaskBottom;

BitmapPicture Bar\_SP1\_MaskTop;

BitmapPicture Bar\_SP1\_MaskBottom;

BitmapPicture Bar\_HP2\_MaskTop;

BitmapPicture Bar\_HP2\_MaskBottom;

BitmapPicture Bar\_SP2\_MaskTop;

BitmapPicture Bar\_SP2\_MaskBottom;

BitmapPicture Player1\_Name;

BitmapPicture Player2\_Name;

BitmapPicture CutInMask;

double WinnerTimer;

double WinnerTimer2;

bool SomeBodyDown;

int WinnerID;

int Battle\_Timer = 99;

int Cycle\_Timer = 60;

bool onBattle = true;

#pragma endregion

//聲音

const Audio\_ID Sounds;//音效資源編碼

//鍵盤

const Keycode Keys;//鍵盤字典物件

KeyBoardState KeyState\_now;//當前的鍵盤狀態

KeyBoardState KeyState\_last;//前一瞬間的鍵盤狀態

#pragma endregion

//開頭畫面變數

#pragma region GameAction\_Title

BitmapPicture BackGround\_Title;

BitmapPicture Title\_Bitmap;

BitmapPicture Title\_Start;

BitmapPicture Title\_SkillTable;

BitmapPicture Title\_Exit;

BitmapPicture Title\_Cursor;

BitmapPicture Title\_About;

BitmapPicture BackGround\_Select;

#pragma endregion

//主選單變數

#pragma region GameAction\_Menu

BitmapPicture BackGround\_Menu;

vector <SelectionBitmap> TitleSelects;

BitmapPicture Characters\_Menu;

BitmapAnimation P1RoleChoose = BitmapAnimation("RoleP1Choose", 70, GroundPosition, true, false, false);

BitmapAnimation P2RoleChoose = BitmapAnimation("RoleP1Choose", 560, GroundPosition, true, false, false);

BitmapAnimation P1RoleSelect = BitmapAnimation("RoleP1Select", 70, GroundPosition, true, false, false);

BitmapAnimation P2RoleSelect = BitmapAnimation("RoleP1Select", 560, GroundPosition, true, false, false);

BitmapAnimation P1Selector = BitmapAnimation("P1Selector", 267, 288, true, false, false);

BitmapAnimation P2Selector = BitmapAnimation("P2Selector", 354, 288, true, false, false);

BitmapAnimation P1P2Selector = BitmapAnimation("P1P2Selector", 267, 288, true, false, false);

BitmapAnimation Number\_Digits = BitmapAnimation("number", 400, 10, true, false, false);

BitmapAnimation Number\_Ten\_Digits = BitmapAnimation("number", 360, 10, true, false, false);

int TitleSelection = 0;

int P1Selection = 0;

int P2Selection = 0;

bool SelectedP1 = false;

bool SelectedP2 = false;

int Charaters\_Menu\_X[6] = { 267, 354, 441, 267, 354, 441 };

int Charaters\_Menu\_Y[6] = { 288, 288, 288, 393, 393, 393 };

BitmapPicture Skill\_List;

BitmapPicture BackGround\_Skill\_List;

BitmapPicture Characters\_Menu\_2;

BitmapAnimation Small\_Selector = BitmapAnimation("Small\_Selector", 39, 24, true, false, false);

BitmapAnimation Stick\_Skill = BitmapAnimation("Stick\_Skill", 400, 24, true, false, false);

BitmapAnimation Rina\_Skill = BitmapAnimation("Rina\_Skill", 400, 24, true, false, false);

int Charaters\_Menu\_2\_X[6] = { 39, 126, 211, 39, 126, 211 };

int Charaters\_Menu\_2\_Y[6] = { 24, 24, 24, 130, 130, 130 };

int SmallSelection;

bool SelectedSmall;

int page;

BitmapPicture Big\_Selector;

int Skill\_Menu\_X[10] = { 428, 585, 428, 585, 428, 584, 428, 584, 428, 584 };

int Skill\_Menu\_Y[10] = { 58, 57, 163, 162, 270, 269, 374, 373, 479, 478 };

int BigSelection;

BitmapPicture About;

int Title\_Menu\_X[4] = { 225, 225, 225, 605 };

int Title\_Menu\_Y[4] = { 300, 400, 500, 550 };

#pragma endregion

//圖取畫面圖片

#pragma region LoadingPicture

BitmapPicture LoadingPicture;//讀取畫面圖示

thread LoadingThread;//讀取執行序

bool LoadingStart = false;//開始讀取布林值

bool LoadingDone = false;//讀取完成布林值

bool LoadingTemp = false;//放大縮小

double Loadingfactor = 1;//放大縮小Double

BitmapPicture LoadingBK;//讀取畫面圖示

#pragma endregion

#pragma endregion

#pragma region 套裝函式內容

//這些函式拿來作套裝程式編寫※

#pragma region RENEWAL Fuction And Objects

void ExitGame()

{

PostMessage(AfxGetMainWnd()->m\_hWnd, WM\_CLOSE, 0, 0);

}

//效率很差盡量不要使用

void Showtext(char \*mes, int X, int Y, int fontsize, COLORREF BK, COLORREF FC, int WorkingLayer, int TargetLayer)

{

if (WorkingLayer == TargetLayer)

{

CDC \*pDC = CDDraw::GetBackCDC(); // 取得 Back Plain 的 CDC

CFont f, \*fp;

f.CreatePointFont(fontsize \* 10, "Times New Roman"); // 產生 font f; 160表示16 point的字

fp = pDC->SelectObject(&f); // 選用 font f

pDC->SetBkColor(BK);

pDC->SetTextColor(FC);

pDC->TextOut(X, Y, mes);

pDC->SelectObject(fp); // 放掉 font f (千萬不要漏了放掉)

CDDraw::ReleaseBackCDC(); // 放掉 Back Plain 的 CDC

}

}

//決定使用角色

//決定使用角色

BattlePlayer \*DecideCharacter(int PlayerIndex, int Decide)

{

Matchstick\_1.Restvalues(1);

Matchstick\_2.Restvalues(2);

Rina\_1.Restvalues(1);

Rina\_2.Restvalues(2);

BattlePlayer \*Player;

if (Decide == 0)//選擇火柴人

{

if (PlayerIndex == 1)

Player = &Matchstick\_1;

else if (PlayerIndex == 2)

Player = &Matchstick\_2;

}

if (Decide == 1)//選擇火柴人

{

if (PlayerIndex == 1)

Player = &Rina\_1;

else if (PlayerIndex == 2)

Player = &Rina\_2;

}

return Player;

}

#pragma endregion

//讀取總檔案

#pragma region GameLoading

void CGameStateInit::GameLoading()

{

//讀取所有圖檔--Begin

ShowInitProgress(10);

LoadingPicture = BitmapPicture("Content\\Bitmaps\\Loading.bmp", 150, 200, true, false, false);

LoadingPicture.LoadTexture(TransparentColor);

BackGround\_Title = BitmapPicture("Content\\Bitmaps\\BackGround\_Title.bmp", 0, 0, true, false, false);

BackGround\_Title.LoadTexture(TransparentColor);

Title\_Bitmap = BitmapPicture("Content\\Bitmaps\\Title.bmp", 100, 0, true, false, false);

Title\_Bitmap.LoadTexture(TransparentColor);

BackGround\_Menu = BitmapPicture("Content\\Bitmaps\\BackGround\_Menu.bmp", 0, 0, true, false, false);

BackGround\_Menu.LoadTexture(TransparentColor);

Title\_Start = BitmapPicture("Content\\Bitmaps\\介面文字1.bmp", 260, 300, true, false, false);

Title\_Start.LoadTexture(TransparentColor);

Title\_SkillTable = BitmapPicture("Content\\Bitmaps\\介面文字2.bmp", 260, 400, true, false, false);

Title\_SkillTable.LoadTexture(TransparentColor);

Title\_Exit = BitmapPicture("Content\\Bitmaps\\介面文字3.bmp", 285, 500, true, false, false);

Title\_Exit.LoadTexture(TransparentColor);

Title\_Cursor = BitmapPicture("Content\\Bitmaps\\游標.bmp", 225, 300, true, false, false);

Title\_Cursor.LoadTexture(TransparentColor);

Title\_About = BitmapPicture("Content\\Bitmaps\\Title\_About.bmp", 640, 550, true, false, false);

Title\_About.LoadTexture(TransparentColor);

BackGround\_Select = BitmapPicture("Content\\Bitmaps\\BackGround\_Select.bmp", -400, 0, true, false, false);

BackGround\_Select.LoadTexture(TransparentColor);

Characters\_Menu = BitmapPicture("Content\\Bitmaps\\Select\\Characters\_Menu.bmp", 265, 300, true, false, false);

Characters\_Menu.LoadTexture(TransparentColor);

P1RoleChoose.AutoLoadBitmaps("Select", "RoleP1Choose", 6, 0, false, TransparentColor);

P2RoleChoose.AutoLoadBitmaps("Select", "RoleP1Choose", 6, 0, false, TransparentColor);

P1RoleSelect.AutoLoadBitmaps("Select", "RoleP1Select", 6, 0, false, TransparentColor);

P2RoleSelect.AutoLoadBitmaps("Select", "RoleP1Select", 6, 0, false, TransparentColor);

P1RoleChoose.BitmapisRight = true;

P1RoleSelect.BitmapisRight = true;

P2RoleChoose.BitmapisRight = false;

P2RoleSelect.BitmapisRight = false;

P1Selector.AutoLoadBitmaps("Select", "P1Selector", 2, 0, false, TransparentColor);

P2Selector.AutoLoadBitmaps("Select", "P2Selector", 2, 0, false, TransparentColor);

P1P2Selector.AutoLoadBitmaps("Select", "P1P2Selector", 2, 0, false, TransparentColor);

Number\_Digits.AutoLoadBitmaps("Number", "number", 10, 0, false, TransparentColor);

Number\_Ten\_Digits.AutoLoadBitmaps("Number", "number", 10, 0, false, TransparentColor);

Skill\_List = BitmapPicture("Content\\Bitmaps\\Skill\_List\\Skill\_List.bmp", 400, 20, true, false, false);

Skill\_List.LoadTexture(TransparentColor);

BackGround\_Skill\_List = BitmapPicture("Content\\Bitmaps\\Whitecover.bmp", 0, 0, true, false, false);

BackGround\_Skill\_List.LoadTexture(TransparentColor);

LoadingBK = BitmapPicture("Content\\Bitmaps\\BackGround\_Loading.bmp", -400, 0, true, false, false);

LoadingBK.LoadTexture(TransparentColor);

Characters\_Menu\_2 = BitmapPicture("Content\\Bitmaps\\Select\\Characters\_Menu.bmp", 40, 25, true, false, false);

Characters\_Menu\_2.LoadTexture(TransparentColor);

Small\_Selector.AutoLoadBitmaps("Skill\_List", "Small\_Selector", 2, 0, false, TransparentColor);

Stick\_Skill.AutoLoadBitmaps("Skill\_List", "Stick\_Skill", 2, 0, false, TransparentColor);

Rina\_Skill.AutoLoadBitmaps("Skill\_List", "Rina\_Skill", 2, 0, false, TransparentColor);

Big\_Selector = BitmapPicture("Content\\Bitmaps\\Skill\_List\\Big\_Selector.bmp", 428, 53, true, false, false);

Big\_Selector.LoadTexture(TransparentColor);

About = BitmapPicture("Content\\Bitmaps\\AboutScreen.bmp", 0, 0, true, false, false);

About.LoadTexture(TransparentColor);

LoadingBK = BitmapPicture("Content\\Bitmaps\\BackGround\_Loading.bmp", -400, 0, true, false, false);

LoadingBK.LoadTexture(TransparentColor);

ShowInitProgress(50);

//讀取所有音效--Begin

LoadSounds(Sounds.Ding, "Content\\Sounds\\ding.wav");

LoadSounds(Sounds.Rush, "Content\\Sounds\\rush.wav");

LoadSounds(Sounds.Jump, "Content\\Sounds\\jump.wav");

LoadSounds(Sounds.SPCharge, "Content\\Sounds\\SPCharge.wav");

LoadSounds(Sounds.NormalHit, "Content\\Sounds\\NormalHit.wav");

LoadSounds(Sounds.HitWall, "Content\\Sounds\\HitWall.wav");

LoadSounds(Sounds.Disable, "Content\\Sounds\\Disable.wav");

LoadSounds(Sounds.Stoned, "Content\\Sounds\\Stoned.wav");

LoadSounds(Sounds.Fire1, "Content\\Sounds\\Fire1.wav");

LoadSounds(Sounds.CutIn, "Content\\Sounds\\CutIn.wav");

LoadSounds(Sounds.NormalHit2, "Content\\Sounds\\NormalHit2.wav");

LoadSounds(Sounds.SbDown, "Content\\Sounds\\SbDown.wav");

LoadSounds(Sounds.Title, "Content\\Sounds\\Title.wav");

LoadSounds(Sounds.Beep, "Content\\Sounds\\Beep.wav");

LoadSounds(Sounds.Choose, "Content\\Sounds\\Choose.wav");

LoadSounds(Sounds.SliceHit, "Content\\Sounds\\SliceHit.wav");

LoadSounds(Sounds.light1, "Content\\Sounds\\light1.wav");

LoadSounds(Sounds.light2, "Content\\Sounds\\light2.wav");

ShowInitProgress(70);

LoadSounds(Sounds.DoubleHelixXi, "Content\\Sounds\\DoubleHelix-Xi.mp3");

ShowInitProgress(80);

}

#pragma endregion

#pragma endregion

#pragma region 遊戲內容

//各大GameAction的Show跟Move

#pragma region GameActions

void GameAction0\_initialization()

{

GameAction = 0;

TitleSelection = 0;

PlaySounds(Sounds.Title, true);

Title\_Cursor.Rect.Y = 300;

Title\_Cursor.Rect.X = 225;

}

void GameAction1\_initialization()

{

GameAction = 1;

SmallSelection = 0;

BigSelection = 0;

SelectedSmall = false;

}

void GameAction2\_initialization()

{

GameAction = 2;

P1Selection = 0;

P2Selection = 1;

page = 0;

SelectedP1 = false;

SelectedP2 = false;

}

void GameAction3\_initialization()

{

GameAction = 3;

}

void GameAction4\_initialization()

{

GameAction = 4;

}

void GameAction5\_initialization()

{

StopSounds(Sounds.Title);

GameAction = 5;

LoadingStart = false;//開始讀取布林值

LoadingDone = false;//讀取完成布林值

}

void GameAction6\_initialization()

{

GameAction = 6;

}

void GameAction7\_initialization()

{

GameAction = 7;

}

void GameAction0\_OnMove()

{

if (GameAction == 0)

{

if (KeyState\_now.Enter == true && KeyState\_last.Enter == false)

{

PlaySounds(Sounds.Choose, false);

if (TitleSelection == 0)

GameAction2\_initialization();

else if (TitleSelection == 1)

GameAction1\_initialization();

else if (TitleSelection == 2)

ExitGame();

else if (TitleSelection == 3)

GameAction3\_initialization();

}

else if ((KeyState\_now.Up == true && KeyState\_last.Up == false) || (KeyState\_now.W == true && KeyState\_last.W == false))

{

PlaySounds(Sounds.Beep, false);

TitleSelection--;

if (TitleSelection < 0)

TitleSelection += 4;

}

else if ((KeyState\_now.Down == true && KeyState\_last.Down == false) || (KeyState\_now.S == true && KeyState\_last.S == false))

{

PlaySounds(Sounds.Beep, false);

TitleSelection++;

if (TitleSelection > 3)

TitleSelection -= 4;

}

Title\_Cursor.Rect.X = Title\_Menu\_X[TitleSelection];

Title\_Cursor.Rect.Y = Title\_Menu\_Y[TitleSelection];

BackGround\_Title.OnUpdate();

Title\_Bitmap.OnUpdate();

Title\_Start.OnUpdate();

Title\_SkillTable.OnUpdate();

Title\_Exit.OnUpdate();

Title\_Cursor.OnUpdate();

Title\_About.OnUpdate();

}

}

void GameAction0\_OnShow(int i)

{

if (GameAction == 0)

{

BackGround\_Title.Draw(i, 1);

Title\_Bitmap.Draw(i, 3);

Title\_Start.Draw(i, 3);

Title\_SkillTable.Draw(i, 3);

Title\_Exit.Draw(i, 3);

Title\_Cursor.Draw(i, 3);

Title\_About.Draw(i, 3);

}

}

void GameAction1\_OnMove()

{

if (GameAction == 1)

{

if (KeyState\_now.ESC == true && KeyState\_last.ESC == false)

{

GameAction0\_initialization();

}

if (KeyState\_now.G == true && KeyState\_last.G == false)

{

SelectedSmall = false;

}

if (SelectedSmall == false)

{

page = 0;

if (KeyState\_now.F == true && KeyState\_last.F == false)

{

if (SmallSelection <= 1)

{

SelectedSmall = true;

Small\_Selector.Step = 0;

PlaySounds(Sounds.Choose, false);

}

}

else if (KeyState\_now.D == true && KeyState\_last.D == false)

{

PlaySounds(Sounds.Beep, false);

SmallSelection++;

if (SmallSelection > 5)

{

SmallSelection = 0;

}

}

else if (KeyState\_now.A == true && KeyState\_last.A == false)

{

PlaySounds(Sounds.Beep, false);

SmallSelection--;

if (SmallSelection < 0)

{

SmallSelection = 5;

}

}

else if (KeyState\_now.W == true && KeyState\_last.W == false)

{

PlaySounds(Sounds.Beep, false);

SmallSelection += 3;

if (SmallSelection > 5)

{

SmallSelection -= 6;

}

}

else if (KeyState\_now.S == true && KeyState\_last.S == false)

{

PlaySounds(Sounds.Beep, false);

SmallSelection -= 3;

if (SmallSelection < 0)

{

SmallSelection += 6;

}

}

Small\_Selector.Rect.X = Charaters\_Menu\_2\_X[SmallSelection];

Small\_Selector.Rect.Y = Charaters\_Menu\_2\_Y[SmallSelection];

}

else

{

if (KeyState\_now.D == true && KeyState\_last.D == false)

{

//if (BigSelection % 2 == 1)

//{

page++;

if (page > 1)

page = 0;

BigSelection--;

//}

//else

BigSelection++;

}

else if (KeyState\_now.A == true && KeyState\_last.A == false)

{

//if (BigSelection % 2 == 0)

//{

page--;

if (page < 0)

page = 1;

BigSelection++;

//}

//else

BigSelection--;

}

else if (KeyState\_now.W == true && KeyState\_last.W == false)

{

BigSelection -= 2;

if (BigSelection < 0)

BigSelection += 10;

}

else if (KeyState\_now.S == true && KeyState\_last.S == false)

{

BigSelection += 2;

if (BigSelection > 9)

BigSelection -= 10;

}

Big\_Selector.Rect.X = Skill\_Menu\_X[BigSelection];

Big\_Selector.Rect.Y = Skill\_Menu\_Y[BigSelection];

}

Stick\_Skill.OnUpdate("Skill\_List", Camera);

Rina\_Skill.OnUpdate("Skill\_List", Camera);

Small\_Selector.OnUpdate("Skill\_List", Camera);

BackGround\_Skill\_List.OnUpdate();

Skill\_List.OnUpdate();

Characters\_Menu\_2.OnUpdate();

Big\_Selector.OnUpdate();

}

}

void GameAction1\_OnShow(int i)

{

if (GameAction == 1)

{

Small\_Selector.DisplayBitmap->Draw(i, 4);

if (!SelectedSmall)

Small\_Selector.AutoPlay(750, true);

if (SelectedSmall)

{

if (SmallSelection == 0)

{

Stick\_Skill.Step = page;

Stick\_Skill.DisplayBitmap->Draw(i, 4);

}

else if (SmallSelection == 1)

{

Rina\_Skill.Step = page;

Rina\_Skill.DisplayBitmap->Draw(i, 4);

}

//Big\_Selector.Draw(i, 4);

}

BackGround\_Skill\_List.Draw(i, 1);

Skill\_List.Draw(i, 3);

Characters\_Menu\_2.Draw(i, 2);

}

}

void GameAction2\_OnMove()

{

if (GameAction == 2)

{

if (SelectedP1 == true && SelectedP2 == true)

{

Player1Character = P1Selection;

Player2Character = P2Selection;

P1P2Selector.Step = 0;

P1P2Selector.OnUpdate("Select", Camera);

P1P2Selector.DisplayBitmap->Draw(4, 4);

Sleep(1000);

GameAction5\_initialization();

}

if (KeyState\_now.G == true && KeyState\_last.G == false)

{

if (SelectedP1 == false)

{

GameAction0\_initialization();

}

SelectedP1 = false;

}

if (KeyState\_now.K == true && KeyState\_last.K == false)

SelectedP2 = false;

if (SelectedP1 == false)

{

if (KeyState\_now.F == true && KeyState\_last.F == false)

{

if (P1Selection <= 1)

{

SelectedP1 = true;

P1Selector.Step = 0;

PlaySounds(Sounds.Choose, false);

}

}

else if (KeyState\_now.D == true && KeyState\_last.D == false)

{

PlaySounds(Sounds.Beep, false);

P1Selection++;

if (P1Selection > 5)

{

P1Selection = 0;

}

}

else if (KeyState\_now.A == true && KeyState\_last.A == false)

{

PlaySounds(Sounds.Beep, false);

P1Selection--;

if (P1Selection < 0)

{

P1Selection = 5;

}

}

else if (KeyState\_now.W == true && KeyState\_last.W == false)

{

PlaySounds(Sounds.Beep, false);

P1Selection += 3;

if (P1Selection > 5)

{

P1Selection -= 6;

}

}

else if (KeyState\_now.S == true && KeyState\_last.S == false)

{

PlaySounds(Sounds.Beep, false);

P1Selection -= 3;

if (P1Selection < 0)

{

P1Selection += 6;

}

}

P1Selector.Rect.X = Charaters\_Menu\_X[P1Selection];

P1Selector.Rect.Y = Charaters\_Menu\_Y[P1Selection];

}

if (SelectedP2 == false)

{

if (KeyState\_now.J == true && KeyState\_last.J == false)

{

if (P2Selection <= 1)

{

SelectedP2 = true;

P2Selector.Step = 0;

PlaySounds(Sounds.Choose, false);

}

}

else if (KeyState\_now.Right == true && KeyState\_last.Right == false)

{

PlaySounds(Sounds.Beep, false);

P2Selection++;

if (P2Selection > 5)

{

P2Selection = 0;

}

}

else if (KeyState\_now.Left == true && KeyState\_last.Left == false)

{

PlaySounds(Sounds.Beep, false);

P2Selection--;

if (P2Selection < 0)

{

P2Selection = 5;

}

}

else if (KeyState\_now.Up == true && KeyState\_last.Up == false)

{

PlaySounds(Sounds.Beep, false);

P2Selection += 3;

if (P2Selection > 5)

{

P2Selection -= 6;

}

}

else if (KeyState\_now.Down == true && KeyState\_last.Down == false)

{

PlaySounds(Sounds.Beep, false);

P2Selection -= 3;

if (P2Selection < 0)

{

P2Selection += 6;

}

}

P2Selector.Rect.X = Charaters\_Menu\_X[P2Selection];

P2Selector.Rect.Y = Charaters\_Menu\_Y[P2Selection];

}

P1P2Selector.Rect.X = Charaters\_Menu\_X[P1Selection];

P1P2Selector.Rect.Y = Charaters\_Menu\_Y[P1Selection];

P1RoleChoose.OnUpdate("Select", Camera);

P2RoleChoose.OnUpdate("Select", Camera);

P1RoleSelect.OnUpdate("Select", Camera);

P2RoleSelect.OnUpdate("Select", Camera);

P1Selector.OnUpdate("Select", Camera);

P2Selector.OnUpdate("Select", Camera);

P1P2Selector.OnUpdate("Select", Camera);

}

}

void GameAction2\_OnShow(int i)

{

if (GameAction == 2)

{

BackGround\_Select.Draw(i, 0);

Characters\_Menu.Draw(i, 1);

P1RoleChoose.Step = P1Selection;

P2RoleChoose.Step = P2Selection;

P1RoleSelect.Step = P1Selection;

P2RoleSelect.Step = P2Selection;

if (P1Selection == P2Selection)

{

P1P2Selector.DisplayBitmap->Draw(i, 4);

P1P2Selector.AutoPlay(750, true);

}

else

{

P1Selector.DisplayBitmap->Draw(i, 4);

if (!SelectedP1)

P1Selector.AutoPlay(750, true);

P2Selector.DisplayBitmap->Draw(i, 4);

if (!SelectedP2)

P2Selector.AutoPlay(750, true);

}

SelectedP1 == true ? P1RoleSelect.DisplayBitmap->Draw(i, 3) : P1RoleChoose.DisplayBitmap->Draw(i, 3);

SelectedP2 == true ? P2RoleSelect.DisplayBitmap->Draw(i, 3) : P2RoleChoose.DisplayBitmap->Draw(i, 3);

}

}

void GameAction3\_OnMove()

{

if (GameAction == 3)

{

if (KeyState\_now.ESC == true && KeyState\_last.ESC == false)

{

GameAction0\_initialization();

}

}

}

void GameAction3\_OnShow(int i)

{

if (GameAction == 3)

{

About.Draw(i, 1);

}

}

void GameAction4\_OnMove()

{

if (GameAction == 4)

{

}

}

void GameAction4\_OnShow(int i)

{

if (GameAction == 4)

{

}

}

//戰鬥畫面

#pragma region 戰鬥環境

//大絕招Cover

void CutInFunction(BitmapPicture \*Cover, BattlePlayer \*Player)

{

Cover->visable = true;

string EffectName = Player->GetName() + "\_US";

Player->Effects.Content[Player->GetName() + "\_US"].visable = true;

if (Player->Effects.Content[Player->GetName() + "\_US"].Rect.X > 100 || Player->Effects.Content[Player->GetName() + "\_US"].Rect.X < -100)

{

Player->Effects.Content[Player->GetName() + "\_US"].Rect.X -= 65;

}

else

{

Player->Effects.Content[Player->GetName() + "\_US"].Rect.X -= 5;

}

if (Player->Effects.Content[Player->GetName() + "\_US"].Rect.X < -800)

{

Cover->visable = false;

Player->Effects.Content[Player->GetName() + "\_US"].visable = false;

Player->NeedCutIn = false;

}

CameraPosition C;

Player->Effects.Content[Player->GetName() + "\_US"].OnUpdate("Effects", C);

}

//撞擊牆壁

void GotTerrainHit(CameraPosition \*C, BattlePlayer \*Player, BitmapPicture BK)

{

Player->Velocity\_X \*= -0.75;

Player->HP -= 15;

Player->HitFly = true;

Player->BeHitTimeMax += 200;

PlaySounds(Sounds.HitWall, false);

Player->BreakPoint += 30;

if (Player->BreakPoint > 90)

Player->BreakPoint = 90;

Player->Effects.BootEffect(&(Player->Effects.Content["HitWall"]), Camera, Player->BodyRect.X, Player->BodyRect.X - 60, Player->Rect.Y + 10, 0, 0, false, Player->IsRight);

Sleep(100);

}

//地形環境

void ProduceTerrain(CameraPosition \*C, BattlePlayer \*P1, BattlePlayer \*P2, BitmapPicture BK)

{

int CameraMax\_right = GAME\_SIZE\_X + ((BK.Rect.Width - GAME\_SIZE\_X) / 2) - GAME\_SIZE\_X;//鏡頭右邊界

int CameraMax\_Left = -(((BK.Rect.Width - GAME\_SIZE\_X) / 2) - GAME\_SIZE\_X) - GAME\_SIZE\_X;//鏡頭左邊界

#pragma region 地形傷害\_X

if ((P1->Rect.X < CameraMax\_Left))

{

if (P1->Action == "受傷"&&P1->Velocity\_X < -9)

{

GotTerrainHit(C, P1, BK);

}

}

if ((P1->Rect.X + P1->Rect.Width > CameraMax\_right + GAME\_SIZE\_X))

{

if (P1->Action == "受傷"&&P1->Velocity\_X > 9)

{

GotTerrainHit(C, P1, BK);

}

}

if ((P2->Rect.X < CameraMax\_Left))

{

if (P2->Action == "受傷"&&P2->Velocity\_X < -9)

{

GotTerrainHit(C, P2, BK);

}

}

if ((P2->Rect.X + P2->Rect.Width > CameraMax\_right + GAME\_SIZE\_X))

{

if (P2->Action == "受傷"&&P2->Velocity\_X > 9)

{

GotTerrainHit(C, P2, BK);

}

}

#pragma endregion

#pragma region 鏡頭控制

//

if (P1->Rect.X\_int + P1->Rect.Width >= GAME\_SIZE\_X&&P1->Velocity\_X > 0)

{

if (C->X\_double < CameraMax\_right)

C->X\_double += P1->Velocity\_X;

if (P2->Rect.X < C->X\_double)

{

P2->Rect.X += P1->Velocity\_X;

}

}

//

if (P1->Rect.X\_int <= 0 && P1->Velocity\_X < 0)

{

if (C->X\_double > CameraMax\_Left)

C->X\_double += P1->Velocity\_X;

if (P2->Rect.X + P2->Rect.Width > C->X\_double + GAME\_SIZE\_X)

{

P2->Rect.X += P1->Velocity\_X;

}

}

if (P2->Rect.X\_int + P2->Rect.Width >= GAME\_SIZE\_X&&P2->Velocity\_X > 0)

{

if (C->X\_double < CameraMax\_right)

C->X\_double += P2->Velocity\_X;

if (P1->Rect.X < C->X\_double)

{

P1->Rect.X += P2->Velocity\_X;

}

}

if (P2->Rect.X\_int <= 0 && P2->Velocity\_X < 0)

{

if (C->X\_double > CameraMax\_Left)

C->X\_double += P2->Velocity\_X;

if (P1->Rect.X + P1->Rect.Width > C->X\_double + GAME\_SIZE\_X)

{

P1->Rect.X += P2->Velocity\_X;

}

}

//

C->X = (int)C->X\_double;

C->Y = (int)C->Y\_double;

if (C->X > CameraMax\_right)

C->X = CameraMax\_right;

if (C->X < CameraMax\_Left)

C->X = CameraMax\_Left;

//

if (P1->Rect.X\_int < 0)

{

P1->Rect.X\_int = 0;

P1->Rect.X = P1->Rect.X\_int + C->X;

}

//

if (P1->Rect.X\_int > GAME\_SIZE\_X - P1->Rect.Width)

{

P1->Rect.X\_int = GAME\_SIZE\_X - P1->Rect.Width;

P1->Rect.X = P1->Rect.X\_int + C->X;

}

//

if (P2->Rect.X\_int < 0)

{

P2->Rect.X\_int = 0;

P2->Rect.X = P2->Rect.X\_int + C->X;

}

if (P2->Rect.X\_int > GAME\_SIZE\_X - P2->Rect.Width)

{

P2->Rect.X\_int = GAME\_SIZE\_X - P2->Rect.Width;

P2->Rect.X = P2->Rect.X\_int + C->X;

}

#pragma endregion

}

//戰鬥讀取

void BattleLoading()

{

if (LoadingDone == false)

{

//重製相機鏡頭

Camera = CameraPosition();

#pragma region 戰鬥背景

BK = BitmapPicture("Content\\Bitmaps\\BackGround\_Fight2.bmp", -400, 0, true, false, true);

BK.LoadTexture(TransparentColor);

BlackCover = BitmapPicture("Content\\Bitmaps\\Blackcover.bmp", 0, 0, false, false, false);

BlackCover.LoadTexture(TransparentColor);

WhiteCover = BitmapPicture("Content\\Bitmaps\\whitecover.bmp", 0, 0, false, false, false);

WhiteCover.LoadTexture(TransparentColor);

ReadyBmp = BitmapPicture("Content\\Bitmaps\\ReadyItem.bmp", 280, 600, false, false, false);

ReadyBmp.LoadTexture(TransparentColor);

KoBmp = BitmapPicture("Content\\Bitmaps\\KO.bmp", 0, 200, false, false, false);

KoBmp.LoadTexture(TransparentColor);

#pragma endregion

#pragma region 建置玩家變數

Player1 = DecideCharacter(1, Player1Character);

Player2 = DecideCharacter(2, Player2Character);

#pragma endregion

#pragma region 讀取玩家圖檔與設定初始參數

Player1->AutoLoadBitmaps(Player2, Camera, KeyState\_now, KeyState\_last, Sounds, TransparentColor);

Player2->AutoLoadBitmaps(Player1, Camera, KeyState\_now, KeyState\_last, Sounds, TransparentColor);

Player1->Rect.X = 230;

Player1->Rect.Y = GroundPosition;

Player2->Rect.X = 400;

Player2->Rect.Y = GroundPosition;

#pragma endregion

#pragma region 讀取血量條等等

Bar\_HP1 = Bar("Content\\Bitmaps\\red\_bar.bmp", 1, 25, 25, true);

Bar\_HP1.LoadTexture(TransparentColor);

Bar\_HP2 = Bar("Content\\Bitmaps\\red\_bar.bmp", 2, 525, 25, true);

Bar\_HP2.LoadTexture(TransparentColor);

Bar\_SP1 = Bar("Content\\Bitmaps\\orange\_bar.bmp", 1, 25, 50, true);

Bar\_SP1.LoadTexture(TransparentColor);

Bar\_SP2 = Bar("Content\\Bitmaps\\orange\_bar.bmp", 2, 625, 50, true);

Bar\_SP2.LoadTexture(TransparentColor);

Bar\_RE1 = Bar("Content\\Bitmaps\\gray\_bar.bmp", 1, 25, 25, true);

Bar\_RE1.LoadTexture(TransparentColor);

Bar\_RE2 = Bar("Content\\Bitmaps\\gray\_bar.bmp", 2, 525, 25, true);

Bar\_RE2.LoadTexture(TransparentColor);

Bar\_Player1Break = Bar("Content\\Bitmaps\\BreakBar.bmp", 1, 0, 0, true);

Bar\_Player1Break.LoadTexture(TransparentColor);

Bar\_Player2Break = Bar("Content\\Bitmaps\\BreakBar.bmp", 1, 0, 0, true);

Bar\_Player2Break.LoadTexture(TransparentColor);

Bar\_HP1\_MaskTop = BitmapPicture("Content\\Bitmaps\\Red\_BarMaskTop.bmp", 20, 25, true, false, false);

Bar\_HP1\_MaskTop.LoadTexture(TransparentColor);

Bar\_HP1\_MaskBottom = BitmapPicture("Content\\Bitmaps\\Red\_BarMaskBottom.bmp", 20, 25, true, false, false);

Bar\_HP1\_MaskBottom.LoadTexture(TransparentColor);

Bar\_SP1\_MaskTop = BitmapPicture("Content\\Bitmaps\\Orange\_BarMaskTop.bmp", 20, 50, true, false, false);

Bar\_SP1\_MaskTop.LoadTexture(TransparentColor);

Bar\_SP1\_MaskBottom = BitmapPicture("Content\\Bitmaps\\Orange\_BarMaskBottom.bmp", 20, 50, true, false, false);

Bar\_SP1\_MaskBottom.LoadTexture(TransparentColor);

Bar\_HP2\_MaskTop = BitmapPicture("Content\\Bitmaps\\Red\_BarMaskTop.bmp", 520, 25, true, false, false);

Bar\_HP2\_MaskTop.LoadTexture(TransparentColor);

Bar\_HP2\_MaskBottom = BitmapPicture("Content\\Bitmaps\\Red\_BarMaskBottom.bmp", 520, 25, true, false, false);

Bar\_HP2\_MaskBottom.LoadTexture(TransparentColor);

Bar\_SP2\_MaskTop = BitmapPicture("Content\\Bitmaps\\Orange\_BarMaskTop.bmp", 620, 50, true, false, false);

Bar\_SP2\_MaskTop.LoadTexture(TransparentColor);

Bar\_SP2\_MaskBottom = BitmapPicture("Content\\Bitmaps\\Orange\_BarMaskBottom.bmp", 620, 50, true, false, false);

Bar\_SP2\_MaskBottom.LoadTexture(TransparentColor);

#pragma endregion

#pragma region 雜圖讀取

Player1\_Name = BitmapPicture("Content\\Bitmaps\\1P.bmp", 40, GroundPosition - 220, true, false, true);

Player1\_Name.LoadTexture(TransparentColor);

Player2\_Name = BitmapPicture("Content\\Bitmaps\\2P.bmp", 620, GroundPosition - 220, true, false, true);

Player2\_Name.LoadTexture(TransparentColor);

CutInMask = BitmapPicture("Content\\Bitmaps\\UltimateSkill.bmp", 0, 0, false, false, false);

CutInMask.LoadTexture(TransparentColor);

#pragma endregion

Player1->CanControl = true;

Player2->CanControl = true;

SomeBodyDown = false;

BlackCoverfactor = 1;

ReadyTimer = 0;

LoadingDone = true;

Battle\_Timer = 99;

Cycle\_Timer = 60;

onBattle = true;

WinnerTimer = 0;

WinnerTimer2 = 0;

}

}

//戰鬥OnMove

void BattleOnMove()

{

BK.OnUpdate(Camera);

#pragma region 正常更新

if (Player1->NeedCutIn == false && Player2->NeedCutIn == false)

{

Player1->OnUpdate(Player2, Camera, KeyState\_now, KeyState\_last, Sounds, TransparentColor);

Player2->OnUpdate(Player1, Camera, KeyState\_now, KeyState\_last, Sounds, TransparentColor);

ProduceTerrain(&Camera, Player1, Player2, BK);

}

if (Player1->NeedCutIn)

CutInFunction(&CutInMask, Player1);

if (Player2->NeedCutIn)

CutInFunction(&CutInMask, Player2);

if (ReadyTimer < 1500)

{

ReadyTimer += TIMER\_TICK\_MILLIDECOND;

Player1->CanControl = false;

Player2->CanControl = false;

ReadyBmp.visable = true;

if (ReadyBmp.Rect.Y > 270)

{

ReadyBmp.Rect.Y -= 20;

if (ReadyBmp.Rect.Y < 270)

{

ReadyBmp.Rect.Y = 270;

}

}

if (ReadyTimer >= 1500)

{

ReadyTimer = 1500;

Player1->CanControl = true;

Player2->CanControl = true;

ReadyBmp.visable = false;

PlaySounds(Sounds.SbDown, false);

PlaySounds(Sounds.DoubleHelixXi, true);

}

ReadyBmp.OnUpdate();

}

if (ReadyTimer >= 1500 && onBattle)

{

Cycle\_Timer--;

if (Cycle\_Timer == 0)

{

Battle\_Timer--;

Cycle\_Timer = 60;

}

if (Battle\_Timer <= 0)

{

Battle\_Timer = 0;

}

Number\_Digits.Step = Battle\_Timer % 10;

Number\_Ten\_Digits.Step = Battle\_Timer / 10;

}

#pragma endregion

#pragma region 有人被打敗

if (Battle\_Timer > 0)

{

if (Player1->HP <= 0 && SomeBodyDown == false)

{

SomeBodyDown = true;

Player1->CanControl = false;

PlaySounds(Sounds.SbDown, false);

WinnerTimer = 0;

WinnerTimer2 = 0;

CAudio::Instance()->Stop(Sounds.DoubleHelixXi);

}

if (Player2->HP <= 0 && SomeBodyDown == false)

{

SomeBodyDown = true;

Player2->CanControl = false;

PlaySounds(Sounds.SbDown, false);

WinnerTimer = 0;

WinnerTimer2 = 0;

CAudio::Instance()->Stop(Sounds.DoubleHelixXi);

}

if (SomeBodyDown == true)

{

onBattle = false;

WinnerTimer += TIMER\_TICK\_MILLIDECOND;

WinnerTimer2 += TIMER\_TICK\_MILLIDECOND;

if (WinnerTimer < 500 && WinnerTimer2>40)

{

WinnerTimer2 = 0;

if (WhiteCover.visable)

{

WhiteCover.visable = false;

}

else

{

WhiteCover.visable = true;

}

}

if (WinnerTimer >= 1000)

{

if (KoBmp.Rect.X < 250)

{

KoBmp.Rect.X += 25;

if (KoBmp.Rect.X > 250)

{

KoBmp.Rect.X = 250;

}

}

KoBmp.visable = true;

KoBmp.OnUpdate();

}

Player1->Invincible = true;

Player2->Invincible = true;

if (Player1->HP <= 0)

{

Player1->CanControl = false;

Player1->Action = "受傷";

if (Player1->Rect.Y < GroundPosition)

Player1->Step = 1;

else

Player1->Step = 2;

Player1->AnimationUpdate(Camera);

Player1->Throughing = true;

Player1->Invincible = true;

}

if (Player2->HP <= 0)

{

Player2->CanControl = false;

Player2->Action = "受傷";

if (Player2->Rect.Y < GroundPosition)

Player2->Step = 1;

else

Player2->Step = 2;

Player2->AnimationUpdate(Camera);

Player2->Throughing = true;

Player2->Invincible = true;

}

if (WinnerTimer > 2000)

{

if (Player1->HP > 0)

{

WinnerID = 1;

}

else if (Player2->HP > 0)

{

WinnerID = 2;

}

else

{

WinnerID = 0;

}

BlackCoverfactor = ((501 - (2500 - WinnerTimer)) / 500);

BlackCover.visable = true;

if (BlackCoverfactor > 1)

{

BlackCoverfactor = 1;

GameAction2\_initialization();

}

BlackCover.OnUpdate();

}

}

}

if (Battle\_Timer == 0 && ((Player1->HP == Player2->HP) == false))

{

onBattle = false;

Player1->Invincible = true;

Player2->Invincible = true;

WinnerTimer += TIMER\_TICK\_MILLIDECOND;

WinnerTimer2 += TIMER\_TICK\_MILLIDECOND;

if (WinnerTimer < 500 && WinnerTimer2>40)

{

WinnerTimer2 = 0;

if (WhiteCover.visable)

WhiteCover.visable = false;

else

WhiteCover.visable = true;

}

if (WinnerTimer >= 1000)

{

if (KoBmp.Rect.X < 250)

{

KoBmp.Rect.X += 25;

if (KoBmp.Rect.X > 250)

{

KoBmp.Rect.X = 250;

}

}

KoBmp.visable = true;

KoBmp.OnUpdate();

}

if (Player1->HP < Player2->HP)

{

Player1->CanControl = false;

Player1->Action = "受傷";

if (Player1->Rect.Y < GroundPosition)

Player1->Step = 1;

else

Player1->Step = 2;

Player1->AnimationUpdate(Camera);

Player1->Throughing = true;

Player1->Invincible = true;

}

else if (Player1->HP > Player2->HP)

{

Player2->CanControl = false;

Player2->Action = "受傷";

if (Player2->Rect.Y < GroundPosition)

Player2->Step = 1;

else

Player2->Step = 2;

Player2->AnimationUpdate(Camera);

Player2->Throughing = true;

Player2->Invincible = true;

}

if (WinnerTimer > 2000)

{

if (Player1->HP > Player2->HP)

{

WinnerID = 1;

}

else if (Player2->HP > Player1->HP)

{

WinnerID = 2;

}

else

{

WinnerID = 0;

}

BlackCoverfactor = ((501 - (2500 - WinnerTimer)) / 500);

BlackCover.visable = true;

if (BlackCoverfactor > 1)

{

BlackCoverfactor = 1;

GameAction2\_initialization();

}

BlackCover.OnUpdate();

}

}

else if (Player1->HP == Player2->HP && Battle\_Timer <= 0)

{

onBattle = false;

Battle\_Timer = 0;

Player1->HP = 1;

Player2->HP = 1;

}

#pragma endregion

Bar\_HP1\_MaskTop.OnUpdate();

Bar\_HP1\_MaskBottom.OnUpdate();

Bar\_SP1\_MaskTop.OnUpdate();

Bar\_SP1\_MaskBottom.OnUpdate();

Bar\_HP2\_MaskTop.OnUpdate();

Bar\_HP2\_MaskBottom.OnUpdate();

Bar\_SP2\_MaskTop.OnUpdate();

Bar\_SP2\_MaskBottom.OnUpdate();

Bar\_Player1Break.Rect.X = Player1->Rect.X + 55;

Bar\_Player1Break.Rect.Y = Player1->Rect.Y + 190;

Bar\_Player2Break.Rect.X = Player2->Rect.X + 55;

Bar\_Player2Break.Rect.Y = Player2->Rect.Y + 190;

Bar\_Player1Break.OnUpdate(Camera);

Bar\_Player2Break.OnUpdate(Camera);

Player1\_Name.Rect.X = Player1->Rect.X + 52;

Player1\_Name.Rect.Y = Player1->Rect.Y - 30;

Player2\_Name.Rect.X = Player2->Rect.X + 52;

Player2\_Name.Rect.Y = Player2->Rect.Y - 30;

Player1\_Name.OnUpdate(Camera);

Player2\_Name.OnUpdate(Camera);

Number\_Digits.OnUpdate("Number", Camera);

Number\_Ten\_Digits.OnUpdate("Number", Camera);

if (KeyState\_now.ESC)

{

StopSounds(Sounds.DoubleHelixXi);

GameAction2\_initialization();

}

if (KeyState\_now.Space&&KeyState\_last.Space == false)

{

密技++;

}

if (密技 > 15)

{

Battle\_Timer = 1;

密技 = 0;

}

}

//戰鬥OnShow

void BattleOnShow(int i)

{

BK.Draw(i, 1);

BlackCover.Draw(i, 8, BlackCoverfactor);

WhiteCover.Draw(i, 8);

KoBmp.Draw(i, 7);

ReadyBmp.Draw(i, 7);

Player1->Draw(i, 3, Camera);

Player2->Draw(i, 3, Camera);

Bar\_HP1.Draw(i, 6, Player1->HP, Player1->HP\_Max);

Bar\_HP2.Draw(i, 6, Player2->HP, Player2->HP\_Max);

Bar\_SP1.Draw(i, 6, Player1->SP, Player1->SP\_Max);

Bar\_SP2.Draw(i, 6, Player2->SP, Player2->SP\_Max);

Bar\_Player1Break.Draw(i, 3, Player1->BreakPoint, 90, Camera);

Bar\_Player2Break.Draw(i, 3, Player2->BreakPoint, 90, Camera);

Bar\_HP1\_MaskTop.Draw(i, 7);

Bar\_HP1\_MaskBottom.Draw(i, 5);

Bar\_SP1\_MaskTop.Draw(i, 7);

Bar\_SP1\_MaskBottom.Draw(i, 5);

Bar\_HP2\_MaskTop.Draw(i, 7);

Bar\_HP2\_MaskBottom.Draw(i, 5);

Bar\_SP2\_MaskTop.Draw(i, 7);

Bar\_SP2\_MaskBottom.Draw(i, 5);

Bar\_RE1.Draw(i, 5, Player1->HP + Player1->recovery, Player1->HP\_Max);

Bar\_RE2.Draw(i, 5, Player2->HP + Player2->recovery, Player2->HP\_Max);

Player1\_Name.Draw(i, 3);

Player2\_Name.Draw(i, 3);

CutInMask.Draw(i, 4);

Number\_Digits.DisplayBitmap->Draw(i, 3);

Number\_Ten\_Digits.DisplayBitmap->Draw(i, 3);

}

#pragma endregion

void GameAction5\_OnMove()

{

if (GameAction == 5)

{

LoadingResource(BattleLoading, &LoadingThread, &LoadingStart, &LoadingDone);

LoadingPicture.OnUpdate();

LoadingBK.OnUpdate();

if (LoadingStart == false && LoadingDone == true)

{

GameAction6\_initialization();

}

}

}

void GameAction5\_OnShow(int i)

{

if (GameAction == 5)

{

if (LoadingTemp == false)

{

Loadingfactor += 0.0025;

if (Loadingfactor > 1.1)

{

LoadingTemp = true;

}

}

else

{

Loadingfactor -= 0.0025;

if (Loadingfactor < 0.9)

{

LoadingTemp = false;

}

}

LoadingPicture.Draw(i, 3, Loadingfactor);

LoadingBK.Draw(i, 1);

}

}

void GameAction6\_OnMove()

{

if (GameAction == 6)

{

BattleOnMove();

}

}

void GameAction6\_OnShow(int i)

{

if (GameAction == 6)

{

BattleOnShow(i);

}

}

void GameAction7\_OnMove()

{

if (GameAction == 7)

{

GameAction5\_initialization();

}

}

void GameAction7\_OnShow(int i)

{

if (GameAction == 7)

{

}

}

#pragma endregion

#pragma endregion

#pragma region 底層mygame.cpp的運作程序(基本上不用更改)

//程式開始

#pragma region Program Initialize

CGameStateInit::CGameStateInit(CGame \*g) : CGameState(g)

{

}

CGameStateInit::~CGameStateInit()

{

}

void CGameStateInit::OnBeginState()

{

}

#pragma endregion

//遊戲開頭讀取畫面

#pragma region Game Loading Progress

void CGameStateInit::OnInit()

{

//讀取開始

ShowInitProgress(0);

GameLoading();

}

void CGameStateRun::OnInit()// 讀取檔案

{

}

void CGameStateOver::OnInit()

{

ShowInitProgress(100);

PlaySounds(Sounds.Ding, false);

}

#pragma endregion

//遊戲開頭畫面

#pragma region Game start screen

void CGameStateInit::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

KeyState\_now.UpdateState\_Down(nChar);

}

void CGameStateInit::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)

{

KeyState\_now.UpdateState\_Up(nChar);

}

void CGameStateInit::OnShow()

{

for (int i = 0; i < 5; i++)

{

BackGround\_Title.Draw(i, 1);

Title\_Bitmap.Draw(i, 3);

Showtext("Press [SPACE] to start the game", 175, 450, 20, RGB(0, 0, 0), RGB(255, 255, 255), i, 3);

Showtext("Cheat:you can press [SPACE] 16 times on the battle to achieve that", 65, 500, 15, RGB(0, 0, 0), RGB(255, 0, 0), i, 3);

}

}

void CGameStateInit::OnMove()

{

BackGround\_Title.OnUpdate();

Title\_Bitmap.OnUpdate();

if (KeyState\_now.Space == true && KeyState\_last.Space == false)

{

GameAction0\_initialization();

GotoGameState(GAME\_STATE\_RUN);

}

KeyState\_last = KeyState\_now;

}

#pragma endregion

//遊戲進行畫面

#pragma region GameRunning

//除了開頭以外的遊戲主體(將以GameAction切換遊戲視窗)

CGameStateRun::CGameStateRun(CGame \*g) : CGameState(g)

{

}

CGameStateRun::~CGameStateRun()

{

}

void CGameStateRun::OnBeginState()

{

PlaySounds(Sounds.Ding, false);

}

//GameState LogicUpdate

void CGameStateRun::OnMove()

{

GameAction0\_OnMove();

GameAction1\_OnMove();

GameAction2\_OnMove();

GameAction3\_OnMove();

GameAction4\_OnMove();

GameAction5\_OnMove();

GameAction6\_OnMove();

GameAction7\_OnMove();

KeyState\_last = KeyState\_now;

}

//GameState ShowBitmaps

void CGameStateRun::OnShow()

{

for (int i = 0; i < 10; i++)

{

GameAction0\_OnShow(i);

GameAction1\_OnShow(i);

GameAction2\_OnShow(i);

GameAction3\_OnShow(i);

GameAction4\_OnShow(i);

GameAction5\_OnShow(i);

GameAction6\_OnShow(i);

GameAction7\_OnShow(i);

}

}

void CGameStateRun::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

KeyState\_now.UpdateState\_Down(nChar);

}

void CGameStateRun::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)

{

KeyState\_now.UpdateState\_Up(nChar);

}

//滑鼠處理事件

#pragma region MouseState

void CGameStateRun::OnLButtonDown(UINT nFlags, CPoint point) // 處理滑鼠的動作

{

}

void CGameStateRun::OnLButtonUp(UINT nFlags, CPoint point) // 處理滑鼠的動作

{

}

void CGameStateRun::OnMouseMove(UINT nFlags, CPoint point) // 處理滑鼠的動作

{

// 沒事。如果需要處理滑鼠移動的話，寫code在這裡

}

void CGameStateRun::OnRButtonDown(UINT nFlags, CPoint point) // 處理滑鼠的動作

{

}

void CGameStateRun::OnRButtonUp(UINT nFlags, CPoint point) // 處理滑鼠的動作

{

}

#pragma endregion

#pragma endregion

//Game End-遊戲結束畫面==退回開頭畫面

#pragma region GameOverState

CGameStateOver::CGameStateOver(CGame \*g) : CGameState(g)

{

}

void CGameStateOver::OnMove()

{

GotoGameState(GAME\_STATE\_INIT);

}

void CGameStateOver::OnBeginState()

{

}

void CGameStateOver::OnShow()

{

}

#pragma endregion

#pragma endregion

}

WKAudio.h

#pragma once

using namespace std;

namespace WKAudio\_namespace

{

// 定義各種音效的編號

//在這裡加入你要使用的聲音名稱，數值不能重複

struct Audio\_ID

{

const int Ding = 0;

const int Rush = 1;

const int Jump = 2;

const int SPCharge = 3;

const int BKMusic = 4;

const int NormalHit = 5;

const int HitWall = 6;

const int Disable = 7;

const int Stoned = 8;

const int Fire1 = 9;

const int CutIn = 10;

const int NormalHit2 = 11;

const int SbDown = 12;

const int DoubleHelixXi = 13;

const int Title = 14;

const int Beep = 15;

const int Choose = 16;

const int SliceHit = 17;

const int light1 = 18;

const int light2 = 19;

};

void LoadSounds(int , string);

void PlaySounds(int, bool);

void StopSounds(int soundID);

}

WKAudio.cpp

#include "stdafx.h"

#include <ddraw.h>

#include "gamelib.h"

#include "audio.h"

#include "WKAudio.h"

using namespace std;

using namespace game\_framework;

namespace WKAudio\_namespace

{

void LoadSounds(int ID, string path)

{

//範例

//CAudio::Instance()->Load(0, "sounds\\ding.wav");

CAudio::Instance()->Load(ID, &path[0]); // 載入編號0的聲音ding.wav

}

//播放聲音(ID,是否重複撥放)

void PlaySounds(int soundID, bool replay)

{

CAudio::Instance()->Play(soundID, replay);

}

void StopSounds(int soundID)

{

CAudio::Instance()->Stop(soundID);

}

}

WKBitmap.h

#pragma once

#include "TypeConverter.h"

using namespace std;

using namespace TypeConverter\_namespace;

namespace game\_framework

{

struct BitMapRectangle

{

int X\_int = 0;

int Y\_int = 0;

double X = 0;

double Y = 0;

int Width = 0;

int Height = 0;

};

struct CameraPosition

{

int X = 0;

int Y = 0;

double X\_double = 0;

double Y\_double = 0;

};

class BitmapPicture :public CMovingBitmap

{

friend class CDDraw;

public:

//隨緣

BitmapPicture();

//(是否可見)

BitmapPicture(bool);

//(路徑,是否可見)

BitmapPicture(string, bool);

//(路徑,是否可見,是否受到鏡頭影響)

BitmapPicture(string, bool, bool);

//(路徑,X,Y,是否可見,是否可碰撞,是否受到鏡頭影響)

BitmapPicture(string, int, int, bool, bool, bool);

//解構子

~BitmapPicture();

//方法函式

//讀取圖檔(只需輸入透明色)

void LoadTexture(COLORREF);

//讀取圖檔(使用ResourcePath,透明色)

void LoadTexture(bool, COLORREF);

//讀取圖檔(路徑,是否可像素碰撞,透明色)

void LoadTexture(char \*, bool, COLORREF);

void OnUpdate();

void OnUpdate(CameraPosition);

//繪出圖型

virtual void Draw(int, int);

virtual void Draw(int, int,double);

//設定資源路徑連結

void SetResourcePath(string);

//得到資源路徑連結位置

string GetResourcePath();

//屬性成員

BitMapRectangle Rect;//圖片材質矩形 掌管座標跟長寬，會在讀取檔案時設定完成，並且可以在外部更動

vector<vector<bool> > EffectRect;//碰撞有效區域，執行像素碰撞使用

bool visable;//是否可見

bool CanPixelCollision;//是否可像素碰撞，如果該圖片無需使用則關閉，可大幅提升效率

bool InSideCamera;

private:

string ResourcePath;//更改名字後建議重讀圖檔

};

class BitmapAnimation

{

public:

//建構子

BitmapAnimation();

//(是否可見)

BitmapAnimation(bool);

//(路徑,是否可見)

BitmapAnimation(string, bool);

//(路徑,是否可見,是否受到鏡頭影響)

BitmapAnimation(string, bool, bool);

//(路徑,X,Y,是否可見,是否可碰撞,是否受到鏡頭影響)

BitmapAnimation(string, int, int, bool, bool, bool);

//解構子

~BitmapAnimation();

//方法函式

void AutoPlay(int, bool);//自動撥放圖片(頻率,是否重複撥放)

virtual void AutoLoadBitmaps(string, int, bool, COLORREF);//依照Name自動讀取檔名

virtual void AutoLoadBitmaps(string, string, int, double, bool, COLORREF);//依照資料夾跟Name自動讀取檔名 //Effect要有左右

virtual void OnUpdate();//更新函式，放在OnMove中(圖片才能動)

virtual void OnUpdate(CameraPosition);//更新函式，且隨著視角移動

virtual void OnUpdate(string, CameraPosition);//Effectfolder，在檔名前預加字串

string GetName();//得到Name

void SetName(string);//更改名字後需要立刻重讀圖檔!!

//屬性變數

int AutoMaxStep;//使用自動播放時的最大步驟數

int drawlayer;//繪圖圖層

double PreAutoFrequence;//預設的播放頻率

bool visable;//是否可見

bool CanPixelCollision;//是否使用像素碰撞，所有動作分割都會套用

bool InSideCamera;//是否受到鏡頭影響

bool BitmapisRight;//是否向右

bool TrackPoint;//是否持續追蹤某一點

bool loop = false;//是否重複("預設=否")

//現狀變數

double AutoPlayTimer;//使用自動播放時的計時器

int Step;//當前步驟數

double Acceleration\_X;//X加速度

double Acceleration\_Y;//Y加速度

double Velocity\_X = 0;//X軸速度

double Velocity\_Y = 0;//Y軸速度

double Acceleration\_gravity = 0.5;//重力加速度

//屬性成員

BitMapRectangle Rect;//圖片材質矩形 掌管座標跟長寬，會在讀取檔案時設定完成，並且可以在外部更動

BitmapPicture \*DisplayBitmap;//永遠把顯示用的Bitmap指向對應的BitmapPicture==當前所顯示的BitmapPicture

map<string, BitmapPicture> BitmapPictures;//該Animation的所有圖片動作

private:

string Name;//此可動圖檔的名稱[資源命名規則:放在Content\\Bitmaps\\Name\\Name\_Step.bmp]

};

}

WKBitmap.cpp

#pragma once

#include "stdafx.h"

#include <ddraw.h>

#include <windows.h>

#include <stdio.h>

#include <sstream>

#include "gamelib.h"

#include "WKBitmap.h"

#include "TypeConverter.h"

using namespace std;

using namespace TypeConverter\_namespace;

namespace game\_framework

{

/////////////////////////////////////////////////////////////////////////////

// BitmapPicture: 提供一個以下功能具備的Bitmap圖片

//1.透明色設定

//2.執行像素碰撞

//3.可以移動

/////////////////////////////////////////////////////////////////////////////

BitmapPicture::BitmapPicture()

{

Rect.X = 0;

Rect.Y = 0;

Rect.X\_int = 0;

Rect.Y\_int = 0;

visable = false;

ResourcePath = "";

CanPixelCollision = false;

InSideCamera = false;

}

BitmapPicture::BitmapPicture(bool vis)

{

Rect.X = 0;

Rect.Y = 0;

Rect.X\_int = 0;

Rect.Y\_int = 0;

visable = vis;

ResourcePath = "";

CanPixelCollision = false;

InSideCamera = false;

}

BitmapPicture::BitmapPicture(string respath, bool vis)

{

Rect.X = 0;

Rect.Y = 0;

Rect.X\_int = 0;

Rect.Y\_int = 0;

visable = vis;

ResourcePath = respath;

CanPixelCollision = false;

InSideCamera = false;

}

BitmapPicture::BitmapPicture(string respath, bool vis, bool inside)

{

Rect.X = 0;

Rect.Y = 0;

Rect.X\_int = 0;

Rect.Y\_int = 0;

visable = vis;

ResourcePath = respath;

CanPixelCollision = false;

InSideCamera = inside;

}

BitmapPicture::BitmapPicture(string respath, int X\_in, int Y\_in, bool vis, bool canhit, bool inside)

{

Rect.X = X\_in;

Rect.Y = Y\_in;

Rect.X\_int = X\_in;

Rect.Y\_int = Y\_in;

visable = vis;

ResourcePath = respath;

CanPixelCollision = canhit;

InSideCamera = inside;

}

BitmapPicture::~BitmapPicture()

{

}

void BitmapPicture::LoadTexture(COLORREF color)

{

char \*cc = new char[3000];

strcpy(cc, ResourcePath.c\_str());

if (this->isBitmapLoaded == false)

{

const int nx = 0;

const int ny = 0;

GAME\_ASSERT(!isBitmapLoaded, "A bitmap has been loaded. You can not load another bitmap !!!");

HBITMAP hbitmap = (HBITMAP)LoadImage(NULL, cc, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

if (hbitmap == NULL)

{

char error\_msg[300];

sprintf(error\_msg, "Loading bitmap from file \"%s\" failed !!!", cc);

GAME\_ASSERT(false, error\_msg);

}

CBitmap \*bmp = CBitmap::FromHandle(hbitmap); // memory will be deleted automatically

BITMAP bitmapSize;

bmp->GetBitmap(&bitmapSize);

this->location.left = nx; this->location.top = ny;

this->location.right = nx + bitmapSize.bmWidth;

this->location.bottom = ny + bitmapSize.bmHeight;

this->SurfaceID = CDDraw::RegisterBitmap(cc, color);

this->isBitmapLoaded = true;

Rect.Width = Width();

Rect.Height = Height();

if (CanPixelCollision)//如果此圖片需要碰撞區才使用

{

CDC dc;

CDC\* pDC = CDDraw::GetBackCDC();

dc.CreateCompatibleDC(pDC);

for (int y = 0; y < bitmapSize.bmHeight; y++)

{

EffectRect.push\_back(vector<bool>());

for (int x = 0; x < bitmapSize.bmWidth; x++)

{

EffectRect[y].push\_back(false);

if (RGB(GetRValue(dc.GetPixel(x, y)), GetGValue(dc.GetPixel(x, y)), GetBValue(dc.GetPixel(x, y))) != color)

{

EffectRect[y][x] = true;

}

}

}

CDDraw::ReleaseBackCDC();

}

bmp->DeleteObject();

}

delete[] cc;

}

void BitmapPicture::LoadTexture(bool CanHit, COLORREF color)

{

char \*cc = new char[3000];

strcpy(cc, ResourcePath.c\_str());

if (this->isBitmapLoaded == false)

{

CanPixelCollision = CanHit;

const int nx = 0;

const int ny = 0;

GAME\_ASSERT(!isBitmapLoaded, "A bitmap has been loaded. You can not load another bitmap !!!");

HBITMAP hbitmap = (HBITMAP)LoadImage(NULL, cc, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

if (hbitmap == NULL)

{

char error\_msg[300];

sprintf(error\_msg, "Loading bitmap from file \"%s\" failed !!!", cc);

GAME\_ASSERT(false, error\_msg);

}

CBitmap \*bmp = CBitmap::FromHandle(hbitmap); // memory will be deleted automatically

BITMAP bitmapSize;

bmp->GetBitmap(&bitmapSize);

this->location.left = nx; this->location.top = ny;

this->location.right = nx + bitmapSize.bmWidth;

this->location.bottom = ny + bitmapSize.bmHeight;

this->SurfaceID = CDDraw::RegisterBitmap(cc, color);

this->isBitmapLoaded = true;

Rect.Width = Width();

Rect.Height = Height();

if (CanHit)//如果此圖片需要碰撞區才使用

{

CDC dc;

CDC\* pDC = CDDraw::GetBackCDC();

dc.CreateCompatibleDC(pDC);

for (int y = 0; y < bitmapSize.bmHeight; y++)

{

EffectRect.push\_back(vector<bool>());

for (int x = 0; x < bitmapSize.bmWidth; x++)

{

EffectRect[y].push\_back(false);

if (RGB(GetRValue(dc.GetPixel(x, y)), GetGValue(dc.GetPixel(x, y)), GetBValue(dc.GetPixel(x, y))) != color)

{

EffectRect[y][x] = true;

}

}

}

CDDraw::ReleaseBackCDC();

}

bmp->DeleteObject();

}

delete[] cc;

}

void BitmapPicture::LoadTexture(char \*filename, bool CanHit, COLORREF color)

{

if (this->isBitmapLoaded == false)

{

CanPixelCollision = CanHit;

const int nx = 0;

const int ny = 0;

GAME\_ASSERT(!isBitmapLoaded, "A bitmap has been loaded. You can not load another bitmap !!!");

HBITMAP hbitmap = (HBITMAP)LoadImage(NULL, filename, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

if (hbitmap == NULL) {

char error\_msg[300];

sprintf(error\_msg, "Loading bitmap from file \"%s\" failed !!!", filename);

GAME\_ASSERT(false, error\_msg);

}

CBitmap \*bmp = CBitmap::FromHandle(hbitmap); // memory will be deleted automatically

BITMAP bitmapSize;

bmp->GetBitmap(&bitmapSize);

this->location.left = nx; this->location.top = ny;

this->location.right = nx + bitmapSize.bmWidth;

this->location.bottom = ny + bitmapSize.bmHeight;

this->SurfaceID = CDDraw::RegisterBitmap(filename, color);

this->isBitmapLoaded = true;

Rect.Width = Width();

Rect.Height = Height();

if (CanHit)//如果此圖片需要碰撞區才使用

{

CDC dc;

CDC\* pDC = CDDraw::GetBackCDC();

dc.CreateCompatibleDC(pDC);

for (int y = 0; y < bitmapSize.bmHeight; y++)

{

EffectRect.push\_back(vector<bool>());

for (int x = 0; x < bitmapSize.bmWidth; x++)

{

EffectRect[y].push\_back(false);

COLORREF rgb = dc.GetPixel(x, y);

BYTE r = GetRValue(rgb);

BYTE g = GetGValue(rgb);

BYTE b = GetBValue(rgb);

if (RGB(r, g, b) != color)

{

EffectRect[y][x] = true;

}

}

}

CDDraw::ReleaseBackCDC();

}

bmp->DeleteObject();

}

}

void BitmapPicture::OnUpdate()

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

}

void BitmapPicture::OnUpdate(CameraPosition Camera)

{

if (InSideCamera)

{

Rect.X\_int = (int)(Rect.X - Camera.X);

Rect.Y\_int = (int)(Rect.Y - Camera.Y);

}

else

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

}

}

void BitmapPicture::Draw(int CurrentLayer, int TargetLayer)

{

if (CurrentLayer == TargetLayer&&this->visable == true)//直到他的圖片層級才可以顯示

{

SetTopLeft(Rect.X\_int, Rect.Y\_int);

ShowBitmap();

}

}

void BitmapPicture::Draw(int CurrentLayer, int TargetLayer, double fra)

{

if (CurrentLayer == TargetLayer&&this->visable == true)//直到他的圖片層級才可以顯示

{

SetTopLeft(Rect.X\_int, Rect.Y\_int);

int xl, xr, yt, yb;

xl = (int)(this->Rect.X + (this->Rect.Width\*(1 - fra)\*0.5));

yt = (int)(this->Rect.Y + (this->Rect.Height\*(1 - fra)\*0.5));

xr = (int)((this->Rect.X + this->Rect.Width) - (this->Rect.Width\*(1 - fra)\*0.5));

yb = (int)((this->Rect.Y + this->Rect.Height) - (this->Rect.Height\*(1 - fra)\*0.5));

ShowBitmap(xl, yt, xr, yb);

}

}

void BitmapPicture::SetResourcePath(string namein)

{

ResourcePath = namein;

}

string BitmapPicture::GetResourcePath()

{

return ResourcePath;

}

/////////////////////////////////////////////////////////////////////////////

// BitmapAnimation: 提供一個以下功能具備的Bitmap圖片

//1.透明色設定

//2.執行像素碰撞(每張圖都有獨立的碰撞判斷)

//3.可以移動

//4.包含了一系列的BitmapPicture，可以此做成會動的圖片

/////////////////////////////////////////////////////////////////////////////

BitmapAnimation::BitmapAnimation()

{

Name = "";

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = false;

visable = false;

BitmapisRight = true;

drawlayer = 0;

TrackPoint = false;

Velocity\_X = 0;

Velocity\_Y = 0;

Acceleration\_X = 0;

Acceleration\_Y = 0;

}

BitmapAnimation::BitmapAnimation(bool vis)

{

Name = "";

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = false;

visable = vis;

BitmapisRight = true;

drawlayer = 0;

TrackPoint = false;

Velocity\_X = 0;

Velocity\_Y = 0;

Acceleration\_X = 0;

Acceleration\_Y = 0;

}

BitmapAnimation::BitmapAnimation(string namein, bool vis)

{

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = false;

Name = namein;

visable = vis;

BitmapisRight = true;

drawlayer = 0;

TrackPoint = false;

Velocity\_X = 0;

Velocity\_Y = 0;

Acceleration\_X = 0;

Acceleration\_Y = 0;

}

BitmapAnimation::BitmapAnimation(string namein, bool vis, bool inside)

{

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = false;

Name = namein;

visable = vis;

InSideCamera = inside;

BitmapisRight = true;

drawlayer = 0;

TrackPoint = false;

Velocity\_X = 0;

Velocity\_Y = 0;

Acceleration\_X = 0;

Acceleration\_Y = 0;

}

BitmapAnimation::BitmapAnimation(string namein, int X, int Y, bool vis, bool CanHit, bool inside)

{

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = CanHit;

Name = namein;

visable = vis;

InSideCamera = inside;

Rect.X = X;

Rect.Y = Y;

Rect.X\_int = X;

Rect.Y\_int = Y;

BitmapisRight = true;

drawlayer = 0;

TrackPoint = false;

Velocity\_X = 0;

Velocity\_Y = 0;

Acceleration\_X = 0;

Acceleration\_Y = 0;

}

BitmapAnimation::~BitmapAnimation()

{

}

void BitmapAnimation::AutoPlay(int frequence, bool replay)

{

if (AutoPlayTimer >= frequence)

{

AutoPlayTimer = 0;

if (Step < AutoMaxStep - 1)

{

Step += 1;

}

else

{

if (replay)

{

Step = 0;

}

else

{

this->visable = false;

this->DisplayBitmap->visable = false;

}

}

}

else

{

AutoPlayTimer += TIMER\_TICK\_MILLIDECOND;

}

}

void BitmapAnimation::AutoLoadBitmaps(string name, int MaxSteps, bool CanPixelCollisionin, COLORREF color)

{

AutoMaxStep = MaxSteps;

CanPixelCollision = CanPixelCollisionin;

string StepString = IntToString(MaxSteps);

for (int i = 0; i < MaxSteps; i += 1)

{

string str = ("Content\\Bitmaps\\" + name + "\\" + name + "\_" + IntToString(i) + ".bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*cc = new char[65535];

strcpy(cc, str.c\_str());

BitmapPictures[str].LoadTexture(cc, CanPixelCollisionin, color);

delete[] cc;

}

OnUpdate();

}

//Effect要有左右

void BitmapAnimation::AutoLoadBitmaps(string folder, string name, int MaxSteps, double pre, bool CanPixelCollisionin, COLORREF color)

{

AutoMaxStep = MaxSteps;

CanPixelCollision = CanPixelCollisionin;

string StepString = IntToString(MaxSteps);

PreAutoFrequence = pre;

for (int i = 0; i < MaxSteps; i += 1)

{

string str;

str = ("Content\\Bitmaps\\" + folder + "\\" + name + "\_" + IntToString(i) + ".bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*cr = new char[65535];

strcpy(cr, str.c\_str());

BitmapPictures[str].LoadTexture(cr, CanPixelCollisionin, color);

delete[] cr;

str = ("Content\\Bitmaps\\" + folder + "\\" + name + "\_" + IntToString(i) + "\_L.bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*cl = new char[65535];

strcpy(cl, str.c\_str());

BitmapPictures[str].LoadTexture(cl, CanPixelCollisionin, color);

delete[] cl;

}

}

void BitmapAnimation::OnUpdate()

{

string Actionstring;

if (BitmapisRight)

{

Actionstring = "Content\\Bitmaps\\" + Name + "\\" + Name + "\_" + IntToString(Step) + ".bmp";

}

else

{

Actionstring = "Content\\Bitmaps\\" + Name + "\\" + Name + "\_" + IntToString(Step) + "\_L.bmp";

}

char \*cc = new char[65535];

strcpy(cc, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cc];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

DisplayBitmap->visable = visable;

delete[] cc;

}

void BitmapAnimation::OnUpdate(CameraPosition Camera)

{

string Actionstring;

if (BitmapisRight)

{

Actionstring = "Content\\Bitmaps\\" + Name + "\\" + Name + "\_" + IntToString(Step) + ".bmp";

}

else

{

Actionstring = "Content\\Bitmaps\\" + Name + "\\" + Name + "\_" + IntToString(Step) + "\_L.bmp";

}

char \*cc = new char[65535];

strcpy(cc, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cc];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

if (InSideCamera)

{

Rect.X\_int = (int)(Rect.X - Camera.X);

Rect.Y\_int = (int)(Rect.Y - Camera.Y);

}

else

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

}

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

DisplayBitmap->visable = visable;

delete[] cc;

}

void BitmapAnimation::OnUpdate(string unsingfolder, CameraPosition Camera)

{

string Actionstring;

if (BitmapisRight)

{

Actionstring = "Content\\Bitmaps\\" + unsingfolder + "\\" + Name + "\_" + IntToString(Step) + ".bmp";

}

else

{

Actionstring = "Content\\Bitmaps\\" + unsingfolder + "\\" + Name + "\_" + IntToString(Step) + "\_L.bmp";

}

char \*cc = new char[65535];

strcpy(cc, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cc];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

if (InSideCamera)

{

Rect.X\_int = (int)(Rect.X - Camera.X);

Rect.Y\_int = (int)(Rect.Y - Camera.Y);

}

else

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

}

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

DisplayBitmap->visable = visable;

delete[] cc;

}

string BitmapAnimation::GetName()

{

return Name;

}

void BitmapAnimation::SetName(string namein)

{

Name = namein;

}

}

AttackObj.h

#pragma once

#include "stdafx.h"

#include "CollisionSensor.h"

#include "TypeConverter.h"

#include "EffectSprite.h"

#include "WKAudio.h"

using namespace std;

using namespace WKAudio\_namespace;

using namespace CollisionSensor\_namespace;

namespace game\_framework

{

#define AttackObjPH AttackObj \*Attack, string BeloneName,BattlePlayer \*Belone,BattlePlayer \*Target,double Damage ,double SP\_Damege,int Mass, bool IsRight,double HitVelocity\_X, double HitVelocity\_Y,double XR,double XL, double Y,double VX,double VY,double HitTime,double MaxAliveTime,int Attributes, bool CanCombo ,bool Drawable,bool Replay ,bool HitNoon,bool HitBreak,bool CanHitFly,bool CanBeDisappear,bool CanCrackOther,string HitEffect,int HitSound,CameraPosition Camera

#define AttackObjPH\_Normal AttackObj \*Attack, BattlePlayer \*Belone, BattlePlayer \*Target, double Damage, double HitVelocity\_X, double HitVelocity\_Y, double XR, double XL, double Y, double VX, double VY, double HitTime, double MaxAliveTime, string HitEffect,const int HitSound, CameraPosition Camera

#define AttackObjPH\_Shot AttackObj \*Attack, BattlePlayer \*Belone, BattlePlayer \*Target, double Damage, double HitVelocity\_X, double HitVelocity\_Y, double XR, double XL, double Y, double VX, double VY, double HitTime, double MaxAliveTime,int Mass,bool HitNoon,bool CanBeDisappear,bool CanCrackOther, string HitEffect,const int HitSound, CameraPosition Camera

////

#define PerfectBlockTime 100

////

class BattlePlayer;

class EffectSprite;

class AttackObj :public BitmapAnimation

{

public:

AttackObj();

AttackObj(string, int, int, bool, bool, bool);

~AttackObj();

virtual void OnUpdate(string, CameraPosition);

virtual void HitAction(CameraPosition);

//現狀變數

double AliveTimer = 0;

double ComboTimer = 0;

double Timer1 = 0;

double Timer2 = 0;

bool IsHited;//已經擊中

//屬性變數

double Damage;//傷害

double SP\_Damege;//對SP造成減少

double HitTime;//使敵人僵直時間

double MaxAliveTime;//攻擊最大顯示時間

double HitVelocity\_X;

double HitVelocity\_Y;

int Mass;//武器力度

int Attributes;//狀態屬性-1=無

bool CanCombo;//可持續擊中

bool Drawable;//是否需要繪製出來

bool Replay;//是否將攻擊圖檔持續撥放

bool HitNoon;//是否攻擊完就消失

bool HitBreak;//是否破防

bool CanHitFly;//是否可擊飛

bool CanBeDisappear;//可被抵銷

bool CanCrackOther;//可抵銷別的攻擊

bool GravityEffect;//是否受重力影響

bool UseRectCollision;//使用矩形碰撞判定

string HitEffect;//擊中特效名稱

int HitSound;//擊中聲音ID

EffectSprite Effects;

double Ahead(double move);

BattlePlayer \*Belone;

BattlePlayer \*Target;

};

class AttackManager

{

public:

AttackManager();

~AttackManager();

map<string, AttackObj> AttackObjects;//儲存所有攻擊物件

virtual void AttackAutoUpdate(AttackObj \* Attack, string BeloneName, int tick, bool replay, CameraPosition Camera);

virtual void AttackReset(AttackObjPH);

virtual void AttackReset\_Normal(AttackObjPH\_Normal);

virtual void AttackReset\_Shot(AttackObjPH\_Shot);

virtual void DrawAllAttacks(int);

virtual void InsertAttacks(string BeloneName, string name, int maxstep, int drawlayer, double pre, int category, COLORREF color, CameraPosition Camera);

virtual void InsertAttacks(string BeloneName, string name, int maxstep, int drawlayer, double pre, int category, int current, COLORREF color, CameraPosition Camera);

};

}

AttackObj.cpp

#pragma once

#include "stdafx.h"

#include "Resource.h"

#include <ddraw.h>

#include "audio.h"

#include "gamelib.h"

#include "WKBitmap.h"

#include "CollisionSensor.h"

#include "WKAudio.h"

#include "AttackObj.h"

#include "EffectSprite.h"

#include "TypeConverter.h"

#include "BattlePlayer.h"

#include "FunctionUser.h"

using namespace std;

using namespace WKAudio\_namespace;

using namespace CollisionSensor\_namespace;

using namespace TypeConverter\_namespace;

using namespace FunctionUser\_namespace;

namespace game\_framework

{

AttackObj::AttackObj() :BitmapAnimation()

{

}

AttackObj::AttackObj(string namein, int X, int Y, bool vis, bool CanHit, bool inside)

{

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = CanHit;

SetName(namein);

visable = vis;

InSideCamera = inside;

Rect.X = X;

Rect.Y = Y;

Rect.X\_int = X;

Rect.Y\_int = Y;

BitmapisRight = true;

drawlayer = 0;

TrackPoint = false;

GravityEffect = false;

}

AttackObj::~AttackObj()

{

}

void AttackObj::OnUpdate(string unsingfolder, CameraPosition Camera)

{

#pragma region 尋找敵人

if (this->visable && this->IsHited == false && this->Target->Invincible == false)

{

if (this->UseRectCollision == true)

{

if (BitmapPicture\_HitRectangle(\*DisplayBitmap, Target->BodyPicture) == true)

HitAction(Camera);

}

else

{

if (PixelCollision(&(this->Target->BodyPicture), this->DisplayBitmap, 2))

HitAction(Camera);

}

}

#pragma endregion

#pragma region 尋找敵人的攻擊物件

if (this->visable&&this->CanBeDisappear)

{

map<string, AttackObj>::iterator iter;

for (iter = Target->Attacks.AttackObjects.begin(); iter != Target->Attacks.AttackObjects.end(); iter++)

{

if (iter->second.visable&&iter->second.CanCrackOther)

{

if (PixelCollision(iter->second.DisplayBitmap, this->DisplayBitmap, 2))

{

if (Mass < iter->second.Mass)

{

double HX, HY = 0;

HX = this->Rect.X + (this->Rect.Width / 2) - (Effects.Content[this->HitEffect].Rect.Width / 2);

HY = this->Rect.Y + (this->Rect.Height / 2) - (Effects.Content[this->HitEffect].Rect.Height / 2);

PlayEffect(&Effects, "Disable", Camera, HX, HX, HY - 30, this->BitmapisRight);

this->visable = false;

this->DisplayBitmap->visable = false;

this->Drawable = false;

this->IsHited = true;

Audio\_ID sounds;

PlaySounds(sounds.Disable, false);

}

else if (Mass == iter->second.Mass)

{

double HX, HY = 0;

HX = this->Rect.X + (this->Rect.Width / 2) - (Effects.Content[this->HitEffect].Rect.Width / 2);

HY = this->Rect.Y + (this->Rect.Height / 2) - (Effects.Content[this->HitEffect].Rect.Height / 2);

PlayEffect(&Effects, "Disable", Camera, HX, HX, HY - 30, this->BitmapisRight);

HX = iter->second.Rect.X + (iter->second.Rect.Width / 2) - (Effects.Content[iter->second.HitEffect].Rect.Width / 2);

HY = iter->second.Rect.Y + (iter->second.Rect.Height / 2) - (Effects.Content[iter->second.HitEffect].Rect.Height / 2);

PlayEffect(&iter->second.Effects, "Disable", Camera, HX, HX, HY - 30, iter->second.BitmapisRight);

this->visable = false;

this->DisplayBitmap->visable = false;

this->Drawable = false;

this->IsHited = true;

if (iter->second.CanBeDisappear)

{

iter->second.visable = false;

iter->second.DisplayBitmap->visable = false;

iter->second.Drawable = false;

iter->second.IsHited = true;

}

Audio\_ID sounds;

PlaySounds(sounds.Disable, false);

}

}

}

}

}

#pragma endregion

#pragma region 圖片更新

string Actionstring;

if (BitmapisRight)

Actionstring = "Content\\Bitmaps\\" + unsingfolder + "\\" + GetName() + "\_" + IntToString(Step) + ".bmp";

else

Actionstring = "Content\\Bitmaps\\" + unsingfolder + "\\" + GetName() + "\_" + IntToString(Step) + "\_L.bmp";

char \*cc = new char[65535];

strcpy(cc, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cc];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

Velocity\_X += Acceleration\_X;

Velocity\_Y += Acceleration\_Y;

if (this->GravityEffect)

Velocity\_Y += Acceleration\_gravity;

Rect.X += Velocity\_X;

Rect.Y += Velocity\_Y;

if (InSideCamera)

{

Rect.X\_int = (int)(Rect.X - Camera.X);

Rect.Y\_int = (int)(Rect.Y - Camera.Y);

}

else

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

}

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

DisplayBitmap->visable = visable;

delete[] cc;

#pragma endregion

}

void AttackObj::HitAction(CameraPosition Camera)

{

#pragma region 防禦狀態

if ((Target->Action == "防禦" || Target->Action == "防禦受傷") && (this->BitmapisRight != Target->IsRight) && this->HitBreak == false)

{

bool IsPerfectBlock = false;

#pragma region 完美格檔

if (Target->ClickDefendTimer < PerfectBlockTime)

{

Audio\_ID Sounds;

this->Target->Attacks.AttackReset\_Normal(

&(this->Target->Attacks.AttackObjects["Counterattact"]), this->Target, this->Belone,

30,

10, 8, this->Target->Rect.X + 0, this->Target->Rect.X - 0, this->Target->Rect.Y, 0, 0,

120, 130, "PunchHit", Sounds.NormalHit, Camera);

this->Target->Attacks.AttackObjects["Counterattact"].Drawable = true;

this->Target->Attacks.AttackObjects["Counterattact"].HitNoon = false;

this->Target->Attacks.AttackObjects["Counterattact"].CanHitFly = true;

IsPerfectBlock = true;

}

#pragma endregion

#pragma region 確保穿越狀態跟無敵狀態

Target->Throughing = false;

Target->Invincible = false;

#pragma endregion

#pragma region 控制方向

Target->IsRight = !(this->BitmapisRight);

this->IsHited = true;

#pragma endregion

#pragma region 消失物件

if (this->HitNoon == true)

{

this->visable = false;

this->DisplayBitmap->visable = false;

this->Drawable = false;

}

#pragma endregion

#pragma region 特效與音效

PlaySounds(this->HitSound, false);

Target->Effects.BootEffect(&(Target->Effects.Content[this->HitEffect]), Camera, Target->BodyRect.X + 3, Target->BodyRect.X - 6, Target->Rect.Y + 30, 0, 0, false, this->BitmapisRight);

#pragma endregion

#pragma region 生命值與體力值的增減

if (IsPerfectBlock == false)

{

Target->GainHP(-(this->Damage / 4));

Target->GainSP(-(this->Damage / 30));

Target->GainSP(-this->SP\_Damege);

}

#pragma endregion

#pragma region 物理狀態控制

Target->Acceleration\_X = 0;

Target->Acceleration\_Y = 0;

Target->Velocity\_X = (this->Ahead(this->HitVelocity\_X)) / 2;

Target->Velocity\_Y = 0;

Target->BeHitTimer = 0;

Target->BeHitTimeMax = 0;

Target->Step = 0;

Target->Action = "防禦受傷";

#pragma endregion

#pragma region 失衡崩解

if (Target->HitFly)

{

Target->BreakPoint += 30 / 3;

if (Target->BreakPoint > 90)

{

Target->BreakPoint = 90;

Target->BreakPointTimer = 0;

}

}

else

{

Target->BreakPoint += (5 + (Damage / 10)) / 2;

if (Target->BreakPoint > 90)

{

Target->BreakPoint = 90;

Target->BreakPointTimer = 0;

}

}

if (HitBreak&&Target->BreakPoint >= 90)

{

Audio\_ID Sounds;

PlaySounds(Sounds.Stoned, false);

Target->BeHitTimer = 0;

Target->BeHitTimeMax = 1250;

Target->BreakPoint = 0;

Target->BreakPointTimer = 0;

PlayEffect(Target, "stun\_star", Camera, Rect.X - 15, Rect.X - 15, Rect.Y);

}

#pragma endregion

//延遲

Sleep(10);

}

#pragma endregion

#pragma region 非防禦狀態或無法防禦

if (!((Target->Action == "防禦" || Target->Action == "防禦受傷") && (this->BitmapisRight != Target->IsRight) && this->HitBreak == false))

{

#pragma region 確保穿越狀態跟無敵狀態

Target->Throughing = false;

Target->Invincible = false;

#pragma endregion

#pragma region 控制方向

Target->IsRight = !(this->BitmapisRight);

this->IsHited = true;

#pragma endregion

#pragma region 消失物件

if (this->HitNoon == true)

{

this->visable = false;

this->DisplayBitmap->visable = false;

this->Drawable = false;

}

#pragma endregion

#pragma region 特效與音效

PlaySounds(this->HitSound, false);

Target->Effects.BootEffect(&(Target->Effects.Content[this->HitEffect]), Camera, Target->BodyRect.X + 3, Target->BodyRect.X - 6, Target->Rect.Y + 30, 0, 0, false, this->BitmapisRight);

#pragma endregion

#pragma region 生命值與體力值的增減

Target->GainHP(-(this->Damage));

Target->GainSP(+(this->Damage / 15));

Target->GainSP(-this->SP\_Damege);

if (((int)Target->HP) > 0)

Target->recovery = Target->recovery + (this->Damage / 1.5);

else

Target->recovery = 0;

#pragma endregion

#pragma region 屬性

if (this->Attributes >= 0)

Target->AttributeState[this->Attributes] = true;

#pragma endregion

#pragma region 物理狀態控制

Target->Acceleration\_X = 0;

Target->Acceleration\_Y = 0;

Target->Velocity\_X = Ahead(this->HitVelocity\_X);

if (HitVelocity\_Y > 0 && -Target->Velocity\_Y <= 0)

Target->Velocity\_Y = -this->HitVelocity\_Y;

else if (HitVelocity\_Y > 0 && -Target->Velocity\_Y > 0)

Target->Velocity\_Y += -this->HitVelocity\_Y;

if (HitVelocity\_Y < 0 && -Target->Velocity\_Y >= 0)

Target->Velocity\_Y = -this->HitVelocity\_Y;

else if (HitVelocity\_Y < 0 && -Target->Velocity\_Y < 0)

Target->Velocity\_Y += -this->HitVelocity\_Y;

Target->BeHitTimer = 0;

Target->BeHitTimeMax = this->HitTime;

if (!(Target->HitFly))

Target->HitFly = this->CanHitFly;

Target->Step = 0;

Target->Action = "受傷";

#pragma endregion

#pragma region 失衡崩解

if (Target->HitFly)

{

Target->BreakPoint += 30;

if (Target->BreakPoint > 90)

{

Target->BreakPoint = 90;

Target->BreakPointTimer = 0;

}

}

else

{

Target->BreakPoint += 5 + (Damage / 10);

if (Target->BreakPoint > 90)

{

Target->BreakPoint = 90;

Target->BreakPointTimer = 0;

}

}

if (HitBreak&&Target->BreakPoint >= 90)

{

Audio\_ID Sounds;

PlaySounds(Sounds.Stoned, false);

Target->BeHitTimer = 0;

Target->BeHitTimeMax = 1250;

Target->BreakPoint = 0;

Target->BreakPointTimer = 0;

PlayEffect(Target, "stun\_star", Camera, Rect.X - 15, Rect.X - 15, Rect.Y);

Target->HitFly = false;

}

#pragma endregion

//延遲

Sleep(10);

}

#pragma endregion

}

double AttackObj::Ahead(double move)

{

{

double returner = 0;

if (BitmapisRight)

{

returner = move;

}

else

{

returner = -move;

}

return returner;

}

}

AttackManager::AttackManager()

{

}

AttackManager::~AttackManager()

{

}

void AttackManager::AttackAutoUpdate(AttackObj \* Attack, string BeloneName, int tick, bool replay, CameraPosition Camera)

{

map<string, BitmapAnimation>::iterator Iter\_Effect;

for (Iter\_Effect = Attack->Effects.Content.begin(); Iter\_Effect != Attack->Effects.Content.end(); Iter\_Effect++)

Attack->Effects.EffectAutoUpdate(&(Iter\_Effect->second), (int)(((Iter\_Effect->second).PreAutoFrequence)), false, Camera);

if (Attack->visable)

{

if (Attack->IsHited&&Attack->CanCombo)

{

Attack->ComboTimer += TIMER\_TICK\_MILLIDECOND;

if (Attack->ComboTimer > TIMER\_TICK\_MILLIDECOND \* 5)

{

Attack->ComboTimer = 0;

Attack->IsHited = false;

}

}

if (Attack->visable = true)

Attack->AliveTimer += TIMER\_TICK\_MILLIDECOND;

if (Attack->AliveTimer >= Attack->MaxAliveTime)

{

Attack->visable = false;

Attack->DisplayBitmap->visable = false;

}

Attack->AutoPlay(tick, replay);

Attack->Rect.X += Attack->Velocity\_X;

Attack->Rect.Y += Attack->Velocity\_Y;

Attack->OnUpdate(BeloneName + "\\Attacks", Camera);

}

}

void AttackManager::AttackReset(AttackObjPH)

{

//屬性設定

Attack->Damage = Damage;

Attack->SP\_Damege = SP\_Damege;

Attack->HitVelocity\_X = HitVelocity\_X;

Attack->HitVelocity\_Y = HitVelocity\_Y;

Attack->BitmapisRight = IsRight;

if (Attack->BitmapisRight)

Attack->Rect.X = XR;

else

Attack->Rect.X = XL;

Attack->Rect.Y = Y;

Attack->Velocity\_X = VX;

Attack->Velocity\_Y = VY;

Attack->HitTime = HitTime;

Attack->MaxAliveTime = MaxAliveTime;

Attack->Attributes = Attributes;

Attack->CanCombo = CanCombo;

Attack->Drawable = Drawable;

Attack->Replay = Replay;

Attack->HitNoon = HitNoon;

Attack->HitBreak = HitBreak;

Attack->CanHitFly = CanHitFly;

Attack->HitEffect = HitEffect;

Attack->HitSound = HitSound;

Attack->visable = true;

Attack->IsHited = false;

Attack->AutoPlayTimer = 0;

Attack->Step = 0;

Attack->AliveTimer = 0;

Attack->Timer1 = 0;

Attack->Timer2 = 0;

Attack->ComboTimer = 0;

Attack->Belone = Belone;

Attack->Target = Target;

Attack->Mass = Mass;

Attack->CanBeDisappear = CanBeDisappear;

Attack->CanCrackOther = CanCrackOther;

Attack->GravityEffect = false;

Attack->UseRectCollision = false;

//初次更新

Attack->OnUpdate(BeloneName + "\\Attacks", Camera);

}

void AttackManager::AttackReset\_Normal(AttackObjPH\_Normal)

{

//屬性設定

Attack->Belone = Belone;

Attack->Target = Target;

Attack->Damage = Damage;

Attack->SP\_Damege = 0;

Attack->HitVelocity\_X = HitVelocity\_X;

Attack->HitVelocity\_Y = HitVelocity\_Y;

Attack->BitmapisRight = Belone->IsRight;

Attack->GravityEffect = false;

if (Attack->BitmapisRight)

Attack->Rect.X = XR;

else

Attack->Rect.X = XL;

Attack->Rect.Y = Y;

Attack->Velocity\_X = VX;

Attack->Velocity\_Y = VY;

Attack->HitTime = HitTime;

Attack->MaxAliveTime = MaxAliveTime;

Attack->Attributes = -1;

Attack->CanCombo = false;

Attack->Drawable = false;

Attack->Replay = true;

Attack->HitNoon = true;

Attack->HitBreak = false;

Attack->CanHitFly = false;

Attack->HitEffect = HitEffect;

Attack->HitSound = HitSound;

Attack->visable = true;

Attack->IsHited = false;

Attack->AutoPlayTimer = 0;

Attack->Step = 0;

Attack->AliveTimer = 0;

Attack->Timer1 = 0;

Attack->Timer2 = 0;

Attack->ComboTimer = 0;

Attack->Mass = 10;

Attack->CanBeDisappear = false;

Attack->CanCrackOther = false;

Attack->UseRectCollision = false;

//初次更新

Attack->OnUpdate(Belone->GetName() + "\\Attacks", Camera);

}

void AttackManager::AttackReset\_Shot(AttackObjPH\_Shot)

{

//屬性設定

Attack->Belone = Belone;

Attack->Target = Target;

Attack->Damage = Damage;

Attack->SP\_Damege = 0;

Attack->HitVelocity\_X = HitVelocity\_X;

Attack->HitVelocity\_Y = HitVelocity\_Y;

Attack->BitmapisRight = Belone->IsRight;

if (Attack->BitmapisRight)

Attack->Rect.X = XR;

else

Attack->Rect.X = XL;

Attack->Rect.Y = Y;

Attack->Velocity\_X = VX;

Attack->Velocity\_Y = VY;

Attack->HitTime = HitTime;

Attack->MaxAliveTime = MaxAliveTime;

Attack->Attributes = -1;

Attack->CanCombo = false;

Attack->Drawable = true;

Attack->Replay = true;

Attack->HitNoon = HitNoon;

Attack->HitBreak = false;

Attack->CanHitFly = false;

Attack->HitEffect = HitEffect;

Attack->HitSound = HitSound;

Attack->visable = true;

Attack->IsHited = false;

Attack->GravityEffect = false;

Attack->AutoPlayTimer = 0;

Attack->Step = 0;

Attack->AliveTimer = 0;

Attack->Timer1 = 0;

Attack->Timer2 = 0;

Attack->ComboTimer = 0;

Attack->Mass = Mass;

Attack->CanBeDisappear = CanBeDisappear;

Attack->CanCrackOther = CanCrackOther;

Attack->UseRectCollision = false;

//初次更新

Attack->OnUpdate(Belone->GetName() + "\\Attacks", Camera);

}

void AttackManager::DrawAllAttacks(int i)

{

map<string, AttackObj>::iterator iter;

for (iter = AttackObjects.begin(); iter != AttackObjects.end(); iter++)

{

if (iter->second.Drawable&&iter->second.visable)

{

iter->second.DisplayBitmap->Draw(i, iter->second.drawlayer);

}

iter->second.Effects.DrawAllEffection(i);

}

}

void AttackManager::InsertAttacks(string BeloneName, string name, int maxstep, int drawlayer, double pre, int category, COLORREF color, CameraPosition Camera)

{

if (category == 0) { AttackObjects.insert(std::pair<string, AttackObj>(name, AttackObj(name, 0, 0, false, true, true))); }

AttackObjects[name].SetName(name);

AttackObjects[name].AutoLoadBitmaps(BeloneName + "\\Attacks", name, maxstep + 1, pre, true, color);

AttackObjects[name].drawlayer = drawlayer;

AttackObjects[name].Effects.AutoLoadEffections(color);

AttackObjects[name].OnUpdate(BeloneName + "\\Attacks", Camera);

}

void AttackManager::InsertAttacks(string BeloneName, string name, int maxstep, int drawlayer, double pre, int category, int current, COLORREF color, CameraPosition Camera)

{

if (category == 0) { AttackObjects.insert(std::pair<string, AttackObj>(name + "\_" + IntToString(0), AttackObj(name, 0, 0, false, true, true))); }

AttackObjects[name + "\_" + IntToString(0)].SetName(name);

AttackObjects[name + "\_" + IntToString(0)].AutoLoadBitmaps(BeloneName + "\\Attacks", name, maxstep + 1, pre, true, color);

AttackObjects[name + "\_" + IntToString(0)].drawlayer = drawlayer;

AttackObjects[name + "\_" + IntToString(0)].Effects.AutoLoadEffections(color);

AttackObjects[name + "\_" + IntToString(0)].OnUpdate(BeloneName + "\\Attacks", Camera);

for (int i = 1; i < current; i++)

{

if (category == 0) { AttackObjects.insert(std::pair<string, AttackObj>(name + "\_" + IntToString(i), AttackObj(name, 0, 0, false, true, true))); }

AttackObjects[name + "\_" + IntToString(i)].SetName(name);

AttackObjects[name + "\_" + IntToString(i)].AutoLoadBitmaps(BeloneName + "\\Attacks", name, maxstep + 1, pre, false, color);

AttackObjects[name + "\_" + IntToString(i)].drawlayer = drawlayer;

AttackObjects[name + "\_" + IntToString(i)].Effects.AutoLoadEffections(color);

map<string, BitmapPicture>::iterator Iter;

map<string, BitmapPicture>::iterator Iter2 = AttackObjects[name + "\_" + IntToString(0)].BitmapPictures.begin();

for (Iter = AttackObjects[name + "\_" + IntToString(i)].BitmapPictures.begin(); Iter != AttackObjects[name + "\_" + IntToString(i)].BitmapPictures.end(); Iter++)

{

Iter->second.CanPixelCollision = true;

Iter->second.EffectRect = Iter2->second.EffectRect;

Iter2++;

}

AttackObjects[name + "\_" + IntToString(i)].OnUpdate(BeloneName + "\\Attacks", Camera);

}

}

}

Characters.h

#pragma once

#include "Keycode.h"

#include "KeyBoardState.h"

#include "CollisionSensor.h"

#include "TypeConverter.h"

#include "WKAudio.h"

#include "AttackObj.h"

#include "EffectSprite.h"

using namespace std;

using namespace WKAudio\_namespace;

using namespace CollisionSensor\_namespace;

using namespace TypeConverter\_namespace;

namespace game\_framework

{

#define CanToNormalAttack2 if (CanControl&&Button\_now.button\_Attack&&Button\_now.button\_Up == false && Button\_now.button\_Down == false&&Button\_last.button\_Attack == false && OnGround){GotoNormalAttack2(GPP);}

#define CanToNormalAttack3 if (CanControl&&Button\_now.button\_Attack&&Button\_now.button\_Up == false && Button\_now.button\_Down == false&&Button\_last.button\_Attack == false && OnGround){GotoNormalAttack3(GPP);}

#pragma region 火柴人

class Matchstick :public BattlePlayer

{

#define Matchstick\_HP\_Max 1000

#define Matchstick\_SP\_Max 100

#define Matchstick\_Rush\_Cost 8

#define Matchstick\_RunSpeed 7.5

#define Matchstick\_StandbySPincrements 0.3

#define Matchstick\_RunningSPincrements 0.15

#define Matchstick\_ChargeSPincrements 13

#define Matchstick\_NormalAttack1\_Cost 2.5

#define Matchstick\_NormalAttack1\_Damage 30

#define Matchstick\_NormalAttack2\_Cost 2.5

#define Matchstick\_NormalAttack2\_Damage 30

#define Matchstick\_NormalAttack3\_Cost 5

#define Matchstick\_NormalAttack3\_Damage 70

#define Matchstick\_AirAttack1\_Cost 2.5

#define Matchstick\_AirAttack1\_Damage 30

#define Matchstick\_AirAttack2\_Cost 5

#define Matchstick\_AirAttack2\_Damage 70

#define Matchstick\_UpAttack\_Cost 4

#define Matchstick\_UpAttack\_Damage 50

#define Matchstick\_UpSkill\_Cost 8

#define Matchstick\_UpSkill\_Damage 15

#define Matchstick\_DownAttack\_Cost 8

#define Matchstick\_DownAttack\_Damage 80

#define Matchstick\_RushAttack\_Cost 4

#define Matchstick\_RushAttack\_Damage 12

#define Matchstick\_RushSkill\_Cost 8

#define Matchstick\_RushSkill\_Damage 20

#define Matchstick\_AirUpAttack\_Cost 4

#define Matchstick\_AirUpAttack\_Damage 50

#define Matchstick\_AirDownAttack\_Cost 4

#define Matchstick\_AirDownAttack\_Damage 50

#define Matchstick\_Skill1\_Cost 4

#define Matchstick\_Skill1\_Damage 25

#define Matchstick\_UltimateSkill\_Cost 0

#define Matchstick\_UltimateSkill\_Damage1 5

#define Matchstick\_UltimateSkill\_Damage2 200

public:

Matchstick(int);

~Matchstick();

void Restvalues(int number);

virtual void AutoLoadBitmaps(GPH);//依照各自角色讀檔

virtual void AutoLoadAttacks(GPH);

virtual void OnUpdate(GPH);//更新函式，且隨著視角移動

//衝刺覆寫

virtual void GotoRush(GPH);

virtual void OnRush(GPH);

virtual void GotoNormalAttack1(GPH);

virtual void OnNormalAttack1(GPH);

virtual void GotoNormalAttack2(GPH);

virtual void OnNormalAttack2(GPH);

virtual void GotoNormalAttack3(GPH);

virtual void OnNormalAttack3(GPH);

virtual void GotoSkill1(GPH);

virtual void OnSkill1(GPH);

virtual void GotoAirAttack1(GPH);

virtual void OnAirAttack1(GPH);

virtual void GotoAirAttack2(GPH);

virtual void OnAirAttack2(GPH);

virtual void GotoUpAttack(GPH);

virtual void OnUpAttack(GPH);

virtual void GotoDownAttack(GPH);

virtual void OnDownAttack(GPH);

virtual void GotoRushAttack(GPH);

virtual void OnRushAttack(GPH);

virtual void GotoAirUpAttack(GPH);

virtual void OnAirUpAttack(GPH);

virtual void GotoAirDownAttack(GPH);

virtual void OnAirDownAttack(GPH);

virtual void GotoUpSkill(GPH);

virtual void OnUpSkill(GPH);

virtual void GotoDownSkill(GPH);

virtual void OnDownSkill(GPH);

virtual void GotoRushSkill(GPH);

virtual void OnRushSkill(GPH);

virtual void GotoAirUpSkill(GPH);

virtual void OnAirUpSkill(GPH);

virtual void GotoAirDownSkill(GPH);

virtual void OnAirDownSkill(GPH);

virtual void GotoUltimateSkill(GPH);

virtual void OnUltimateSkill(GPH);

//Timer及次數控制參數------------------------------------------------------------------------------------------------------

double NormalAttack1Timer = 0;

double UltraSkillTimer1 = 0;

double Shot1Timer = 0;

int Shot1Current = 0;

int UpSkillCurrent = 0;

bool IsRushAttack = false;

double UltraSkillcostSP = 0;

};

#pragma endregion

#pragma region Rina

class Rina :public BattlePlayer

{

#define Rina\_HP\_Max 1000

#define Rina\_SP\_Max 100

#define Rina\_Rush\_Cost 8

#define Rina\_RunSpeed 8

#define Rina\_StandbySPincrements 0.3

#define Rina\_RunningSPincrements 0.15

#define Rina\_ChargeSPincrements 13

#define Rina\_NormalAttack1\_Cost 2.5

#define Rina\_NormalAttack1\_Damage 30

#define Rina\_NormalAttack2\_Cost 2.5

#define Rina\_NormalAttack2\_Damage 30

#define Rina\_NormalAttack3\_Cost 5

#define Rina\_NormalAttack3\_Damage 70

#define Rina\_AirAttack1\_Cost 2.5

#define Rina\_AirAttack1\_Damage 30

#define Rina\_AirAttack2\_Cost 5

#define Rina\_AirAttack2\_Damage 70

#define Rina\_UpAttack\_Cost 4

#define Rina\_UpAttack\_Damage 50

#define Rina\_UpSkill\_Cost 4

#define Rina\_UpSkill\_Damage 20

#define Rina\_DownSkill\_Cost 15

#define Rina\_DownSkill\_Damage 20

#define Rina\_DownAttack\_Cost 8

#define Rina\_DownAttack\_Damage 80

#define Rina\_RushAttack\_Cost 5

#define Rina\_RushAttack\_Damage 20

#define Rina\_RushSkill\_Cost 8

#define Rina\_RushSkill\_Damage 20

#define Rina\_AirUpAttack\_Cost 4

#define Rina\_AirUpAttack\_Damage 50

#define Rina\_AirDownAttack\_Cost 10

#define Rina\_AirDownAttack\_Damage 50

#define Rina\_Skill1\_Cost 4

#define Rina\_Skill1\_Damage 25

#define Rina\_UltimateSkill\_Cost 0

#define Rina\_UltimateSkill\_Damage 300

public:

Rina(int);

~Rina();

void Restvalues(int number);

virtual void AutoLoadBitmaps(GPH);//依照各自角色讀檔

virtual void AutoLoadAttacks(GPH);

virtual void OnUpdate(GPH);//更新函式，且隨著視角移動

//衝刺覆寫

virtual void GotoRush(GPH);

virtual void OnRush(GPH);

//跑步覆寫

virtual void GotoRunning(GPH);

virtual void OnRunning(GPH);

//跳躍覆寫

virtual void GotoJump(GPH);

virtual void OnJump(GPH);

//練氣覆寫

virtual void GotoCharge(GPH);

virtual void OnCharge(GPH);

virtual void GotoNormalAttack1(GPH);

virtual void OnNormalAttack1(GPH);

virtual void GotoNormalAttack2(GPH);

virtual void OnNormalAttack2(GPH);

virtual void GotoNormalAttack3(GPH);

virtual void OnNormalAttack3(GPH);

virtual void GotoSkill1(GPH);

virtual void OnSkill1(GPH);

virtual void GotoAirAttack1(GPH);

virtual void OnAirAttack1(GPH);

virtual void GotoAirAttack2(GPH);

virtual void OnAirAttack2(GPH);

virtual void GotoUpAttack(GPH);

virtual void OnUpAttack(GPH);

virtual void GotoDownAttack(GPH);

virtual void OnDownAttack(GPH);

virtual void GotoRushAttack(GPH);

virtual void OnRushAttack(GPH);

virtual void GotoAirUpAttack(GPH);

virtual void OnAirUpAttack(GPH);

virtual void GotoAirDownAttack(GPH);

virtual void OnAirDownAttack(GPH);

virtual void GotoUpSkill(GPH);

virtual void OnUpSkill(GPH);

virtual void GotoDownSkill(GPH);

virtual void OnDownSkill(GPH);

virtual void GotoRushSkill(GPH);

virtual void OnRushSkill(GPH);

virtual void GotoAirUpSkill(GPH);

virtual void OnAirUpSkill(GPH);

virtual void GotoAirDownSkill(GPH);

virtual void OnAirDownSkill(GPH);

virtual void GotoUltimateSkill(GPH);

virtual void OnUltimateSkill(GPH);

//Timer及次數控制參數------------------------------------------------------------------------------------------------------

double NormalAttack1Timer = 0;

double JumpTimer2 = 0;

double UltraSkillTimer1 = 0;

double Shot1Timer = 0;

int Shot1Current = 0;

int UpSkillCurrent = 0;

bool IsRushAttack = false;

double UltraSkillcostSP = 0;

double DownSkillTimer = 0;

double DownSkillTimer2 = 0;

int DownSkillCurrent = 0;

double DownSkillXpoint = 0;

double DownSkillYpoint = 0;

bool useDownSkill = false;

};

#pragma endregion

}

Characters.cpp

#pragma once

#include "stdafx.h"

#include "Resource.h"

#include <mmsystem.h>

#include <ddraw.h>

#include <windows.h>

#include <vector>

#include "audio.h"

#include "gamelib.h"

#include "Keycode.h"

#include "KeyBoardState.h"

#include "WKBitmap.h"

#include "BattlePlayer.h"

#include "CollisionSensor.h"

#include "TypeConverter.h"

#include "WKAudio.h"

#include "AttackObj.h"

#include "Characters.h"

#include "EffectSprite.h"

#include "FunctionUser.h"

using namespace std;

using namespace WKAudio\_namespace;

using namespace CollisionSensor\_namespace;

using namespace TypeConverter\_namespace;

using namespace FunctionUser\_namespace;

namespace game\_framework

{

#pragma region 新增動作教學

/\*

如何新增一個角色動作

1.準備好圖片，圖片命名規則(動作向右):角色名稱\動作名稱\_Step.bmp

※全部準備好後，準備左右相反的圖片，並且檔名為:角色名稱\動作名稱\_Step\_L.bmp

2.先到void Matchstick::AutoLoadBitmaps(GPH)裡面新增動作→InsertAction("動作名稱", 動作最大數, color);

3.如果有攻擊物件，至void Matchstick::AutoLoadAttacks(GPH)新增

Attacks.InsertAttacks(GetName(), "動作名稱", 攻擊物件最大Step數, 繪製圖層, 圖片變化速度, 攻擊種類, color, Camera);

4.在void Matchstick::OnUpdate(GPH)中新增相對應的函式

5.編寫Goto函式，主要是用來控制進入該動作時的初始化變數。

6.到BattlePlayer編寫CanTo。

7.編寫On函式，動作的實體，以一個IF判斷是否在這個動作裡面，以STEP和TIMER控制動作流程，最後給予一段僵直時間讓角色可以到其他可行的動作(CanTo)。

\*/

#pragma endregion

#pragma region 火柴人

Matchstick::Matchstick(int number) :BattlePlayer()

{

Restvalues(number);

}

Matchstick::~Matchstick()

{

}

void Matchstick::Restvalues(int number)

{

//能力值變數

HP\_Max = Matchstick\_HP\_Max;//最大生命值

SP\_Max = Matchstick\_SP\_Max;//最大氣力

recovery = 0;

Rush\_cost = Matchstick\_Rush\_Cost;//衝刺消耗量

StandbySPincrements = Matchstick\_StandbySPincrements;

RunningSPincrements = Matchstick\_RunningSPincrements;

RunSpeed = Matchstick\_RunSpeed;

ChargeSPincrements = Matchstick\_ChargeSPincrements;

//現狀變數

SetName("Matchstick");

PlayerNumber = number;//玩家編號

if (number == 1)

{

IsRight = true;

}

else if (number == 2)

{

IsRight = false;

}

HP = HP\_Max;//當前生命

SP = SP\_Max / 2;//當前氣力

CanControl = false;//可以控制

Invincible = false;//無敵狀態

Action = "待機";//動作狀態

Step = 0;//當前步驟數

visable = true;//是否可見

CanPixelCollision = true;//是否使用像素碰撞，所有動作分割都會套用

InSideCamera = true;//是否受到鏡頭影響

Velocity\_X = 0;//X速度

Velocity\_Y = 0;//Y速度

Acceleration\_X = 0;//X加速度

Acceleration\_Y = 0;//Y加速度

Throughing = false;

HitFly = false;

BreakPoint = 0;

Acceleration\_gravity = 0.5;

}

//讀檔在此

void Matchstick::AutoLoadBitmaps(GPH)

{

//有效判定區BitRect

BodyPicture.LoadTexture(color);

InsertAction("待機", 1, color);

InsertAction("移動", 4, color);

InsertAction("衝刺", 1, color);

InsertAction("跳躍", 4, color);

InsertAction("防禦", 0, color);

InsertAction("練氣", 3, color);

InsertAction("受傷", 2, color);

InsertAction("防禦受傷", 0, color);

InsertAction("普攻1", 4, color);

InsertAction("普攻2", 4, color);

InsertAction("普攻3", 4, color);

InsertAction("上普", 4, color);

InsertAction("上特技", 3, color);

InsertAction("下普", 5, color);

InsertAction("特技1", 5, color);

InsertAction("空普1", 4, color);

InsertAction("空普2", 4, color);

InsertAction("空下普", 7, color);

InsertAction("空上普", 7, color);

InsertAction("衝刺普", 0, color);

InsertAction("衝刺特技", 1, color);

InsertAction("大絕", 6, color);

//LoadEffects

Effects.AutoLoadEffections(color);

//LoadAttacks

AutoLoadAttacks(GPP);

AnimationUpdate(Camera);

}

//註冊攻擊物件

void Matchstick::AutoLoadAttacks(GPH)

{

Attacks.InsertAttacks(GetName(), "Normal1", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal2", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal3", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal4", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal5", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal6", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal7", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "RushSkill", 2, 5, 8, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "UpSkill", 2, 5, 8, 0, 1, color, Camera);

Attacks.InsertAttacks(GetName(), "Counterattact", 4, 5, 20, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Skill1", 2, 5, 20, 0, 5, color, Camera);//多一個參數是具有編號的

}

//註冊動作在此

void Matchstick::OnUpdate(GPH)

{

InputJudge(KeyState\_now, KeyState\_last);

CheckHit(GPP);

OnStandby(GPP);

OnRunning(GPP);

OnRush(GPP);

OnJump(GPP);

OnGuard(GPP);

OnCharge(GPP);

OnHit(GPP);

OnHitGuard(GPP);

OnNormalAttack1(GPP);

OnNormalAttack2(GPP);

OnNormalAttack3(GPP);

OnSkill1(GPP);

OnAirAttack1(GPP);

OnAirAttack2(GPP);

OnAirDownAttack(GPP);

OnAirUpAttack(GPP);

OnDownAttack(GPP);

OnUpAttack(GPP);

OnRushAttack(GPP);

OnRushSkill(GPP);

OnUpSkill(GPP);

OnUltimateSkill(GPP);

//更新所有Effect的動作

map<string, BitmapAnimation>::iterator Iter\_Effect;

for (Iter\_Effect = Effects.Content.begin(); Iter\_Effect != Effects.Content.end(); Iter\_Effect++)

Effects.EffectAutoUpdate(&(Iter\_Effect->second), (int)(((Iter\_Effect->second).PreAutoFrequence)), (Iter\_Effect->second).loop, Camera);

//更新所有Attacks的動作

map<string, AttackObj>::iterator Iter\_Attack;

for (Iter\_Attack = Attacks.AttackObjects.begin(); Iter\_Attack != Attacks.AttackObjects.end(); Iter\_Attack++)

Attacks.AttackAutoUpdate(&(Iter\_Attack->second), GetName(), (int)(((Iter\_Attack->second).PreAutoFrequence)), (Iter\_Attack->second).Replay, Camera);

this->PhysicalMovement(GPP);

AnimationUpdate(Camera);

//雜項

ProduceBreakPoint(GPP);

ProduceRecovery(GPP);

CheckPerfectBlock(GPP);

#pragma region 例外狀況

if (!(Action == "大絕"))

{

Effects.Content["OraOraFire"].visable = false;

}

if (!(Action == "衝刺特技"))

{

Attacks.AttackObjects["RushSkill"].visable = false;

}

#pragma endregion

}

void Matchstick::GotoRush(GPH)

{

if (SP >= Rush\_cost)

{

GainSP(-Rush\_cost);

Action = "衝刺";

Step = 0;

RushTimer = 0;

}

}

void Matchstick::OnRush(GPH)

{

if (Action == "衝刺")

{

#pragma region 衝刺主程序

RushTimer += TIMER\_TICK\_MILLIDECOND;

if (Step == 1)

Velocity\_Y = 0;

if (RushTimer < 40 && Step == 0)

Velocity\_X = 0;

if (RushTimer >= 40 && Step == 0)

{

RushTimer = 0;

Step = 1;

Effects.BootEffect(&Effects.Content["Airboost"], Camera, Rect.X, Rect.X + 30, Rect.Y - 30, 0, 0, false, IsRight);

Throughing = true;

PlaySounds(Sounds.Rush, false);

if (IsRight)

{

Velocity\_X = 38;

Acceleration\_X = -3.5;

Invincible = true;

}

else

{

Velocity\_X = -38;

Acceleration\_X = +3.5;

Invincible = true;

}

}

else if (RushTimer <= 80 && Step == 1)//煞車

{

if (IsRight&&Velocity\_X < 0)

Velocity\_X = 0;

else if (IsRight == false && Velocity\_X > 0)

Velocity\_X = 0;

CanToRushAttack;

CanToRushSkill;

}

#pragma endregion

//正常結束

if (RushTimer > 80 && Step == 1)

{

RushTimer = 0;

Throughing = false;

Invincible = false;

Velocity\_X = 0;

Acceleration\_X = 0;

#pragma region 回收特效

Effects.Content["AirBoost"].Step = 0;

Effects.Content["AirBoost"].visable = false;

Effects.Content["AirBoost"].OnUpdate("Effects", Camera);

#pragma endregion

#pragma region 判斷應該回到哪個動作

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

#pragma endregion

}

}

}

void Matchstick::GotoNormalAttack1(GPH)

{

if (SP >= Matchstick\_NormalAttack1\_Cost)

{

GainSP(-Matchstick\_NormalAttack1\_Cost);

Action = "普攻1";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnNormalAttack1(GPH)

{

if (Action == "普攻1")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 16 && Step <= 2)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

//出拳

if (Step >= 3)

{

Velocity\_X += Ahead(3.5);

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Matchstick\_NormalAttack1\_Damage,

2, 2, Rect.X + 108, Rect.X - 2, Rect.Y + 52, 0, 0,

120, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

else if (NormalAttack1Timer >= 40 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 4)

{

//到別的動作

CanToNormalAttack2;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoNormalAttack2(GPH)

{

if (SP >= Matchstick\_NormalAttack2\_Cost)

{

GainSP(-Matchstick\_NormalAttack2\_Cost);

Action = "普攻2";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnNormalAttack2(GPH)

{

if (Action == "普攻2")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 16 && Step <= 2)

{

NormalAttack1Timer = 0;

Step += 1;

if (Step >= 3)

{

Velocity\_X += Ahead(3.5);

#pragma region 產生攻擊物件

//出拳

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Matchstick\_NormalAttack2\_Damage,

3.5, 2, Rect.X + 108, Rect.X - 2, Rect.Y + 52, 0, 0,

120, 30, "PunchHit", Sounds.NormalHit, Camera);

#pragma endregion

}

}

else if (NormalAttack1Timer >= 50 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 4)

{

//到別的動作

CanToNormalAttack3;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoNormalAttack3(GPH)

{

if (SP >= Matchstick\_NormalAttack3\_Cost)

{

GainSP(-Matchstick\_NormalAttack3\_Cost);

Velocity\_X += Ahead(5);

Velocity\_Y -= 5;

Action = "普攻3";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnNormalAttack3(GPH)

{

if (Action == "普攻3")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

ProduceFriction(0.2, 0.25);

if (NormalAttack1Timer >= 84 && Step <= 2)

{

Step += 1;

if (Step >= 3)

{

#pragma region 產生攻擊物件

//基本設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal2"]), this, Enemy,

Matchstick\_NormalAttack3\_Damage,

12, 7.5, Rect.X + 147, Rect.X - 7, Rect.Y + 25, 0, 0,

200, 30, "PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["Normal2"].CanHitFly = true;

#pragma endregion

}

}

else if (NormalAttack1Timer >= 150 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 4)

{

CanToJump;

CanToRush;

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoSkill1(GPH)

{

if (SP >= Matchstick\_Skill1\_Cost)

{

GainSP(-Matchstick\_Skill1\_Cost);

if (Velocity\_Y > 0 && OnGround == false)

Velocity\_Y = 0;

Action = "特技1";

Step = 0;

Shot1Timer = 0;

}

}

void Matchstick::OnSkill1(GPH)

{

if (Action == "特技1")

{

Shot1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

if (OnGround)

ProduceFriction(1, 1);

else

ProduceFriction(0.15, 0.75);

if (Shot1Timer >= 50 && Step == 0)

{

Step = 1;

if (Velocity\_Y > 0 && OnGround == false)

Velocity\_Y = 0;

}

if (Shot1Timer >= 16 && Step >= 1 && Step <= 3)

{

if (Shot1Current >= 5)

Shot1Current = 0;

Shot1Timer = 0;

Step += 1;

if (Step == 3)

{

#pragma region 產生攻擊物件

//出拳

Attacks.AttackReset\_Shot(&(Attacks.AttackObjects["Skill1\_" + IntToString(Shot1Current)]), this, Enemy,

Rina\_DownSkill\_Damage,

2, 3.5,

Rect.X + 75, Rect.X + 10, Rect.Y + 52, Ahead(4.5), 0,

120, 1000, 2,

true, true, true,

"PunchHit", Sounds.NormalHit, Camera);

Shot1Current += 1;

#pragma endregion

}

}

else if (Shot1Timer >= 50 && Step == 4)

{

Shot1Timer = 0;

Step = 5;

}

#pragma endregion

#pragma region 到別的動作

if (Shot1Timer < 100 && Step >= 5)

{

//到別的可能動作

if (OnGround)

{

CanToNormalAttack1;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

}

else

{

CanToAirUpAttack;

CanToAirDownAttack;

CanToAirAttack1;

CanToFastDrop;

}

CanToRush;

CanToJump;

}

else if (Shot1Timer >= 100 && Step >= 5)

{

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoAirAttack1(GPH)

{

if (SP >= Matchstick\_AirAttack1\_Cost)

{

GainSP(-Matchstick\_AirAttack1\_Cost);

Action = "空普1";

Step = 0;

if (Velocity\_Y > 0)

{

Velocity\_Y = 0;

}

NormalAttack1Timer = 0;

}

}

void Matchstick::OnAirAttack1(GPH)

{

if (Action == "空普1")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 左右移動

if (CanControl&&Button\_now.button\_Right == false && CanControl&&Button\_now.button\_Left == false)

{

ProduceFriction(0.15, 1);

}

else if (CanControl&&Button\_now.button\_Right == true)

{

IsRight = true;

RunAhead(0.5, RunSpeed / 2);

}

else if (CanControl&&Button\_now.button\_Left == true)

{

IsRight = false;

RunAhead(0.5, RunSpeed / 2);

}

#pragma endregion

#pragma region 動作主體

if (NormalAttack1Timer >= 50 && Step == 0)

{

Step = 1;

}

if (NormalAttack1Timer >= 16 && Step >= 1 && Step <= 3)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

if (Step == 3)

{

//出拳

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal3"]), this, Enemy,

Matchstick\_AirAttack1\_Damage,

2, 5, Rect.X + 108, Rect.X - 2, Rect.Y + 90, Velocity\_X / 3, 0,

150, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

else if (NormalAttack1Timer >= 125 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 125 && Step >= 4)

{

//到別的可能動作

CanToAirAttack2;

CanToSkill1;

CanToRush;

CanToAirDownAttack;

CanToAirUpAttack;

CanToJump;

CanToFastDrop;

}

else if (NormalAttack1Timer >= 125 && Step >= 4)

{

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoAirAttack2(GPH)

{

if (SP >= Matchstick\_AirAttack2\_Cost)

{

GainSP(-Matchstick\_AirAttack2\_Cost);

Action = "空普2";

Step = 0;

Velocity\_X += Ahead(4);

Velocity\_Y = -3.5;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnAirAttack2(GPH)

{

if (Action == "空普2")

{

ProduceFriction(0.1, 0.15);

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

if (NormalAttack1Timer >= 84 && Step <= 2)

{

Step += 1;

if (Step >= 3)

{

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal2"]), this, Enemy,

Matchstick\_AirAttack2\_Damage,

12, 7.5, Rect.X + 142, Rect.X, Rect.Y + 60, Velocity\_X / 3, 0,

250, 50, "PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["Normal2"].CanHitFly = true;

}

}

else if (NormalAttack1Timer >= 150 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

else if (NormalAttack1Timer < 100 && Step >= 4)

{

//到別的動作

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

}

}

void Matchstick::GotoUpAttack(GPH)

{

if (SP >= Matchstick\_UpAttack\_Cost)

{

GainSP(-Matchstick\_UpAttack\_Cost);

Action = "上普";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnUpAttack(GPH)

{

if (Action == "上普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 80 && Step == 0)

{

NormalAttack1Timer = 0;

Step = 1;

}

else if (NormalAttack1Timer >= 16 && Step >= 1 && Step <= 2)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

//出拳

if (Step == 3)

{

Velocity\_X += Ahead(7);

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal2"]), this, Enemy,

Matchstick\_UpAttack\_Damage,

2, 12, Rect.X + 147, Rect.X - 7, Rect.Y + 25, 0, 0,

220, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

else if (NormalAttack1Timer >= 40 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 4)

{

//到別的動作

CanToNormalAttack1;

CanToSkill1;

CanToJump;

CanToRush;

CanToDownAttack;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoDownAttack(GPH)

{

if (SP >= Matchstick\_DownAttack\_Cost)

{

GainSP(-Matchstick\_DownAttack\_Cost);

Action = "下普";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnDownAttack(GPH)

{

if (Action == "下普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 110 && Step == 0)

{

NormalAttack1Timer = 0;

Step = 1;

}

else if (NormalAttack1Timer >= 16 && Step >= 1 && Step <= 3)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

//出拳

if (Step >= 4)

{

Velocity\_X += Ahead(3.5);

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Matchstick\_DownAttack\_Damage,

2, 3, Rect.X + 111, Rect.X, Rect.Y + 52, 0, 0,

200, 30, "PunchHit", Sounds.NormalHit, Camera);

Attacks.AttackObjects["Normal1"].HitBreak = true;

}

#pragma endregion

}

else if (NormalAttack1Timer >= 40 && Step == 4)

{

NormalAttack1Timer = 0;

Step = 5;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 5)

{

//到別的動作

CanToNormalAttack1;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 5)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoRushAttack(GPH)

{

if (SP >= Matchstick\_RushAttack\_Cost)

{

GainSP(-Matchstick\_RushAttack\_Cost);

if (Velocity\_Y > 0 && OnGround == false)

Velocity\_Y = 0;

RushTimer = 0;

IsRushAttack = false;

Throughing = false;

Invincible = false;

Velocity\_X = 0;

Acceleration\_X = 0;

Action = "衝刺普";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnRushAttack(GPH)

{

if (Action == "衝刺普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

if (NormalAttack1Timer <= 140 && Step == 0)

{

Velocity\_Y = 0;

Acceleration\_X = Ahead(1.2);

if (abs(Velocity\_X) > 12)

{

Velocity\_X = Ahead(12);

}

#pragma region 更新攻擊物件位置

if (IsRight)

Attacks.AttackObjects["Normal6"].Rect.X = Rect.X + 60;

else

Attacks.AttackObjects["Normal6"].Rect.X = Rect.X - 10;

Attacks.AttackObjects["Normal6"].Rect.Y = Rect.Y + 35;

if (Attacks.AttackObjects["Normal6"].IsHited&&Attacks.AttackObjects["Normal6"].ComboTimer > TIMER\_TICK\_MILLIDECOND \* 2)

{

Attacks.AttackObjects["Normal6"].ComboTimer = 0;

Attacks.AttackObjects["Normal6"].IsHited = false;

}

#pragma endregion

#pragma region 產生攻擊物件

if (IsRushAttack == false)

{

IsRushAttack = true;

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal6"]), this, Enemy,

Matchstick\_RushAttack\_Damage,

8.5, 1, Rect.X + 98, 0, 0, 0, 0,

50, 100, "PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["Normal6"].CanCombo = true;//可連擊

Attacks.AttackObjects["Normal6"].HitNoon = false;

}

#pragma endregion

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer > 140 && Step == 0)

{

Acceleration\_X = 0;

Velocity\_X = 0;

#pragma region 判斷應該回到哪個動作

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

#pragma endregion

}

#pragma endregion

}

}

void Matchstick::GotoAirUpAttack(GPH)

{

if (SP >= Matchstick\_AirUpAttack\_Cost)

{

GainSP(-Matchstick\_AirUpAttack\_Cost);

Action = "空上普";

Step = 0;

if (Velocity\_Y > 0)

Velocity\_Y = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnAirUpAttack(GPH)

{

if (Action == "空上普")

{

#pragma region 左右移動

if (CanControl&&Button\_now.button\_Right == false && CanControl&&Button\_now.button\_Left == false)

{

ProduceFriction(0.15, 1);

}

else if (CanControl&&Button\_now.button\_Right == true)

{

IsRight = true;

RunAhead(0.5, RunSpeed / 2);

}

else if (CanControl&&Button\_now.button\_Left == true)

{

IsRight = false;

RunAhead(0.5, RunSpeed / 2);

}

#pragma endregion

#pragma region 動作主體

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

if (NormalAttack1Timer >= 100 && Step == 0)

{

Step = 1;

if (Velocity\_Y > 1)

Velocity\_Y = 1;

}

if (NormalAttack1Timer >= 16 && Step >= 1 && Step <= 3)

{

NormalAttack1Timer = 0;

Step += 1;

Velocity\_Y = -6;

#pragma region 產生攻擊物件

if (Step == 2)

{

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal5"]), this, Enemy,

Matchstick\_AirUpAttack\_Damage,

2.5, 12, Rect.X + 20, Rect.X, Rect.Y, Velocity\_X / 4, 0,

250, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

else if (NormalAttack1Timer >= 16 && Step >= 4 && Step <= 6)

{

if (Velocity\_Y > 2)

Velocity\_Y = 2;

NormalAttack1Timer = 0;

Step += 1;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 7)

{

if (Velocity\_Y > 1)

Velocity\_Y = 1;

//到別的可能動作

CanToSkill1;

CanToRush;

CanToJump;

CanToAirAttack1;

CanToAirDownAttack;

CanToFastDrop;

}

else if (NormalAttack1Timer >= 100 && Step >= 7)

{

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoAirDownAttack(GPH)

{

if (SP >= Matchstick\_AirDownAttack\_Cost)

{

GainSP(-Matchstick\_AirDownAttack\_Cost);

Action = "空下普";

Step = 0;

if (Velocity\_Y > 0)

Velocity\_Y = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnAirDownAttack(GPH)

{

if (Action == "空下普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 左右移動

if (CanControl&&Button\_now.button\_Right == false && CanControl&&Button\_now.button\_Left == false)

{

ProduceFriction(0.15, 1);

}

else if (CanControl&&Button\_now.button\_Right == true)

{

IsRight = true;

RunAhead(0.5, RunSpeed / 2);

}

else if (CanControl&&Button\_now.button\_Left == true)

{

IsRight = false;

RunAhead(0.5, RunSpeed / 2);

}

#pragma endregion

#pragma region 動作主體

if (NormalAttack1Timer >= 75 && Step == 0)

{

if (Velocity\_Y > 1)

Velocity\_Y = 1;

Step = 1;

}

if (NormalAttack1Timer >= 8 && Step >= 1 && Step <= 3)

{

NormalAttack1Timer = 0;

Step += 1;

}

else if (NormalAttack1Timer >= 20 && Step >= 4 && Step <= 6)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

if (Step == 5)

{

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal4"]), this, Enemy,

Matchstick\_AirDownAttack\_Damage,

2.5, -16, Rect.X + 90, Rect.X, Rect.Y + 20, Velocity\_X / 2, Velocity\_Y / 3,

200, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer > 50 && NormalAttack1Timer < 120 && Step >= 7)

{

if (Velocity\_Y > 1)

Velocity\_Y = 1;

//到別的可能動作

CanToSkill1;

CanToRush;

CanToJump;

CanToFastDrop;

}

else if (NormalAttack1Timer >= 120 && Step >= 7)

{

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoUpSkill(GPH)

{

if (SP >= Matchstick\_UpSkill\_Cost)

{

GainSP(-Matchstick\_UpSkill\_Cost);

Action = "上特技";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnUpSkill(GPH)

{

if (Action == "上特技")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 150 && Step == 0)

{

NormalAttack1Timer = 0;

Step = 1;

}

else if (NormalAttack1Timer >= 16 && Step >= 1 && Step <= 2)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

//出拳

if (Step == 3)

{

PlaySounds(Sounds.Fire1, false);

//基礎設定

Attacks.AttackReset\_Shot(&(Attacks.AttackObjects["UpSkill\_" + IntToString(UpSkillCurrent)]), this, Enemy,

Matchstick\_UpSkill\_Damage,

1, 5,

Rect.X, Rect.X, Rect.Y + 17, 0, -9,

50, 400, 5,

true, true, true,

"PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["UpSkill\_" + IntToString(UpSkillCurrent)].Drawable = true;

Attacks.AttackObjects["UpSkill\_" + IntToString(UpSkillCurrent)].HitBreak = false;

Attacks.AttackObjects["UpSkill\_" + IntToString(UpSkillCurrent)].HitNoon = false;

Attacks.AttackObjects["UpSkill\_" + IntToString(UpSkillCurrent)].CanCombo = true;

UpSkillCurrent = 0;

}

#pragma endregion

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer >= 40 && NormalAttack1Timer < 100 && Step >= 3)

{

//到別的動作

CanToNormalAttack1;

CanToSkill1;

CanToJump;

CanToRush;

CanToDownAttack;

}

else if (NormalAttack1Timer >= 100 && Step >= 3)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Matchstick::GotoDownSkill(GPH)

{

}

void Matchstick::OnDownSkill(GPH)

{

}

void Matchstick::GotoRushSkill(GPH)

{

if (SP >= Matchstick\_RushSkill\_Cost)

{

GainSP(-Matchstick\_RushSkill\_Cost);

if (Velocity\_Y > 0 && OnGround == false)

Velocity\_Y = 0;

RushTimer = 0;

IsRushAttack = false;

Throughing = true;

Invincible = false;

Velocity\_X = 0;

Acceleration\_X = 0;

Action = "衝刺特技";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Matchstick::OnRushSkill(GPH)

{

if (Action == "衝刺特技")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

if (NormalAttack1Timer <= 150 && Step == 0)

{

Velocity\_Y = -4;

Velocity\_X = Ahead(2.5);

}

else if (NormalAttack1Timer > 150 && Step == 0)

{

Step = 1;

NormalAttack1Timer = 0;

}

if (NormalAttack1Timer <= 200 && Step == 1)

{

Velocity\_Y = 0;

Acceleration\_X = Ahead(4);

if (abs(Velocity\_X) > 15)

{

Velocity\_X = Ahead(15);

}

#pragma region 更新攻擊物件位置

if (IsRight)

Attacks.AttackObjects["RushSkill"].Rect.X = Rect.X - 47;

else

Attacks.AttackObjects["RushSkill"].Rect.X = Rect.X - 15;

Attacks.AttackObjects["RushSkill"].Rect.Y = Rect.Y + 0;

if (Attacks.AttackObjects["RushSkill"].IsHited&&Attacks.AttackObjects["RushSkill"].ComboTimer > TIMER\_TICK\_MILLIDECOND \* 4)

{

Attacks.AttackObjects["RushSkill"].ComboTimer = 0;

Attacks.AttackObjects["RushSkill"].IsHited = false;

}

#pragma endregion

#pragma region 產生攻擊物件

if (IsRushAttack == false)

{

IsRushAttack = true;

PlaySounds(Sounds.Fire1, false);

//基礎設定

Attacks.AttackReset\_Shot(&(Attacks.AttackObjects["RushSkill"]), this, Enemy,

Matchstick\_RushSkill\_Damage,

12, 5,

Rect.X - 47, Rect.X - 15, Rect.Y, 0, 0,

250, 200, 5,

true, true, true,

"PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["RushSkill"].Drawable = true;

Attacks.AttackObjects["RushSkill"].HitBreak = false;

Attacks.AttackObjects["RushSkill"].HitNoon = false;

Attacks.AttackObjects["RushSkill"].CanCombo = true;

}

#pragma endregion

}

#pragma endregion

#pragma region 到別的動作

if ((NormalAttack1Timer > 200) && Step == 1)

{

Throughing = false;

Acceleration\_X = 0;

Velocity\_X /= 2;

#pragma region 判斷應該回到哪個動作

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

#pragma endregion

}

#pragma endregion

}

}

void Matchstick::GotoAirUpSkill(GPH)

{

}

void Matchstick::OnAirUpSkill(GPH)

{

}

void Matchstick::GotoAirDownSkill(GPH)

{

}

void Matchstick::OnAirDownSkill(GPH)

{

}

void Matchstick::GotoUltimateSkill(GPH)

{

if (SP > (SP\_Max / 3))

{

UltraSkillcostSP = SP - (SP\_Max / 3);

SP = 0;

Action = "大絕";

Step = 0;

NormalAttack1Timer = 0;

UltraSkillTimer1 = 0;

}

}

void Matchstick::OnUltimateSkill(GPH)

{

if (Action == "大絕")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

UltraSkillTimer1 += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 300 && Step == 0)

{

PlaySounds(Sounds.CutIn, false);

NeedCutIn = true;

NormalAttack1Timer = 0;

Step = 1;

Effects.BootEffect(&Effects.Content["Matchstick\_US"], Camera, 800, 800, 0, 0, 0, false, true);

Effects.BootEffect(&Effects.Content["OraOraFire"], Camera, Rect.X + 60, Rect.X - 60, Rect.Y, 0, 0, false, IsRight);

Effects.Content["OraOraFire"].loop = true;

}

if (Step >= 1 && Step <= 5 && NormalAttack1Timer >= 40)

{

if (IsRight)

Effects.Content["OraOraFire"].Rect.X = Rect.X + 60;

else

Effects.Content["OraOraFire"].Rect.X = Rect.X - 60;

Effects.Content["OraOraFire"].Rect.Y = Rect.Y;

Effects.Content["OraOraFire"].visable = true;

NormalAttack1Timer = 0;

Step += 1;

if (Step > 5)

{

Step = 1;

}

#pragma region 產生攻擊物件

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal7"]), this, Enemy,

Matchstick\_UltimateSkill\_Damage1\*(1 + (UltraSkillcostSP / SP\_Max)),

0, 0.75, Rect.X + 60, Rect.X - 60, Rect.Y, 0, 0,

100, 30, "PunchHit", Sounds.NormalHit2, Camera);

#pragma endregion

}

if (UltraSkillTimer1 >= 1200 && Step < 6)

{

Step = 6;

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal7"]), this, Enemy,

Matchstick\_UltimateSkill\_Damage2\*(1 + (UltraSkillcostSP / SP\_Max)),

15, 10, Rect.X + 60, Rect.X - 60, Rect.Y, 0, 0,

200, 30, "PunchHit", Sounds.Stoned, Camera);

Attacks.AttackObjects["Normal7"].CanHitFly = true;

Attacks.AttackObjects["Normal7"].HitBreak = false;

Effects.Content["OraOraFire"].visable = false;

Effects.Content["OraOraFire"].loop = false;

}

if (Step == 6 && UltraSkillTimer1 > 1700)

{

GotoStandby(GPP);

}

#pragma endregion

}

}

#pragma endregion

#pragma region Rina

Rina::Rina(int number) :BattlePlayer()

{

Restvalues(number);

}

Rina::~Rina()

{

}

void Rina::Restvalues(int number)

{

//能力值變數

HP\_Max = Rina\_HP\_Max;//最大生命值

SP\_Max = Rina\_SP\_Max;//最大氣力

recovery = 0;

Rush\_cost = Rina\_Rush\_Cost;//衝刺消耗量

StandbySPincrements = Rina\_StandbySPincrements;

RunningSPincrements = Rina\_RunningSPincrements;

RunSpeed = Rina\_RunSpeed;

ChargeSPincrements = Rina\_ChargeSPincrements;

//現狀變數

SetName("Rina");

PlayerNumber = number;//玩家編號

if (number == 1)

{

IsRight = true;

}

else if (number == 2)

{

IsRight = false;

}

HP = HP\_Max;//當前生命

SP = SP\_Max / 2;//當前氣力

CanControl = false;//可以控制

Invincible = false;//無敵狀態

Action = "待機";//動作狀態

Step = 0;//當前步驟數

visable = true;//是否可見

CanPixelCollision = true;//是否使用像素碰撞，所有動作分割都會套用

InSideCamera = true;//是否受到鏡頭影響

Velocity\_X = 0;//X速度

Velocity\_Y = 0;//Y速度

Acceleration\_X = 0;//X加速度

Acceleration\_Y = 0;//Y加速度

Throughing = false;

HitFly = false;

BreakPoint = 0;

Acceleration\_gravity = 0.5;

}

//讀檔在此

void Rina::AutoLoadBitmaps(GPH)

{

//有效判定區BitRect

BodyPicture.LoadTexture(color);

InsertAction("待機", 1, color);

InsertAction("移動", 7, color);

InsertAction("跳躍", 4, color);

InsertAction("防禦", 0, color);

InsertAction("衝刺", 2, color);

InsertAction("防禦受傷", 0, color);

InsertAction("受傷", 2, color);

InsertAction("練氣", 4, color);

InsertAction("普攻1", 4, color);

InsertAction("普攻2", 4, color);

InsertAction("普攻3", 6, color);

InsertAction("下特技", 4, color);

InsertAction("衝刺普", 2, color);

InsertAction("特技", 5, color);

InsertAction("上特技", 4, color);

InsertAction("上普", 5, color);

InsertAction("空下普", 2, color);

InsertAction("下普", 5, color);

InsertAction("大絕", 2, color);

//LoadEffects

Effects.AutoLoadEffections(color);

//LoadAttacks

AutoLoadAttacks(GPP);

AnimationUpdate(Camera);

}

//註冊攻擊物件

void Rina::AutoLoadAttacks(GPH)

{

Attacks.InsertAttacks(GetName(), "Normal1", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal2", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "Normal3", 0, 5, 16, 0, color, Camera);

Attacks.InsertAttacks(GetName(), "flashblade", 1, 5, 8, 0, 16, color, Camera);//多一個參數是具有編號的

Attacks.InsertAttacks(GetName(), "Bigflashblade", 0, 5, 8, 0, color, Camera);//多一個參數是具有編號的

Attacks.InsertAttacks(GetName(), "flashblade\_H", 1, 5, 8, 0, 4, color, Camera);//多一個參數是具有編號的

Attacks.InsertAttacks(GetName(), "Counterattact", 4, 5, 20, 0, color, Camera);

}

//註冊動作在此

void Rina::OnUpdate(GPH)

{

InputJudge(KeyState\_now, KeyState\_last);

CheckHit(GPP);

OnStandby(GPP);

OnRunning(GPP);

OnRush(GPP);

OnJump(GPP);

OnGuard(GPP);

OnCharge(GPP);

OnHit(GPP);

OnHitGuard(GPP);

OnNormalAttack1(GPP);

OnNormalAttack2(GPP);

OnNormalAttack3(GPP);

OnSkill1(GPP);

OnAirAttack1(GPP);

OnAirAttack2(GPP);

OnAirDownAttack(GPP);

OnAirUpAttack(GPP);

OnDownAttack(GPP);

OnDownSkill(GPP);

OnUpAttack(GPP);

OnRushAttack(GPP);

OnRushSkill(GPP);

OnUpSkill(GPP);

OnUltimateSkill(GPP);

//更新所有Effect的動作

map<string, BitmapAnimation>::iterator Iter\_Effect;

for (Iter\_Effect = Effects.Content.begin(); Iter\_Effect != Effects.Content.end(); Iter\_Effect++)

Effects.EffectAutoUpdate(&(Iter\_Effect->second), (int)(((Iter\_Effect->second).PreAutoFrequence)), (Iter\_Effect->second).loop, Camera);

//更新所有Attacks的動作

map<string, AttackObj>::iterator Iter\_Attack;

for (Iter\_Attack = Attacks.AttackObjects.begin(); Iter\_Attack != Attacks.AttackObjects.end(); Iter\_Attack++)

Attacks.AttackAutoUpdate(&(Iter\_Attack->second), GetName(), (int)(((Iter\_Attack->second).PreAutoFrequence)), (Iter\_Attack->second).Replay, Camera);

this->PhysicalMovement(GPP);

AnimationUpdate(Camera);

//雜項

ProduceBreakPoint(GPP);

ProduceRecovery(GPP);

CheckPerfectBlock(GPP);

#pragma region 例外狀況

#pragma region 下特技

if (useDownSkill)

{

DownSkillTimer2 += TIMER\_TICK\_MILLIDECOND;

if (DownSkillTimer2 > 50)

{

PlaySounds(Sounds.light2, false);

DownSkillTimer2 = 0;

Attacks.AttackReset\_Shot(&(Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)]), this, Enemy,

Matchstick\_UpSkill\_Damage,

0, 0,

DownSkillXpoint + 60, DownSkillXpoint + 60, DownSkillYpoint - 150, 0, 2,

20, 750, 5,

true, true, true,

"PunchHit", Sounds.SliceHit, Camera);

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].HitNoon = false;

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].CanBeDisappear = false;

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].CanCrackOther = false;

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].UseRectCollision = true;

//額外設定

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].GravityEffect = true;

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Acceleration\_gravity = 0.5;

double Positions[] = { 0,30,-30,10,0,-60,80,-40,80,-10,-70,-25,10,-80,0,80 };

if (DownSkillCurrent == 0)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[0];

else if (DownSkillCurrent % 16 == 1)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[1];

else if (DownSkillCurrent % 16 == 2)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[2];

else if (DownSkillCurrent % 16 == 3)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[3];

else if (DownSkillCurrent % 16 == 4)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[4];

else if (DownSkillCurrent % 16 == 5)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[5];

else if (DownSkillCurrent % 16 == 6)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[6];

else if (DownSkillCurrent % 16 == 7)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[7];

else if (DownSkillCurrent % 16 == 8)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[8];

else if (DownSkillCurrent % 16 == 9)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[9];

else if (DownSkillCurrent % 16 == 10)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[10];

else if (DownSkillCurrent % 16 == 11)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[11];

else if (DownSkillCurrent % 16 == 12)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[12];

else if (DownSkillCurrent % 16 == 13)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[13];

else if (DownSkillCurrent % 16 == 14)

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[14];

else if (DownSkillCurrent % 16 == 15)

{

Attacks.AttackObjects["flashblade\_" + IntToString(DownSkillCurrent)].Rect.X += Positions[15];

useDownSkill = false;

DownSkillCurrent = 0;

}

DownSkillCurrent += 1;

}

#pragma endregion

}

#pragma endregion

}

void Rina::GotoRush(GPH)

{

if (SP >= Rush\_cost)

{

GainSP(-Rush\_cost);

Action = "衝刺";

Step = 0;

RushTimer = 0;

}

}

void Rina::OnRush(GPH)

{

if (Action == "衝刺")

{

#pragma region 衝刺主程序

RushTimer += TIMER\_TICK\_MILLIDECOND;

if (Step == 1)

Velocity\_Y = 0;

if (RushTimer < 40 && Step == 0)

{

Velocity\_X = Ahead(2.5);

Velocity\_Y = -2;

}

if (RushTimer >= 40 && Step == 0)

{

RushTimer = 0;

Step = 1;

Effects.BootEffect(&Effects.Content["Airboost"], Camera, Rect.X, Rect.X + 30, Rect.Y - 30, 0, 0, false, IsRight);

Throughing = true;

PlaySounds(Sounds.Rush, false);

if (IsRight)

{

Velocity\_X = 38;

Acceleration\_X = -3.5;

Velocity\_Y = 2;

Invincible = true;

}

else

{

Velocity\_X = -38;

Acceleration\_X = +3.5;

Velocity\_Y = 2;

Invincible = true;

}

}

else if (RushTimer <= 16 && Step == 1)//煞車

{

CanToRushAttack;

CanToRushSkill;

}

else if (RushTimer >= 16 && Step >= 1)//煞車

{

Step = 2;

CanToRushAttack;

CanToRushSkill;

}

if (RushTimer >= 40 && RushTimer <= 80 && Step == 2)

{

if (IsRight&&Velocity\_X < 0)

Velocity\_X = 0;

else if (IsRight == false && Velocity\_X > 0)

Velocity\_X = 0;

CanToRushAttack;

CanToRushSkill;

}

#pragma endregion

//正常結束

if (RushTimer > 80 && Step == 2)

{

RushTimer = 0;

Throughing = false;

Invincible = false;

Velocity\_X = 0;

Acceleration\_X = 0;

#pragma region 回收特效

Effects.Content["AirBoost"].Step = 0;

Effects.Content["AirBoost"].visable = false;

Effects.Content["AirBoost"].OnUpdate("Effects", Camera);

#pragma endregion

#pragma region 判斷應該回到哪個動作

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

#pragma endregion

}

}

}

void Rina::GotoRunning(GPH)

{

Action = "移動";

Step = 0;

RunningTimer = 0;

}

void Rina::OnRunning(GPH)

{

if (Action == "移動")

{

AddSP(RunningSPincrements);

RunningTimer += TIMER\_TICK\_MILLIDECOND;

#pragma region 左右移動

if (CanControl&&IsRight&&OnGround && (Button\_now.button\_Right == true || Button\_now.button\_Left == true))

{

if (Button\_now.button\_Left == true)

IsRight = false;

if (RunningTimer >= 40)

{

RunningTimer = 0;

LoopStep(7);

}

RunAhead(0.8, RunSpeed);

}

else if (CanControl&&IsRight == false && OnGround && (Button\_now.button\_Right == true || Button\_now.button\_Left == true))

{

if (Button\_now.button\_Right == true)

IsRight = true;

if (RunningTimer >= 40)

{

RunningTimer = 0;

LoopStep(7);

}

RunAhead(0.8, RunSpeed);

}

#pragma endregion

#pragma region 到別的動作

CanToCharge;

CanToGuard;

CanToJump;

CanToNormalAttack1;

CanToRush;

CanToSkill1;

CanToUpAttack;

CanToUpSkill;

//正常結束(左右放開)

if (Button\_now.button\_Right == false && Button\_now.button\_Left == false)

CanToStandby;

#pragma endregion

}

}

void Rina::GotoJump(GPH)

{

if (OnGround)

{

JumpTimer2 = 0;

Action = "跳躍";

Step = 0;

RushTimer = 0;

JumpTimer = 0;

}

else

{

if (SP >= 10)

{

JumpTimer2 = 0;

GainSP(-10);

Action = "跳躍";

RushTimer = 0;

Step = 3;

JumpTimer = 0;

Velocity\_Y = -12;

OnGround = false;

PlayEffect(this, "Airboost2", Camera, Rect.X - 30, Rect.X - 35, Rect.Y + 80);

PlaySounds(Sounds.Jump, false);

}

}

}

void Rina::OnJump(GPH)

{

if (Action == "跳躍")

{

JumpTimer += TIMER\_TICK\_MILLIDECOND;

#pragma region 跳躍主程序

if (JumpTimer >= 10 && Step < 2)

{

ProduceFriction(0.15, 1);

Step += 1;

JumpTimer = 0;

}

else if (JumpTimer >= 20 && Step == 2)

{

Step = 3;

JumpTimer = 0;

Velocity\_Y = -12;

OnGround = false;

PlayEffect(this, "Airboost2", Camera, Rect.X - 30, Rect.X - 35, Rect.Y + 80);

PlaySounds(Sounds.Jump, false);

}

else if (Step >= 3 && Velocity\_Y < 0)

{

if (CanControl&&Button\_now.button\_Jump == true && Velocity\_Y < 2 && JumpTimer < 120)

Velocity\_Y -= 0.25;

}

else if (Velocity\_Y >= 0 && Step >= 3)

{

JumpTimer = 0;

Step = 4;

}

#pragma endregion

#pragma region 起跳後

if (Step >= 3)

{

#pragma region 空中移動

if (CanControl&&Button\_now.button\_Right == false && CanControl&&Button\_now.button\_Left == false)

{

ProduceFriction(0.15, 1);

}

else if (CanControl&&Button\_now.button\_Right == true)

{

IsRight = true;

RunAhead(0.5, RunSpeed / 2);

}

else if (CanControl&&Button\_now.button\_Left == true)

{

IsRight = false;

RunAhead(0.5, RunSpeed / 2);

}

#pragma endregion

#pragma region 到別的動作

CanToAirAttack1;

CanToRush;

CanToSkill1;

CanToAirDownAttack;

CanToAirUpAttack;

CanToJump;

CanToFastDrop;

//正常落地

if (OnGround)

CanToStandby;

#pragma endregion

}

#pragma endregion

}

}

void Rina::GotoCharge(GPH)

{

Action = "練氣";

Step = 0;

ChargeTimer = 0;

//啟動特效

PlayEffect(this, "SPCharge", Camera, Rect.X - 35, Rect.X - 30, Rect.Y - 45);

}

void Rina::OnCharge(GPH)

{

if (Action == "練氣")

{

ProduceFriction(1, 1);

ChargeTimer2 += TIMER\_TICK\_MILLIDECOND;

ChargeTimer += TIMER\_TICK\_MILLIDECOND;

if (ChargeTimer >= 50 && Step == 0)

{

Step = 1;

ChargeTimer = 0;

}

else if (ChargeTimer >= 16 && Step >= 1 && Step < 3)

{

Step += 1;

ChargeTimer = 0;

if (Step == 3)

{

ChargeTimer2 = 0;

Chargecount = 0;

PlaySounds(Sounds.SPCharge, false);

}

}

else if (ChargeTimer2 >= 10 && Step >= 3)

{

Chargecount += 1;

if (Chargecount < 10)

{

SP += ChargeSPincrements / 10;

if (SP > SP\_Max)

SP = SP\_Max;

}

if (Step == 3)

{

Step = 4;

}

else

{

Step = 3;

}

ChargeTimer2 = 0;

}

else if (ChargeTimer >= 90 && Step >= 3)//正常結束

{

ChargeTimer = 0;

if (Button\_now.button\_Guard == true)

{

GotoGuard(GPP);

}

else

{

GotoStandby(GPP);

}

}

}

}

void Rina::GotoNormalAttack1(GPH)

{

if (SP >= Rina\_NormalAttack1\_Cost)

{

GainSP(-Rina\_NormalAttack1\_Cost);

Action = "普攻1";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Rina::OnNormalAttack1(GPH)

{

if (Action == "普攻1")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 16 && Step <= 2)

{

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

//出拳

if (Step >= 3)

{

Velocity\_X += Ahead(4);

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Rina\_NormalAttack1\_Damage,

2, 2, Rect.X + 108, Rect.X - 2, Rect.Y + 52, 0, 0,

150, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

else if (NormalAttack1Timer >= 40 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 4)

{

//到別的動作

CanToNormalAttack2;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoNormalAttack2(GPH)

{

if (SP >= Rina\_NormalAttack2\_Cost)

{

GainSP(-Rina\_NormalAttack2\_Cost);

Action = "普攻2";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Rina::OnNormalAttack2(GPH)

{

if (Action == "普攻2")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 20 && Step >= 0 && Step < 2)

{

NormalAttack1Timer = 0;

Step += 1;

}

if (NormalAttack1Timer >= 12 && Step >= 2 && Step < 3)

{

NormalAttack1Timer = 0;

Step += 1;

if (Step == 3)

{

Velocity\_X += Ahead(4);

#pragma region 產生攻擊物件

//出拳

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Rina\_NormalAttack2\_Damage,

3.5, 2, Rect.X + 127, Rect.X - 15, Rect.Y + 52, 0, 0,

120, 30, "PunchHit", Sounds.NormalHit, Camera);

#pragma endregion

}

}

else if (NormalAttack1Timer >= 40 && Step == 3)

{

NormalAttack1Timer = 0;

Step = 4;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 4)

{

//到別的動作

CanToNormalAttack3;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoNormalAttack3(GPH)

{

if (SP >= Rina\_NormalAttack3\_Cost)

{

GainSP(-Rina\_NormalAttack3\_Cost);

Velocity\_X += Ahead(4);

Action = "普攻3";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Rina::OnNormalAttack3(GPH)

{

if (Action == "普攻3")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

ProduceFriction(0.25, 0.5);

if (NormalAttack1Timer >= 40 && Step == 0)

{

Step += 1;

NormalAttack1Timer = 0;

}

if (NormalAttack1Timer >= 24 && Step <= 4 && Step >= 1)

{

Step += 1;

NormalAttack1Timer = 0;

if (Step == 4)

{

Velocity\_Y -= 4;

#pragma region 產生攻擊物件

//基本設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Rina\_NormalAttack3\_Damage,

12, 7.5, Rect.X + 112, Rect.X - 5, Rect.Y + 67, 0, 0,

200, 100, "PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["Normal1"].CanHitFly = true;

#pragma endregion

}

}

else if (NormalAttack1Timer >= 30 && Step == 5)

{

NormalAttack1Timer = 0;

Step = 6;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 6)

{

CanToJump;

CanToRush;

}

else if (NormalAttack1Timer >= 100 && Step >= 6)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoSkill1(GPH)

{

if (SP >= Rina\_Skill1\_Cost)

{

GainSP(-Rina\_Skill1\_Cost);

if (Velocity\_Y > 0 && OnGround == false)

Velocity\_Y = 0;

Action = "特技";

Step = 0;

Shot1Timer = 0;

Velocity\_X = Ahead(30);

Throughing = true;

if (Button\_now.button\_Right)

IsRight = true;

else if (Button\_now.button\_Left)

IsRight = false;

}

}

void Rina::OnSkill1(GPH)

{

if (Action == "特技")

{

Shot1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

ProduceFriction(3, 1);

if (Shot1Timer >= 30 && Step == 0)

{

Shot1Timer = 0;

Step = 1;

}

else if (Shot1Timer >= 30 && abs(Velocity\_X) < 1 && (Step == 1 || Step == 2))

{

if (Button\_now.button\_Right)

IsRight = true;

else if (Button\_now.button\_Left)

IsRight = false;

if (Shot1Current >= 4)

Shot1Current = 0;

Shot1Timer = 0;

Step = 3;

#pragma region 產生攻擊物件

//出拳

PlaySounds(Sounds.light1, false);

Attacks.AttackReset\_Shot(&(Attacks.AttackObjects["flashblade\_H\_" + IntToString(Shot1Current)]), this, Enemy,

Rina\_DownSkill\_Damage,

2, 3.5,

Rect.X + 75, Rect.X + 10, Rect.Y + 60, Ahead(6), 0,

120, 1000, 2,

true, true, true,

"PunchHit", Sounds.SliceHit, Camera);

Shot1Current += 1;

(Attacks.AttackObjects["flashblade\_H\_" + IntToString(Shot1Current)]).HitNoon = false;

#pragma endregion

}

if (Shot1Timer >= 16 && Step >= 3 && Step < 5)

{

Step += 1;

Shot1Timer = 0;

}

#pragma endregion

#pragma region 到別的動作

if (Shot1Timer < 100 && Step >= 5)

{

Throughing = false;

//到別的可能動作

if (OnGround)

{

CanToNormalAttack1;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

CanToSkill1;

}

else

{

CanToAirUpAttack;

CanToAirDownAttack;

CanToAirAttack1;

CanToFastDrop;

CanToSkill1;

}

CanToRush;

CanToJump;

}

else if (Shot1Timer >= 100 && Step >= 5)

{

Throughing = false;

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

#pragma endregion

}

}

void Rina::GotoAirAttack1(GPH)

{

}

void Rina::OnAirAttack1(GPH)

{

if (Action == "空普1")

{

}

}

void Rina::GotoAirAttack2(GPH)

{

}

void Rina::OnAirAttack2(GPH)

{

if (Action == "空普2")

{

}

}

void Rina::GotoUpAttack(GPH)

{

if (SP >= Rina\_UpAttack\_Cost)

{

GainSP(-Rina\_UpAttack\_Cost);

Action = "上普";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Rina::OnUpAttack(GPH)

{

if (Action == "上普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 60 && Step == 0)

{

NormalAttack1Timer = 0;

Step = 1;

}

else if (NormalAttack1Timer >= 16 && Step >= 1 && Step <= 3)

{

NormalAttack1Timer = 0;

Step += 1;

if (Step == 2)

{

Velocity\_Y = -10;

}

#pragma region 產生攻擊物件

//出拳

if (Step == 3)

{

PlayEffect(this, "Airboost2", Camera, Rect.X - 15, Rect.X - 50, Rect.Y + 90);

Velocity\_X += Ahead(8);

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal2"]), this, Enemy,

Rina\_UpAttack\_Damage,

5, 11, Rect.X + 147, Rect.X - 7, Rect.Y + 40, 0, 0,

220, 30, "PunchHit", Sounds.NormalHit, Camera);

}

#pragma endregion

}

else if (NormalAttack1Timer >= 40 && Step == 4)

{

NormalAttack1Timer = 0;

Step = 5;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 5)

{

//到別的動作

CanToSkill1;

CanToAirDownAttack;

CanToJump;

CanToRush;

}

else if (NormalAttack1Timer >= 100 && Step >= 5)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoDownAttack(GPH)

{

if (SP >= Rina\_DownAttack\_Cost)

{

GainSP(-Rina\_DownAttack\_Cost);

Action = "下普";

Step = 0;

NormalAttack1Timer = 0;

Velocity\_X = Ahead(-12);

Velocity\_Y = -3;

}

}

void Rina::OnDownAttack(GPH)

{

if (Action == "下普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

if (Step == 0)

ProduceFriction(0.5, 0.5);

else if (Step == 1)

ProduceFriction(0.3, 0.5);

else if (Step >= 2)

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 50 && Step == 0)

{

NormalAttack1Timer = 0;

Step = 1;

}

else if (NormalAttack1Timer >= 150 && Step == 1)

{

NormalAttack1Timer = 0;

Step = 2;

Velocity\_X = Ahead(35);

}

else if (NormalAttack1Timer >= 25 && Step == 2)

{

NormalAttack1Timer = 0;

Step = 3;

}

else if (NormalAttack1Timer >= 16 && Step >= 3 && Step <= 4)

{

Velocity\_X = Ahead(3);

NormalAttack1Timer = 0;

Step += 1;

#pragma region 產生攻擊物件

//出拳

if (Step == 4)

{

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Rina\_DownAttack\_Damage,

2, 3, Rect.X + 90, Rect.X + 20, Rect.Y + 65, 0, 0,

200, 30, "PunchHit", Sounds.NormalHit, Camera);

Attacks.AttackObjects["Normal1"].HitBreak = true;

Attacks.AttackObjects["Normal1"].Drawable = false;

}

#pragma endregion

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 5)

{

//到別的動作

CanToNormalAttack1;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpSkill;

}

else if (NormalAttack1Timer >= 100 && Step >= 5)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoRushAttack(GPH)

{

if (SP >= Rina\_RushAttack\_Cost)

{

GainSP(-Rina\_RushAttack\_Cost);

if (Velocity\_Y > 0 && OnGround == false)

Velocity\_Y = 0;

RushTimer = 0;

IsRushAttack = false;

Throughing = false;

Invincible = false;

Acceleration\_X = 0;

Action = "衝刺普";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Rina::OnRushAttack(GPH)

{

if (Action == "衝刺普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

ProduceFriction(0.25, 0.5);

#pragma region 動作主體

if (NormalAttack1Timer <= 110 && Step >= 0)

{

if (abs(Velocity\_X) > 12)

{

Velocity\_X = Ahead(12);

}

if (NormalAttack1Timer >= 25)

Step = 1;

if (NormalAttack1Timer >= 50)

Step = 2;

#pragma region 更新攻擊物件位置

if (IsRight)

Attacks.AttackObjects["Normal1"].Rect.X = Rect.X + 90;

else

Attacks.AttackObjects["Normal1"].Rect.X = Rect.X;

Attacks.AttackObjects["Normal1"].Rect.Y = Rect.Y + 55;

if (Attacks.AttackObjects["Normal1"].IsHited && (Attacks.AttackObjects["Normal1"].ComboTimer > TIMER\_TICK\_MILLIDECOND \* 4))

{

Attacks.AttackObjects["Normal1"].ComboTimer = 0;

Attacks.AttackObjects["Normal1"].IsHited = false;

}

#pragma endregion

#pragma region 產生攻擊物件

if (IsRushAttack == false)

{

IsRushAttack = true;

//基礎設定

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal1"]), this, Enemy,

Matchstick\_RushAttack\_Damage,

9.5, 0, Rect.X + 98, 0, 0, 0, 0,

100, 100, "PunchHit", Sounds.NormalHit, Camera);

//額外設定

Attacks.AttackObjects["Normal1"].CanCombo = true;//可連擊

Attacks.AttackObjects["Normal1"].HitNoon = false;

Attacks.AttackObjects["Normal1"].Drawable = false;

}

#pragma endregion

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer > 110 && Step >= 0)

{

Acceleration\_X = 0;

Velocity\_X = 0;

#pragma region 判斷應該回到哪個動作

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

#pragma endregion

}

#pragma endregion

}

}

void Rina::GotoAirUpAttack(GPH)

{

}

void Rina::OnAirUpAttack(GPH)

{

if (Action == "空上普")

{

}

}

void Rina::GotoAirDownAttack(GPH)

{

if (SP >= Rina\_AirDownAttack\_Cost)

{

GainSP(-Rina\_AirDownAttack\_Cost);

Action = "空下普";

Step = 0;

Velocity\_Y = -9;

Velocity\_X = Ahead(-4.5);

NormalAttack1Timer = 0;

}

}

void Rina::OnAirDownAttack(GPH)

{

if (Action == "空下普")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

if (NormalAttack1Timer < 150 && Step == 0)

{

ProduceFriction(0.25, 1);

}

else if (NormalAttack1Timer >= 150 && Step == 0)

{

Velocity\_X = Ahead(7.25);

Velocity\_Y = 20;

Step = 1;

NormalAttack1Timer = 0;

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Normal3"]), this, Enemy,

Rina\_AirDownAttack\_Damage,

2.5, -18, Rect.X + 90, Rect.X, Rect.Y + 20, 0, 0,

200, 30, "PunchHit", Sounds.NormalHit2, Camera);

}

else if (Rect.Y < GroundPosition && Step == 1)

{

Attacks.AttackObjects["Normal3"].Rect.X = Rect.X + Ahead(30);

Attacks.AttackObjects["Normal3"].Rect.Y = Rect.Y + 110;

Attacks.AttackObjects["Normal3"].AliveTimer = 0;

}

else if (Rect.Y >= GroundPosition && Step == 1)

{

NormalAttack1Timer = 0;

Step = 2;

}

#pragma endregion

#pragma region 到別的動作

if (NormalAttack1Timer < 100 && Step >= 2)

{

//到別的動作

CanToNormalAttack1;

CanToSkill1;

CanToJump;

CanToRush;

CanToUpSkill;

CanToAirUpAttack;

}

else if (NormalAttack1Timer >= 100 && Step >= 2)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoUpSkill(GPH)

{

if (SP >= Rina\_UpSkill\_Cost)

{

GainSP(-Rina\_UpSkill\_Cost);

Action = "上特技";

Step = 0;

Shot1Timer = 0;

Velocity\_X = -Ahead(12);

Velocity\_Y = -6;

if (Button\_now.button\_Right)

IsRight = true;

else if (Button\_now.button\_Left)

IsRight = false;

}

}

void Rina::OnUpSkill(GPH)

{

if (Action == "上特技")

{

Shot1Timer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

ProduceFriction(0.25, 0.5);

if (Shot1Timer >= 30 && Step == 0)

{

Shot1Timer = 0;

Step = 1;

}

if (Shot1Timer >= 80 && Step == 1)

{

if (Shot1Current >= 4)

Shot1Current = 0;

Shot1Timer = 0;

Step = 2;

#pragma region 產生攻擊物件

//出拳

PlaySounds(Sounds.light1, false);

Attacks.AttackReset\_Shot(&(Attacks.AttackObjects["flashblade\_H\_" + IntToString(Shot1Current)]), this, Enemy,

Rina\_DownSkill\_Damage,

2, 3.5,

Rect.X + 75, Rect.X + 10, Rect.Y + 60, Ahead(6), 0,

120, 1000, 2,

true, true, true,

"PunchHit", Sounds.SliceHit, Camera);

Shot1Current += 1;

(Attacks.AttackObjects["flashblade\_H\_" + IntToString(Shot1Current)]).HitNoon = false;

#pragma endregion

}

if (Shot1Timer >= 16 && Step >= 2 && Step < 4)

{

Step += 1;

Shot1Timer = 0;

}

#pragma endregion

#pragma region 到別的動作

if (Shot1Timer < 100 && Step >= 4)

{

Throughing = false;

//到別的可能動作

if (OnGround)

{

CanToNormalAttack1;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

CanToSkill1;

}

else

{

CanToAirUpAttack;

CanToAirDownAttack;

CanToAirAttack1;

CanToFastDrop;

CanToSkill1;

}

CanToRush;

CanToJump;

}

else if (Shot1Timer >= 100 && Step >= 4)

{

Throughing = false;

//正常結束

if (OnGround)

GotoStandby(GPP);

else

GotoDrop(GPP);

}

#pragma endregion

}

}

void Rina::GotoDownSkill(GPH)

{

if (SP >= Rina\_DownSkill\_Cost&&useDownSkill == false)

{

GainSP(-Rina\_DownSkill\_Cost);

Action = "下特技";

Step = 0;

NormalAttack1Timer = 0;

}

}

void Rina::OnDownSkill(GPH)

{

if (Action == "下特技")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

DownSkillTimer += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 24 && Step >= 0 && Step <= 3)

{

NormalAttack1Timer = 0;

Step += 1;

DownSkillTimer = 0;

}

#pragma endregion

//出拳

if (Step >= 4 && DownSkillTimer >= 30)

{

DownSkillTimer = 0;

if (DownSkillCurrent >= 8)

DownSkillCurrent = 0;

DownSkillTimer2 = 0;

useDownSkill = true;

DownSkillXpoint = Enemy->Rect.X;

DownSkillYpoint = Enemy->Rect.Y;

}

#pragma region 到別的動作

if (NormalAttack1Timer >= 20 && NormalAttack1Timer < 150 && Step >= 4)

{

//到別的動作

CanToNormalAttack1;

CanToSkill1;

CanToJump;

CanToRush;

CanToDownAttack;

}

else if (NormalAttack1Timer >= 150 && Step >= 4)

{

//正常結束

GotoStandby(GPP);

}

#pragma endregion

}

}

void Rina::GotoRushSkill(GPH)

{

if (SP >= Rina\_RushSkill\_Cost)

{

}

}

void Rina::OnRushSkill(GPH)

{

if (Action == "衝刺特技")

{

}

}

void Rina::GotoAirUpSkill(GPH)

{

}

void Rina::OnAirUpSkill(GPH)

{

}

void Rina::GotoAirDownSkill(GPH)

{

}

void Rina::OnAirDownSkill(GPH)

{

}

void Rina::GotoUltimateSkill(GPH)

{

if (SP > (SP\_Max / 3))

{

UltraSkillcostSP = SP - (SP\_Max / 3);

SP = 0;

Action = "大絕";

Step = 0;

NormalAttack1Timer = 0;

UltraSkillTimer1 = 0;

}

}

void Rina::OnUltimateSkill(GPH)

{

if (Action == "大絕")

{

NormalAttack1Timer += TIMER\_TICK\_MILLIDECOND;

UltraSkillTimer1 += TIMER\_TICK\_MILLIDECOND;

#pragma region 動作主體

//處理摩擦力

ProduceFriction(1, 1);

if (NormalAttack1Timer >= 300 && Step == 0)

{

PlaySounds(Sounds.CutIn, false);

NeedCutIn = true;

NormalAttack1Timer = 0;

Step = 1;

Effects.BootEffect(&Effects.Content["Rina\_US"], Camera, 800, 800, 0, 0, 0, false, true);

}

if (Step == 1 && NormalAttack1Timer >= 20)

{

Step = 2;

Attacks.AttackReset\_Normal(

&(Attacks.AttackObjects["Bigflashblade"]), this, Enemy,

Rina\_UltimateSkill\_Damage,

5, 20, Enemy->Rect.X - 100, Enemy->Rect.X, -700, Ahead(3), 8,

200, 1500, "PunchHit", Sounds.NormalHit, Camera);

Attacks.AttackObjects["Bigflashblade"].HitBreak = true;

Attacks.AttackObjects["Bigflashblade"].HitNoon = false;

Attacks.AttackObjects["Bigflashblade"].Drawable = true;

Attacks.AttackObjects["Bigflashblade"].GravityEffect = true;

Attacks.AttackObjects["Bigflashblade"].Acceleration\_Y = 1;

}

if (UltraSkillTimer1 >= 500 && Step == 2)

{

GotoStandby(GPP);

}

#pragma endregion

}

}

#pragma endregion

}

Bar.h

#pragma once

namespace game\_framework

{

class Bar :public BitmapPicture

{

friend class CDDraw;

public:

Bar();

Bar(string, int, int, int, bool);

~Bar();

int BelongPlayer;//屬於誰的(靠左或靠右)

int newWidth;//新的寬度

irtual void Draw(int, int,double,double);//繪出圖型

virtual void Draw(int, int, double, double,CameraPosition);//繪出圖型

};

}

Bar.cpp

#include "stdafx.h"

#include <ddraw.h>

#include <windows.h>

#include <stdio.h>

#include <sstream>

#include "gamelib.h"

#include "WKBitmap.h"

#include "TypeConverter.h"

#include "Bar.h"

namespace game\_framework

{

Bar::Bar()

{

}

Bar::Bar(string path, int BelongPlayerin, int X, int Y, bool vis) : BitmapPicture(path, X, Y, vis, false, false)

{

BelongPlayer = BelongPlayerin;

newWidth = 200;

}

Bar::~Bar()

{

}

void Bar::Draw(int CurrentLayer, int TargetLayer, double now, double max)

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

newWidth = (int)(((double)Rect.Width)\*(now / max));

if (newWidth <= 0)

this->visable = false;

else if (newWidth > 0)

this->visable = true;

if (CurrentLayer == TargetLayer&&this->visable == true)//直到他的圖片層級才可以顯示

{

if (BelongPlayer == 1)

{

SetTopLeft(Rect.X\_int, Rect.Y\_int);

ShowBitmap(this->Rect.X\_int, this->Rect.Y\_int, this->Rect.X\_int + newWidth, this->Rect.Y\_int + this->Rect.Height);

}

else if (BelongPlayer == 2)

{

SetTopLeft(Rect.X\_int, Rect.Y\_int);

ShowBitmap(this->Rect.X\_int + (Rect.Width - newWidth), this->Rect.Y\_int, this->Rect.X\_int + Rect.Width, this->Rect.Y\_int + this->Rect.Height);

}

}

}

void Bar::Draw(int CurrentLayer, int TargetLayer, double now, double max, CameraPosition Camera)

{

Rect.X\_int = (int)(Rect.X - Camera.X\_double);

Rect.Y\_int = (int)(Rect.Y - Camera.Y\_double);

newWidth = (int)(((double)Rect.Width)\*(now / max));

if (newWidth <= 0)

this->visable = false;

else if (newWidth > 0)

this->visable = true;

if (CurrentLayer == TargetLayer&&this->visable == true)//直到他的圖片層級才可以顯示

{

if (BelongPlayer == 1)

{

SetTopLeft(Rect.X\_int, Rect.Y\_int);

ShowBitmap(this->Rect.X\_int, this->Rect.Y\_int, this->Rect.X\_int + newWidth, this->Rect.Y\_int + this->Rect.Height);

}

else if (BelongPlayer == 2)

{

SetTopLeft(Rect.X\_int, Rect.Y\_int);

ShowBitmap(this->Rect.X\_int + (Rect.Width - newWidth), this->Rect.Y\_int, this->Rect.X\_int + Rect.Width, this->Rect.Y\_int + this->Rect.Height);

}

}

}

}

BattlePlayer.h

#pragma once

#include "stdafx.h"

#include "Keycode.h"

#include "KeyBoardState.h"

#include "CollisionSensor.h"

#include "WKAudio.h"

#include "EffectSprite.h"

#include "AttackObj.h"

using namespace std;

using namespace WKAudio\_namespace;

using namespace CollisionSensor\_namespace;

namespace game\_framework

{

#pragma region Defines

//程式編碼用定義

//Game\_Parameter\_Header

#define GPH BattlePlayer \*Enemy, CameraPosition Camera, KeyBoardState KeyState\_now, KeyBoardState KeyState\_last, Audio\_ID Sounds, COLORREF color

//Game\_Parameter\_Parameter

#define GPP Enemy, Camera, KeyState\_now, KeyState\_last, Sounds,color

//GameCanTo

#define CanToStandby GotoStandby(GPP)

#define CanToRunning GotoRunning(GPP)

#define CanToJump if (CanControl&&Button\_now.button\_Jump&&Button\_last.button\_Jump == false){GotoJump(GPP);}

#define CanToRush if (CanControl&&Button\_now.button\_Rush&&Button\_last.button\_Rush == false&&Button\_now.button\_Down == false){GotoRush(GPP);}

#define CanToGuard if (CanControl&&Button\_now.button\_Guard && Button\_now.button\_Down == false && OnGround){GotoGuard(GPP);}

#define CanToCharge if (CanControl&&Button\_now.button\_Guard&&Button\_last.button\_Guard == false && Button\_now.button\_Down && OnGround){GotoCharge(GPP);}

#define CanToNormalAttack1 if (CanControl&&Button\_now.button\_Attack&&Button\_now.button\_Up == false && Button\_now.button\_Down == false&&Button\_last.button\_Attack == false && OnGround){GotoNormalAttack1(GPP);}

#define CanToSkill1 if (CanControl&&Button\_now.button\_Skill&&Button\_now.button\_Up == false&& Button\_now.button\_Down == false&&Button\_last.button\_Skill == false){GotoSkill1(GPP);}

#define CanToAirAttack1 if (CanControl&&OnGround == false&&Button\_now.button\_Up == false && Button\_now.button\_Down == false &&Button\_now.button\_Attack&& Button\_last.button\_Attack == false){GotoAirAttack1(GPP);}

#define CanToAirAttack2 if (CanControl&&OnGround == false&&Button\_now.button\_Up == false && Button\_now.button\_Down == false &&Button\_now.button\_Attack&& Button\_last.button\_Attack == false){GotoAirAttack2(GPP);}

#define CanToAirDownAttack if (CanControl&&OnGround == false&&Button\_now.button\_Up == false && Button\_now.button\_Down == true &&Button\_now.button\_Attack&& Button\_last.button\_Attack == false){GotoAirDownAttack(GPP);}

#define CanToAirUpAttack if (CanControl&&Button\_now.button\_Up == true && Button\_now.button\_Down == false &&Button\_now.button\_Attack&& Button\_last.button\_Attack == false){GotoAirUpAttack(GPP);}

#define CanToUpAttack if (CanControl&&Button\_now.button\_Up == true && Button\_now.button\_Down == false &&Button\_now.button\_Attack&& Button\_last.button\_Attack == false){GotoUpAttack(GPP);}

#define CanToDownAttack if (CanControl&&Button\_now.button\_Up == false && Button\_now.button\_Down == true &&Button\_now.button\_Attack&& Button\_last.button\_Attack == false){GotoDownAttack(GPP);}

#define CanToDownSkill if (CanControl&&Button\_now.button\_Up == false && Button\_now.button\_Down == true &&Button\_now.button\_Skill&& Button\_last.button\_Skill == false){GotoDownSkill(GPP);}

#define CanToFastDrop if (CanControl&&Button\_now.button\_Down == true && Button\_now.button\_Rush == true && Button\_last.button\_Rush == false){GotoDrop(GPP);FastDrop(GPP);}

#define CanToRushAttack if (CanControl&& Button\_now.button\_Attack == true){GotoRushAttack(GPP);}

#define CanToRushSkill if (CanControl&& Button\_now.button\_Skill == true){GotoRushSkill(GPP);}

#define CanToUpSkill if (CanControl&&Button\_now.button\_Up == true && Button\_now.button\_Down == false &&Button\_now.button\_Skill&& Button\_last.button\_Skill == false){GotoUpSkill(GPP);}

#define CanToUltimateSkill if (CanControl&&Button\_now.button\_Technique == true && Button\_last.button\_Technique == false&&OnGround){GotoUltimateSkill(GPP);}

//Inputconfigure

#define Player1\_Left A

#define Player1\_Right D

#define Player1\_Up W

#define Player1\_Down S

#define Player1\_Attack G

#define Player1\_Skill T

#define Player1\_Technique Y

#define Player1\_Jump F

#define Player1\_Rush R

#define Player1\_Guard E

#define Player2\_Left Left

#define Player2\_Right Right

#define Player2\_Up Up

#define Player2\_Down Down

#define Player2\_Attack K

#define Player2\_Skill L

#define Player2\_Technique O

#define Player2\_Jump J

#define Player2\_Rush I

#define Player2\_Guard P

//共用設定

#define GuardSPCost 0.15

#define MaxGravity 9

#pragma endregion

struct BattleInput

{

bool button\_Right = false;

bool button\_Left = false;

bool button\_Up = false;

bool button\_Down = false;

bool button\_Attack = false;

bool button\_Skill = false;

bool button\_Technique = false;

bool button\_Jump = false;

bool button\_Rush = false;

bool button\_Guard = false;

};

class BattlePlayer :public BitmapAnimation

{

public:

BattlePlayer();

virtual ~BattlePlayer();

//共用建置函數(大多需要繼承額外撰寫)

virtual void AnimationUpdate(CameraPosition);

virtual void Draw(int, int, CameraPosition);//更新函式，且隨著視角移動

virtual void AutoLoadBitmaps(GPH);//依照各自角色讀檔

virtual void OnUpdate(GPH);//更新函式，且隨著視角移動

virtual void InsertBitmapPicture(string, int, COLORREF);//使BimapAnimation讀取圖檔

virtual void InsertAction(string, int, COLORREF);//讀取一個動作的所有圖檔

//共用基礎動作函數

virtual void CheckHit(GPH);//受傷觸發瞬間函式

virtual void InputJudge(KeyBoardState, KeyBoardState);//輸入更新按鈕

//待機動作

virtual void GotoStandby(GPH);

virtual void OnStandby(GPH);

//跑步動作

virtual void GotoRunning(GPH);

virtual void OnRunning(GPH);

//衝刺動作

virtual void GotoRush(GPH);

virtual void OnRush(GPH);

//急墜動作

virtual void FastDrop(GPH);

//跳躍動作

virtual void GotoJump(GPH);

virtual void OnJump(GPH);

//防禦動作

virtual void GotoGuard(GPH);

virtual void OnGuard(GPH);

//練氣動作

virtual void GotoCharge(GPH);

virtual void OnCharge(GPH);

//受傷動作

virtual void OnHit(GPH);

virtual void OnHitGuard(GPH);

//攻擊動作

virtual void GotoNormalAttack1(GPH) = 0;//需要每個角色個別撰寫

virtual void OnNormalAttack1(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoAirAttack1(GPH) = 0;//需要每個角色個別撰寫

virtual void OnAirAttack1(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoSkill1(GPH) = 0;//需要每個角色個別撰寫

virtual void OnSkill1(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoUpAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void OnUpAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoDownAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void OnDownAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoRushAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void OnRushAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoAirUpAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void OnAirUpAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoAirDownAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void OnAirDownAttack(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoUpSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void OnUpSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoDownSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void OnDownSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoRushSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void OnRushSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoAirUpSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void OnAirUpSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoAirDownSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void OnAirDownSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void GotoUltimateSkill(GPH) = 0;//需要每個角色個別撰寫

virtual void OnUltimateSkill(GPH) = 0;//需要每個角色個別撰寫

//套裝函式------------------------------------------------------------------------------------------------------

virtual void AddSP(double mathin);//增加SP

virtual void GainHP(double mathin);//增加HP

virtual void GainSP(double mathin);//增加SP

virtual void ProduceFriction(double, double);//處理摩擦力(讓角色漸漸慢下來)

virtual void GotoDrop(GPH);

virtual void LoopStep(int);//(重複同樣動作)

virtual void RunAhead(double, double);//(正確方向走)

virtual void PhysicalMovement(GPH);//物理移動，全角色共用，除非特例

virtual void ProduceBreakPoint(GPH);//處理失衡值

virtual void ProduceRecovery(GPH);//處理失生命回復

virtual void CheckPerfectBlock(GPH);//處理完美格黨

virtual double Ahead(double move);//依照正確方向行動回傳正負(以右為準)

//能力值變數(由子類別初始化)------------------------------------------------------------------------------------------------------

double HP\_Max;//最大生命值

double SP\_Max;//最大氣力

double recovery;

double Rush\_cost;//衝刺消耗氣力量

double RunSpeed;//跑速

double StandbySPincrements;//待機時回復的氣力量

double RunningSPincrements;//移動時回復的氣力量

double ChargeSPincrements;//移動時回復的氣力量

//現狀變數------------------------------------------------------------------------------------------------------

int PlayerNumber;//玩家編號

double HP;//當前生命

double SP;//當前氣力

double BreakPoint;//失衡值

bool CanControl;//可以控制

bool Invincible;//無敵狀態

bool visable;//是否可見

bool CanPixelCollision;//是否使用像素碰撞，所有動作分割都會套用

bool InSideCamera;//是否受到鏡頭影響

bool IsRight;//面相右邊

bool OnGround;//是否在地面上

bool Throughing;//可否穿越

bool HitFly;//被擊非

bool NeedCutIn;//是否需要大絕進場特效

string Action;//動作狀態

int Step;//當前步驟數

vector<bool> AttributeState;//異常狀態

//Timer及次數控制參數------------------------------------------------------------------------------------------------------

double SPincrementsTimer = 0;

double StandbyTimer = 0;

double RunningTimer = 0;

double RushTimer = 0;

double JumpTimer = 0;

double ChargeTimer = 0;

double ChargeTimer2 = 0;

double BreakPointTimer = 0;

int Chargecount = 0;

double BeHitTimer = 0;

double BeHitTimeMax = 0;

double NotHitTimer = 0;

double OnHitTimer = 0;

double ClickDefendTimer = 0;

double BetweenTwiceClickTimer = 0;

//輸入按鍵參數------------------------------------------------------------------------------------------------------

BattleInput Button\_now;//現在的鍵盤狀態

BattleInput Button\_last;//上一瞬間的鍵盤狀態

//物理參數------------------------------------------------------------------------------------------------------

double Velocity\_X;//X速度

double Velocity\_Y;//Y速度

double Acceleration\_X;//X加速度

double Acceleration\_Y;//Y加速度

double Acceleration\_gravity = 0.55;//重力加速度

//成員------------------------------------------------------------------------------------------------------

BitmapPicture \*DisplayBitmap;//永遠把顯示用的Bitmap指向對應的BitmapPicture==當前所顯示的BitmapPicture

BitmapPicture BodyPicture;//有效碰撞區

BitMapRectangle Rect;//圖片材質矩形 掌管座標跟長寬，會在讀取檔案時設定完成，並且可以在外部更動

BitMapRectangle BodyRect;//人物碰撞的矩形

map<string, BitmapPicture> BitmapPictures;//該Animation的所有圖片動作

EffectSprite Effects;//特效物件集合

AttackManager Attacks;//攻擊物件集合

};

}

BattlePlayer.cpp

#pragma once

#include "stdafx.h"

#include <ddraw.h>

#include <windows.h>

#include <stdio.h>

#include <sstream>

#include "gamelib.h"

#include "WKBitmap.h"

#include "Keycode.h"

#include "KeyBoardState.h"

#include "WKAudio.h"

#include "CollisionSensor.h"

#include "EffectSprite.h"

#include "BattlePlayer.h"

#include "FunctionUser.h"

using namespace std;

using namespace WKAudio\_namespace;

using namespace CollisionSensor\_namespace;

using namespace FunctionUser\_namespace;

namespace game\_framework

{

BattlePlayer::BattlePlayer() :BitmapAnimation()

{

BodyPicture = BitmapPicture("Content\\Bitmaps\\BodyRect.bmp", 0, 0, true, true, true);

AttributeState = vector<bool>(10, false);

Effects = EffectSprite();

Attacks = AttackManager();

NeedCutIn = false;

}

BattlePlayer::~BattlePlayer()

{

}

void BattlePlayer::AnimationUpdate(CameraPosition Camera)

{

#pragma region 確定圖檔名稱

//確定圖檔名稱

if (IsRight)

{

string Actionstring = "Content\\Bitmaps\\" + GetName() + "\\" + Action + "\_" + IntToString(Step) + ".bmp";

char \*cc = new char[65535];

strcpy(cc, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cc];

delete[] cc;

}

else

{

string Actionstring = "Content\\Bitmaps\\" + GetName() + "\\" + Action + "\_" + IntToString(Step) + "\_L.bmp";

char \*cc = new char[65535];

strcpy(cc, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cc];

delete[] cc;

}

#pragma endregion

#pragma region 決定相對座標

//決定相對座標

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

if (InSideCamera)

{

Rect.X\_int = (int)(Rect.X - Camera.X);

Rect.Y\_int = (int)(Rect.Y - Camera.Y);

}

else

{

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

}

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

BodyRect.X = (Rect.X + 65);

BodyRect.Y = (Rect.Y + 41);

BodyRect.Width = 52;

BodyRect.Height = 120;

BodyPicture.Rect.X = BodyRect.X;

BodyPicture.Rect.Y = BodyRect.Y;

BodyPicture.OnUpdate(Camera);

DisplayBitmap->OnUpdate();

#pragma endregion

}

void BattlePlayer::OnUpdate(GPH)

{

AnimationUpdate(Camera);

}

void BattlePlayer::PhysicalMovement(GPH)

{

#pragma region 基礎移動

Rect.X += Velocity\_X;

Rect.Y += Velocity\_Y;

Velocity\_X += Acceleration\_X;

Velocity\_Y += Acceleration\_Y + (Acceleration\_gravity\*0.9);

if (Velocity\_Y > MaxGravity)//如果大於最大正常下降速度，則不加重力加速度

{

Velocity\_Y -= Acceleration\_gravity;

}

if (Rect.Y >= GroundPosition)

{

Rect.Y = GroundPosition;

if (this->Action == "受傷"&&OnGround == false && this->Velocity\_Y > 14)

{

this->Velocity\_Y \*= -0.45;

this->HP -= 15;

this->HitFly = true;

this->BeHitTimeMax += 200;

PlaySounds(Sounds.HitWall, false);

Sleep(100);

OnGround = false;

BreakPoint += 30;

if (BreakPoint > 90)

{

BreakPoint = 90;

}

}

else

{

Velocity\_Y = 0;

OnGround = true;

}

Acceleration\_Y = 0;

}

else if (Rect.Y < GroundPosition)

{

OnGround = false;

if (Action == "待機")

{

Action = "跳躍";

Step = 4;

}

}

if (BitmapPicture\_HitRectangle((this->BodyRect), (Enemy->BodyRect)) == true && this->Throughing == false && Enemy->Throughing == false)

{

if (Velocity\_X == 0 && Enemy->Velocity\_X == 0 && Enemy->BodyRect.X >= BodyRect.X)

{

while (BitmapPicture\_HitRectangle((this->BodyRect), (Enemy->BodyRect)))

{

Rect.X -= 1;

Rect.X\_int -= 1;

BodyRect.X -= 1;

Enemy->Rect.X += 1;

Enemy->Rect.X\_int += 1;

Enemy->BodyRect.X += 1;

}

}

if (Velocity\_X > 0 && Enemy->Velocity\_X == 0 && Enemy->BodyRect.X >= BodyRect.X)

{

while (BitmapPicture\_HitRectangle((this->BodyRect), (Enemy->BodyRect)))

{

Rect.X -= 1;

Rect.X\_int -= 1;

BodyRect.X -= 1;

Enemy->Rect.X += 1;

Enemy->Rect.X\_int += 1;

Enemy->BodyRect.X += 1;

}

}

if (Velocity\_X < 0 && Enemy->Velocity\_X == 0 && Enemy->BodyRect.X <= BodyRect.X)

{

while (BitmapPicture\_HitRectangle((this->BodyRect), (Enemy->BodyRect)))

{

Rect.X += 1;

Rect.X\_int += 1;

BodyRect.X += 1;

Enemy->Rect.X -= 1;

Enemy->Rect.X\_int -= 1;

Enemy->BodyRect.X -= 1;

}

}

if (Velocity\_X > 0 && Enemy->Velocity\_X < 0 && Enemy->BodyRect.X >= BodyRect.X)

{

while (BitmapPicture\_HitRectangle((this->BodyRect), (Enemy->BodyRect)))

{

Rect.X -= 1;

Rect.X\_int -= 1;

BodyRect.X -= 1;

Enemy->Rect.X += 1;

Enemy->Rect.X\_int += 1;

Enemy->BodyRect.X += 1;

}

}

}

#pragma endregion

}

void BattlePlayer::ProduceBreakPoint(GPH)

{

#pragma region 失衡值

if (BreakPoint > 0 && BreakPoint < 90)

{

BreakPoint -= 0.05;

}

if (BreakPoint > 90)

{

BreakPoint = 90;

BreakPointTimer = 0;

}

if (BreakPoint == 90)

{

BreakPoint = 90;

BreakPointTimer += TIMER\_TICK\_MILLIDECOND;

if (BreakPointTimer > 2500)

{

BreakPointTimer = 0;

BreakPoint = 0;

}

}

#pragma endregion

}

void BattlePlayer::ProduceRecovery(GPH)

{

#pragma region 回復血量

if (recovery > 0)

{

GainHP(0.1);

recovery -= 0.1;

}

#pragma endregion

}

void BattlePlayer::CheckPerfectBlock(GPH)

{

if (Button\_now.button\_Guard == Button\_last.button\_Guard && Button\_now.button\_Guard == true)

{

BetweenTwiceClickTimer = 0;

ClickDefendTimer += TIMER\_TICK\_MILLIDECOND;

}

if (Button\_now.button\_Guard == false)

{

BetweenTwiceClickTimer += TIMER\_TICK\_MILLIDECOND;

}

if (BetweenTwiceClickTimer > 100)

{

BetweenTwiceClickTimer = 0;

ClickDefendTimer = 0;

}

}

void BattlePlayer::Draw(int i, int j, CameraPosition Camera)

{

this->DisplayBitmap->Draw(i, j);

//BodyPicture.Draw(i, j);

Effects.DrawAllEffection(i);

Attacks.DrawAllAttacks(i);

}

void BattlePlayer::AutoLoadBitmaps(GPH)

{

}

void BattlePlayer::InsertBitmapPicture(string action, int step, COLORREF color)

{

string str;

str = ("Content\\Bitmaps\\" + GetName() + "\\" + action + "\_" + IntToString(step) + ".bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*cr = new char[65535];

strcpy(cr, str.c\_str());

BitmapPictures[str].LoadTexture(cr, false, color);

str = ("Content\\Bitmaps\\" + GetName() + "\\" + action + "\_" + IntToString(step) + "\_L.bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*cl = new char[65535];

strcpy(cl, str.c\_str());

BitmapPictures[str].LoadTexture(cl, false, color);

delete[] cr;

delete[] cl;

}

void BattlePlayer::InsertAction(string actionname, int maxstep, COLORREF color)

{

for (int i = 0; i <= maxstep; i += 1)

{

InsertBitmapPicture(actionname, i, color);

}

}

void BattlePlayer::InputJudge(KeyBoardState KeyState\_now, KeyBoardState KeyState\_last)

{

if (PlayerNumber == 1)

{

Button\_now.button\_Attack = KeyState\_now.Player1\_Attack;

Button\_now.button\_Down = KeyState\_now.Player1\_Down;

Button\_now.button\_Guard = KeyState\_now.Player1\_Guard;

Button\_now.button\_Jump = KeyState\_now.Player1\_Jump;

Button\_now.button\_Left = KeyState\_now.Player1\_Left;

Button\_now.button\_Right = KeyState\_now.Player1\_Right;

Button\_now.button\_Rush = KeyState\_now.Player1\_Rush;

Button\_now.button\_Skill = KeyState\_now.Player1\_Skill;

Button\_now.button\_Technique = KeyState\_now.Player1\_Technique;

Button\_now.button\_Up = KeyState\_now.Player1\_Up;

Button\_last.button\_Attack = KeyState\_last.Player1\_Attack;

Button\_last.button\_Down = KeyState\_last.Player1\_Down;

Button\_last.button\_Guard = KeyState\_last.Player1\_Guard;

Button\_last.button\_Jump = KeyState\_last.Player1\_Jump;

Button\_last.button\_Left = KeyState\_last.Player1\_Left;

Button\_last.button\_Right = KeyState\_last.Player1\_Right;

Button\_last.button\_Rush = KeyState\_last.Player1\_Rush;

Button\_last.button\_Skill = KeyState\_last.Player1\_Skill;

Button\_last.button\_Technique = KeyState\_last.Player1\_Technique;

Button\_last.button\_Up = KeyState\_last.Player1\_Up;

}

else if (PlayerNumber == 2)

{

Button\_now.button\_Attack = KeyState\_now.Player2\_Attack;

Button\_now.button\_Down = KeyState\_now.Player2\_Down;

Button\_now.button\_Guard = KeyState\_now.Player2\_Guard;

Button\_now.button\_Jump = KeyState\_now.Player2\_Jump;

Button\_now.button\_Left = KeyState\_now.Player2\_Left;

Button\_now.button\_Right = KeyState\_now.Player2\_Right;

Button\_now.button\_Rush = KeyState\_now.Player2\_Rush;

Button\_now.button\_Skill = KeyState\_now.Player2\_Skill;

Button\_now.button\_Technique = KeyState\_now.Player2\_Technique;

Button\_now.button\_Up = KeyState\_now.Player2\_Up;

Button\_last.button\_Attack = KeyState\_last.Player2\_Attack;

Button\_last.button\_Down = KeyState\_last.Player2\_Down;

Button\_last.button\_Guard = KeyState\_last.Player2\_Guard;

Button\_last.button\_Jump = KeyState\_last.Player2\_Jump;

Button\_last.button\_Left = KeyState\_last.Player2\_Left;

Button\_last.button\_Right = KeyState\_last.Player2\_Right;

Button\_last.button\_Rush = KeyState\_last.Player2\_Rush;

Button\_last.button\_Skill = KeyState\_last.Player2\_Skill;

Button\_last.button\_Technique = KeyState\_last.Player2\_Technique;

Button\_last.button\_Up = KeyState\_last.Player2\_Up;

}

}

void BattlePlayer::CheckHit(GPH)

{

//20190413已將此程式碼移植至AttackObj所屬

/\*

//可受傷狀態

if (this->Invincible == false)

{

map<string, AttackObj>::iterator iter;

for (iter = Enemy->Attacks.AttackObjects.begin(); iter != Enemy->Attacks.AttackObjects.end(); iter++)

{

if (iter->second.visable && (iter->second.IsHited == false || iter->second.CanCombo))

{

if (PixelCollision(&(this->BodyPicture), iter->second.DisplayBitmap, 2))

{

#pragma region 防禦狀態

if ((Action == "防禦" || Action == "防禦受傷") && (iter->second.BitmapisRight != IsRight) && iter->second.HitBreak == false)

{

IsRight = !(iter->second.BitmapisRight);

iter->second.IsHited = true;

if (iter->second.HitNoon == true)

{

iter->second.visable = false;

iter->second.DisplayBitmap->visable = false;

iter->second.Drawable = false;

}

PlaySounds(iter->second.HitSound, false);

Effects.BootEffect(&(Effects.Content[iter->second.HitEffect]), Camera, BodyRect.X + 3, BodyRect.X - 6, Rect.Y + 30, 0, 0, false, iter->second.BitmapisRight);

GainHP(-(iter->second.Damage / 3));

GainSP(-(iter->second.Damage / 10));

GainSP(-(iter->second.SP\_Damege / 2));

Velocity\_X = iter->second.Ahead(iter->second.HitVelocity\_X) / 3;

Velocity\_Y -= 0;

BeHitTimer = 0;

BeHitTimeMax = (iter->second.HitTime / 2.5);

Sleep(25);

Action = "防禦受傷";

Step = 0;

}

#pragma endregion

#pragma region 非防禦狀態或無法防禦

if (!((Action == "防禦" || Action == "防禦受傷") && (iter->second.BitmapisRight != IsRight) && iter->second.HitBreak == false))

{

IsRight = !(iter->second.BitmapisRight);

iter->second.IsHited = true;

if (iter->second.HitNoon == true)

{

iter->second.visable = false;

iter->second.DisplayBitmap->visable = false;

iter->second.Drawable = false;

}

PlaySounds(iter->second.HitSound, false);

Effects.BootEffect(&(Effects.Content[iter->second.HitEffect]), Camera, BodyRect.X + 3, BodyRect.X - 6, Rect.Y + 30, 0, 0, false, iter->second.BitmapisRight);

GainHP(-(iter->second.Damage));

GainSP(+(iter->second.Damage / 15));

GainSP(-iter->second.SP\_Damege);

if (((int)HP) > 0)

recovery = recovery + (iter->second.Damage / 1.5);

else

recovery = 0;

if (-iter->second.Attributes >= 0)

AttributeState[-iter->second.Attributes] = true;

Velocity\_X = iter->second.Ahead(iter->second.HitVelocity\_X);

Velocity\_Y = -(iter->second.HitVelocity\_Y);

BeHitTimer = 0;

BeHitTimeMax = iter->second.HitTime;

Sleep(25);

HitFly = iter->second.CanHitFly;

Step = 0;

Action = "受傷";

}

#pragma endregion

}

}

}

}

\*/

}

void BattlePlayer::GotoStandby(GPH)

{

Action = "待機";

Step = 0;

RunningTimer = 0;

StandbyTimer = 0;

RushTimer = 0;

}

void BattlePlayer::OnStandby(GPH)

{

if (Action == "待機")

{

StandbyTimer += TIMER\_TICK\_MILLIDECOND;

AddSP(StandbySPincrements);

#pragma region 處理摩擦力

if (Button\_now.button\_Left == false && Button\_now.button\_Right == false)

ProduceFriction(1, 1);

#pragma endregion

#pragma region 待機擺頭動作

if (StandbyTimer >= 500)

{

StandbyTimer = 0;

if (Step == 0)

Step = 1;

else if (Step == 1)

Step = 0;

}

#pragma endregion

#pragma region 到別的動作

#pragma region 左右移動

if (CanControl&&Button\_now.button\_Right&&OnGround)

{

IsRight = true;

GotoRunning(GPP);

}

else if (CanControl&&Button\_now.button\_Left&&OnGround)

{

IsRight = false;

GotoRunning(GPP);

}

#pragma endregion

CanToCharge;

CanToGuard;

CanToRush;

CanToJump;

CanToNormalAttack1;

CanToSkill1;

CanToUpAttack;

CanToDownAttack;

CanToUpSkill;

CanToUltimateSkill;

CanToDownSkill;

#pragma endregion

}

}

void BattlePlayer::GotoRunning(GPH)

{

Action = "移動";

Step = 0;

RunningTimer = 0;

}

void BattlePlayer::OnRunning(GPH)

{

if (Action == "移動")

{

AddSP(RunningSPincrements);

RunningTimer += TIMER\_TICK\_MILLIDECOND;

#pragma region 左右移動

if (CanControl&&IsRight&&OnGround && (Button\_now.button\_Right == true || Button\_now.button\_Left == true))

{

if (Button\_now.button\_Left == true)

IsRight = false;

if (RunningTimer >= 25)

{

RunningTimer = 0;

LoopStep(4);

}

RunAhead(0.75, RunSpeed);

}

else if (CanControl&&IsRight == false && OnGround && (Button\_now.button\_Right == true || Button\_now.button\_Left == true))

{

if (Button\_now.button\_Right == true)

IsRight = true;

if (RunningTimer >= 25)

{

RunningTimer = 0;

LoopStep(4);

}

RunAhead(0.75, RunSpeed);

}

#pragma endregion

#pragma region 到別的動作

CanToCharge;

CanToGuard;

CanToJump;

CanToNormalAttack1;

CanToRush;

CanToSkill1;

CanToUpAttack;

CanToUpSkill;

//正常結束(左右放開)

if (Button\_now.button\_Right == false && Button\_now.button\_Left == false)

CanToStandby;

#pragma endregion

}

}

void BattlePlayer::GotoRush(GPH)

{

if (this->SP >= Rush\_cost)

{

this->SP -= Rush\_cost;

Action = "衝刺";

Step = 0;

RushTimer = 0;

}

}

void BattlePlayer::OnRush(GPH)

{

}

void BattlePlayer::FastDrop(GPH)

{

if (SP >= 4)

{

GainSP(-4);

PlaySounds(Sounds.Rush, false);

PlayEffect(this, "Airboost3", Camera, Rect.X - 20, Rect.X - 20, Rect.Y);

Velocity\_Y = 12;

}

}

void BattlePlayer::GotoJump(GPH)

{

if (OnGround)

{

Action = "跳躍";

Step = 0;

RushTimer = 0;

JumpTimer = 0;

}

else

{

if (SP >= 10)

{

GainSP(-10);

Action = "跳躍";

RushTimer = 0;

Step = 3;

JumpTimer = 0;

Velocity\_Y = -12;

OnGround = false;

PlayEffect(this, "Airboost2", Camera, Rect.X - 30, Rect.X - 35, Rect.Y + 80);

PlaySounds(Sounds.Jump, false);

}

}

}

void BattlePlayer::OnJump(GPH)

{

if (Action == "跳躍")

{

JumpTimer += TIMER\_TICK\_MILLIDECOND;

#pragma region 跳躍主程序

if (JumpTimer >= 10 && Step < 2)

{

ProduceFriction(0.15, 1);

Step += 1;

JumpTimer = 0;

}

else if (JumpTimer >= 20 && Step == 2)

{

Step = 3;

JumpTimer = 0;

Velocity\_Y = -12;

OnGround = false;

PlayEffect(this, "Airboost2", Camera, Rect.X - 30, Rect.X - 35, Rect.Y + 80);

PlaySounds(Sounds.Jump, false);

}

else if (Step == 3 && Velocity\_Y < 0)

{

if (CanControl&&Button\_now.button\_Jump == true && Velocity\_Y < 2 && JumpTimer < 120)

Velocity\_Y -= 0.25;

}

else if (Velocity\_Y >= 0 && Step == 3)

{

JumpTimer = 0;

Step = 4;

}

#pragma endregion

#pragma region 起跳後

if (Step >= 3)

{

#pragma region 空中移動

if (CanControl&&Button\_now.button\_Right == false && CanControl&&Button\_now.button\_Left == false)

{

ProduceFriction(0.15, 1);

}

else if (CanControl&&Button\_now.button\_Right == true)

{

IsRight = true;

RunAhead(0.5, RunSpeed / 2);

}

else if (CanControl&&Button\_now.button\_Left == true)

{

IsRight = false;

RunAhead(0.5, RunSpeed / 2);

}

#pragma endregion

#pragma region 到別的動作

CanToAirAttack1;

CanToRush;

CanToSkill1;

CanToAirDownAttack;

CanToAirUpAttack;

CanToJump;

CanToFastDrop;

//正常落地

if (OnGround)

CanToStandby;

#pragma endregion

}

#pragma endregion

}

}

void BattlePlayer::GotoGuard(GPH)

{

if (this->SP > 5)

{

Action = "防禦";

Step = 0;

}

}

void BattlePlayer::OnGuard(GPH)

{

if (Action == "防禦")

{

if (SP <= 0 || Button\_now.button\_Guard == false)

{

GotoStandby(GPP);

}

else

{

ProduceFriction(1, 1);

SP -= GuardSPCost;

}

#pragma region 到別的動作

if (CanControl&&Button\_now.button\_Down == true && Button\_last.button\_Down == false)

{

GotoCharge(GPP);

}

#pragma endregion

}

}

void BattlePlayer::GotoCharge(GPH)

{

Action = "練氣";

Step = 0;

ChargeTimer = 0;

//啟動特效

PlayEffect(this, "SPCharge", Camera, Rect.X - 60, Rect.X - 60, Rect.Y - 55);

}

void BattlePlayer::OnCharge(GPH)

{

if (Action == "練氣")

{

ProduceFriction(1, 1);

ChargeTimer2 += TIMER\_TICK\_MILLIDECOND;

ChargeTimer += TIMER\_TICK\_MILLIDECOND;

if (ChargeTimer >= 50 && Step == 0)

{

Step = 1;

ChargeTimer = 0;

}

else if (ChargeTimer >= 16 && Step >= 1 && Step < 3)

{

Step += 1;

ChargeTimer = 0;

if (Step == 3)

{

ChargeTimer2 = 0;

Chargecount = 0;

PlaySounds(Sounds.SPCharge, false);

}

}

else if (ChargeTimer2 >= 10 && Step == 3)

{

Chargecount += 1;

if (Chargecount < 10)

{

SP += ChargeSPincrements / 10;

if (SP > SP\_Max)

SP = SP\_Max;

}

ChargeTimer2 = 0;

}

else if (ChargeTimer >= 90 && Step == 3)//正常結束

{

ChargeTimer = 0;

if (Button\_now.button\_Guard == true)

GotoGuard(GPP);

else

GotoStandby(GPP);

}

}

}

void BattlePlayer::OnHit(GPH)

{

NotHitTimer += TIMER\_TICK\_MILLIDECOND;

if (Action == "受傷")

{

OnHitTimer += TIMER\_TICK\_MILLIDECOND;

ProduceFriction(0.25, 0.35);

BeHitTimer += TIMER\_TICK\_MILLIDECOND;

if (Effects.Content["stun\_star"].visable == true)

{

Effects.Content["stun\_star"].Rect.X = Rect.X;

Effects.Content["stun\_star"].Rect.Y = Rect.Y;

Effects.Content["stun\_star"].loop = true;

}

if (HitFly)

{

Acceleration\_Y = -0.075;

}

if (BeHitTimer >= BeHitTimeMax&&HitFly == false)

{

BeHitTimer = 0;

BeHitTimeMax = 0;

OnGround = false;

HitFly = false;

GotoStandby(GPP);

Effects.Content["stun\_star"].loop = false;

}

if (HitFly == true)

{

if (Step == 0)

Step = 1;

if (BeHitTimer >= 80 && BeHitTimer <= 180 && Button\_now.button\_Jump&&Button\_last.button\_Jump == false && OnGround == false)

{

if (SP >= 10)

{

GainSP(-10);

Action = "跳躍";

RushTimer = 0;

Step = 3;

JumpTimer = 0;

Velocity\_Y = -10;

OnGround = false;

HitFly = false;

Invincible = false;

Throughing = false;

Effects.Content["stun\_star"].loop = false;

PlayEffect(this, "Airboost2", Camera, Rect.X - 30, Rect.X - 35, Rect.Y + 80);

PlaySounds(Sounds.Jump, false);

}

}

else if (BeHitTimer > 300 || (BeHitTimer > 100 && OnGround))

{

if (OnGround&&Velocity\_Y <= 14)

{

if (Step < 2)

{

BeHitTimer = 0;

Invincible = true;

Throughing = true;

Step = 2;

Acceleration\_Y = 0;

}

}

}

}

if (Step >= 2 && BeHitTimer < 1000 && BeHitTimer >= 200 && OnGround&&Button\_now.button\_Jump&&Button\_last.button\_Jump == false)

{

GainSP(8);

GotoJump(GPP);

Effects.Content["stun\_star"].loop = false;

Step = 3;

JumpTimer = 0;

Velocity\_Y = -1;

Velocity\_X /= 2;

OnGround = false;

HitFly = false;

BeHitTimer = 0;

Invincible = false;

Throughing = false;

HitFly = false;

PlayEffect(this, "Airboost2", Camera, Rect.X - 30, Rect.X - 35, Rect.Y + 80);

PlaySounds(Sounds.Jump, false);

Acceleration\_Y = 0;

}

if (Step >= 2 && BeHitTimer >= 1200)

{

BeHitTimer = 0;

Invincible = false;

Throughing = false;

HitFly = false;

GotoStandby(GPP);

Effects.Content["stun\_star"].loop = false;

Acceleration\_Y = 0;

}

if (NotHitTimer - OnHitTimer > 1500)

{

recovery = 0;

OnHitTimer = 0;

NotHitTimer = 0;

}

}

}

void BattlePlayer::OnHitGuard(GPH)

{

if (Action == "防禦受傷")

{

ProduceFriction(0.5, 0.5);

BeHitTimer += TIMER\_TICK\_MILLIDECOND;

if (BeHitTimer >= BeHitTimeMax)

{

BeHitTimer = 0;

BeHitTimeMax = 0;

if (Button\_now.button\_Guard)

{

if (SP > 0)

{

GotoGuard(GPP);

}

else

{

GotoStandby(GPP);

}

}

else

{

GotoStandby(GPP);

}

}

}

}

void BattlePlayer::AddSP(double mathin)

{

if (SP < SP\_Max)

{

SPincrementsTimer += TIMER\_TICK\_MILLIDECOND;

if (SPincrementsTimer >= 25)

{

SPincrementsTimer = 0;

SP += mathin;

if (SP > SP\_Max)

{

SP = SP\_Max;

}

}

}

}

void BattlePlayer::GainHP(double mathin)

{

HP += mathin;

if (HP > HP\_Max)

HP = HP\_Max;

else if (HP <= 0)

HP = 0;

}

void BattlePlayer::GainSP(double mathin)

{

SP += mathin;

if (SP > SP\_Max)

SP = SP\_Max;

else if (SP <= 0)

SP = 0;

}

void BattlePlayer::ProduceFriction(double power, double range)

{

if (Velocity\_X <= range && Velocity\_X >= -range)

{

Velocity\_X = 0;

}

else if (Velocity\_X > range)

{

Velocity\_X -= power;

}

else if (Velocity\_X < -range)

{

Velocity\_X += power;

}

}

void BattlePlayer::GotoDrop(GPH)

{

Action = "跳躍";

Step = 4;

JumpTimer = 0;

}

void BattlePlayer::LoopStep(int maxstep)

{

if (Step <= maxstep)

{

Step += 1;

if (Step == maxstep)

{

Step = 0;

}

}

}

void BattlePlayer::RunAhead(double Addspeed, double Maxspeed)

{

if (IsRight)

{

if (Velocity\_X < Maxspeed)

{

Velocity\_X += Addspeed;

}

}

else

{

if (Velocity\_X > -Maxspeed)

{

Velocity\_X -= Addspeed;

}

}

}

double BattlePlayer::Ahead(double move)

{

double returner = 0;

if (IsRight)

{

returner = move;

}

else

{

returner = -move;

}

return returner;

}

}

EffectSprite.h

#pragma once

#include "stdafx.h"

namespace game\_framework

{

class EffectSprite

{

public:

EffectSprite();

~EffectSprite();

map<string, BitmapAnimation> Content;//儲存所有特效

void EffectAutoUpdate(BitmapAnimation\*, int, bool, CameraPosition);//(讓特效自動更新狀態，放在Update裡)

//執行一個Effect(BitmapAnimation,Camera,XR,XL,Y,VX,VY,Track,左右)

void BootEffect(BitmapAnimation\*, CameraPosition, double, double, double,double,double,bool,bool);

void DrawAllEffection(int);//更新函式，且隨著視角移動

void AutoLoadEffections(COLORREF);//讀取遊戲中全部特效

void InsertEffection(string, int, int, double, COLORREF);//讀取遊戲中全部特效

};

}

EffectSprite.cpp

#pragma once

#include "stdafx.h"

#include "Resource.h"

#include <ddraw.h>

#include <windows.h>

#include "audio.h"

#include "gamelib.h"

#include "WKBitmap.h"

#include "BattlePlayer.h"

#include "EffectSprite.h"

using namespace std;

namespace game\_framework

{

EffectSprite::EffectSprite()

{

}

EffectSprite::~EffectSprite()

{

}

//所有特效都寫這

void EffectSprite::AutoLoadEffections(COLORREF color)

{

InsertEffection("Airboost", 5, 4, 10, color);

InsertEffection("Airboost2", 5, 4, 10, color);

InsertEffection("Airboost3", 5, 4, 10, color);

InsertEffection("SPCharge", 12, 4, 10, color);

InsertEffection("PunchHit", 3, 4, 16, color);

InsertEffection("ResetBody", 4, 4, 16, color);

InsertEffection("Disable", 4, 4, 16, color);

InsertEffection("stun\_star", 3, 4, 16, color);

InsertEffection("HitWall", 3, 4, 16, color);

InsertEffection("Matchstick\_US", 0, 6, 16, color);

InsertEffection("Rina\_US", 0, 6, 16, color);

InsertEffection("OraOraFire", 4, 4, 30, color);

}

void EffectSprite::EffectAutoUpdate(BitmapAnimation \* Effection, int tick, bool replay, CameraPosition Camera)

{

if (Effection->visable)

{

Effection->InSideCamera = true;

Effection->AutoPlay(tick, replay);

Effection->Rect.X += Effection->Velocity\_X;

Effection->Rect.Y += Effection->Velocity\_Y;

Effection->OnUpdate("Effects", Camera);

}

}

void EffectSprite::BootEffect(BitmapAnimation \*Effection, CameraPosition Camera, double XR, double XL, double Y, double VX, double VY, bool Track, bool IsRight)

{

Effection->AutoPlayTimer = 0;

Effection->Step = 0;

Effection->BitmapisRight = IsRight;

if (Effection->BitmapisRight)

Effection->Rect.X = XR;

else

Effection->Rect.X = XL;

Effection->Rect.Y = Y;

Effection->visable = true;

Effection->Velocity\_X = VX;

Effection->Velocity\_X = VY;

Effection->TrackPoint = Track;

EffectAutoUpdate(Effection, (int)(Effection->PreAutoFrequence), false, Camera);

}

void EffectSprite::DrawAllEffection(int i)

{

map<string, BitmapAnimation>::iterator iter;

for (iter = Content.begin(); iter != Content.end(); iter++)

if (iter->second.visable)

iter->second.DisplayBitmap->Draw(i, iter->second.drawlayer);

}

void EffectSprite::InsertEffection(string name, int maxstep, int drawlayer, double pre, COLORREF color)

{

Content.insert(std::pair<string, BitmapAnimation>(name, BitmapAnimation(false)));

Content[name].SetName(name);

Content[name].AutoLoadBitmaps("Effects", name, maxstep + 1, pre, false, color);

Content[name].drawlayer = drawlayer;

Content[name].OnUpdate();

}

}

SelectionBitmap.h

#pragma once

#include "TypeConverter.h"

using namespace std;

using namespace TypeConverter\_namespace;

namespace game\_framework

{

class SelectionBitmap : public BitmapAnimation

{

public:

SelectionBitmap();

~SelectionBitmap();

virtual void OnUpdate(int,int);

virtual void AutoLoadBitmaps(string, COLORREF);//依照Name自動讀取檔名

};

}

SelectionBitmap.cpp

#pragma once

#include "stdafx.h"

#include <ddraw.h>

#include <windows.h>

#include <stdio.h>

#include <sstream>

#include "gamelib.h"

#include "WKBitmap.h"

#include "TypeConverter.h"

#include "SelectionBitmap.h"

using namespace std;

using namespace TypeConverter\_namespace;

namespace game\_framework

{

SelectionBitmap::SelectionBitmap()

{

Step = 0;

AutoMaxStep = 0;

AutoPlayTimer = 0;

CanPixelCollision = false;

visable = false;

}

SelectionBitmap::~SelectionBitmap()

{

}

void SelectionBitmap::OnUpdate(int Nowselect, int Actselect)

{

if (Nowselect == Actselect)

this->Step = 1;

else

this->Step = 0;

string Actionstring = "Content\\Bitmaps\\Selection\\" + this->GetName() + "\_" + IntToString(Step) + ".bmp";

char \*cd = new char[65535];

strcpy(cd, Actionstring.c\_str());

DisplayBitmap = &BitmapPictures[cd];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

delete[] cd;

}

void SelectionBitmap::AutoLoadBitmaps(string name, COLORREF color)

{

this->SetName(name);

CanPixelCollision = false;

string str = ("Content\\Bitmaps\\Selection\\" + name + "\_0.bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*cc = new char[65535];

strcpy(cc, str.c\_str());

BitmapPictures[str].LoadTexture(cc, false, color);

DisplayBitmap = &BitmapPictures[cc];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

delete[] cc;

str = ("Content\\Bitmaps\\Selection\\" + name + "\_1.bmp");

BitmapPictures.insert(std::pair<string, BitmapPicture>(str, BitmapPicture(visable)));

char \*ca = new char[65535];

strcpy(ca, str.c\_str());

BitmapPictures[str].LoadTexture(ca, false, color);

DisplayBitmap = &BitmapPictures[ca];

Rect.Width = DisplayBitmap->Rect.Width;

Rect.Height = DisplayBitmap->Rect.Height;

Rect.X\_int = (int)(Rect.X);

Rect.Y\_int = (int)(Rect.Y);

DisplayBitmap->Rect.X = Rect.X\_int;

DisplayBitmap->Rect.Y = Rect.Y\_int;

DisplayBitmap->OnUpdate();

delete[] ca;

}

}

KeyBoardState.h

#pragma once

#include "Keycode.h"

using namespace std;

namespace game\_framework {

class KeyBoardState

{

public:

KeyBoardState();

~KeyBoardState();

bool ESC = false;

bool Backspace = false;

bool Tab = false;

bool Clear = false;

bool Enter = false;

bool Space = false;

bool Shift = false;

bool Control = false;

bool Alt = false;

bool CapsLock = false;

bool PageUp = false;

bool PageDown = false;

bool End = false;

bool Home = false;

bool Left = false;

bool Up = false;

bool Right = false;

bool Down = false;

bool Insert = false;

bool Del = false;

bool Help = false;

bool A = false;

bool B = false;

bool C = false;

bool D = false;

bool E = false;

bool F = false;

bool G = false;

bool H = false;

bool I = false;

bool J = false;

bool K = false;

bool L = false;

bool M = false;

bool N = false;

bool O = false;

bool P = false;

bool Q = false;

bool R = false;

bool S = false;

bool T = false;

bool U = false;

bool V = false;

bool W = false;

bool X = false;

bool Y = false;

bool Z = false;

bool Num\_1 = false;

bool Num\_2 = false;

bool Num\_3 = false;

bool Num\_4 = false;

bool Num\_5 = false;

bool Num\_6 = false;

bool Num\_7 = false;

bool Num\_8 = false;

bool Num\_9 = false;

bool Num\_0 = false;

bool Pad\_1 = false;

bool Pad\_2 = false;

bool Pad\_3 = false;

bool Pad\_4 = false;

bool Pad\_5 = false;

bool Pad\_6 = false;

bool Pad\_7 = false;

bool Pad\_8 = false;

bool Pad\_9 = false;

bool Pad\_0 = false;

bool Pad\_Enter = false;

void UpdateState\_Down(UINT btnIn);

void UpdateState\_Up(UINT btnIn);

};

}

KeyBoardState.cpp

#include "stdafx.h"

#include "KeyBoardState.h"

#include "Keycode.h"

namespace game\_framework {

KeyBoardState::KeyBoardState()

{

}

KeyBoardState::~KeyBoardState()

{

}

void KeyBoardState::UpdateState\_Down(UINT btnIn)

{

Keycode Keys;

if (btnIn == Keys.A)A = true;

else if (btnIn == Keys.B)B = true;

else if (btnIn == Keys.C)C = true;

else if (btnIn == Keys.D)D = true;

else if (btnIn == Keys.E)E = true;

else if (btnIn == Keys.F)F = true;

else if (btnIn == Keys.G)G = true;

else if (btnIn == Keys.H)H = true;

else if (btnIn == Keys.I)I = true;

else if (btnIn == Keys.J)J = true;

else if (btnIn == Keys.K)K = true;

else if (btnIn == Keys.L)L = true;

else if (btnIn == Keys.M)M = true;

else if (btnIn == Keys.N)N = true;

else if (btnIn == Keys.O)O = true;

else if (btnIn == Keys.P)P = true;

else if (btnIn == Keys.Q)Q = true;

else if (btnIn == Keys.R)R = true;

else if (btnIn == Keys.S)S = true;

else if (btnIn == Keys.T)T = true;

else if (btnIn == Keys.U)U = true;

else if (btnIn == Keys.V)V = true;

else if (btnIn == Keys.W)W = true;

else if (btnIn == Keys.X)X = true;

else if (btnIn == Keys.Y)Y = true;

else if (btnIn == Keys.Z)Z = true;

else if (btnIn == Keys.ESC)ESC = true;

else if (btnIn == Keys.Backspace)Backspace = true;

else if (btnIn == Keys.Tab)Tab = true;

else if (btnIn == Keys.Clear) Clear = true;

else if (btnIn == Keys.Enter)Enter = true;

else if (btnIn == Keys.Space) Space = true;

else if (btnIn == Keys.Shift)Shift = true;

else if (btnIn == Keys.Control)Control = true;

else if (btnIn == Keys.Alt)Alt = true;

else if (btnIn == Keys.CapsLock)CapsLock = true;

else if (btnIn == Keys.PageUp)PageUp = true;

else if (btnIn == Keys.PageDown) PageDown = true;

else if (btnIn == Keys.End)End = true;

else if (btnIn == Keys.Home) Home = true;

else if (btnIn == Keys.Left)Left = true;

else if (btnIn == Keys.Up)Up = true;

else if (btnIn == Keys.Right) Right = true;

else if (btnIn == Keys.Down)Down = true;

else if (btnIn == Keys.Insert)Insert = true;

else if (btnIn == Keys.Del)Del = true;

else if (btnIn == Keys.Help) Help = true;

else if (btnIn == Keys.Num\_1)Num\_1 = true;

else if (btnIn == Keys.Num\_2)Num\_2 = true;

else if (btnIn == Keys.Num\_3)Num\_3 = true;

else if (btnIn == Keys.Num\_4)Num\_4 = true;

else if (btnIn == Keys.Num\_5)Num\_5 = true;

else if (btnIn == Keys.Num\_6)Num\_6 = true;

else if (btnIn == Keys.Num\_7)Num\_7 = true;

else if (btnIn == Keys.Num\_8)Num\_8 = true;

else if (btnIn == Keys.Num\_9)Num\_9 = true;

else if (btnIn == Keys.Num\_0)Num\_0 = true;

else if (btnIn == Keys.Pad\_1)Pad\_1 = true;

else if (btnIn == Keys.Pad\_2)Pad\_2 = true;

else if (btnIn == Keys.Pad\_3)Pad\_3 = true;

else if (btnIn == Keys.Pad\_4)Pad\_4 = true;

else if (btnIn == Keys.Pad\_5)Pad\_5 = true;

else if (btnIn == Keys.Pad\_6)Pad\_6 = true;

else if (btnIn == Keys.Pad\_7)Pad\_7 = true;

else if (btnIn == Keys.Pad\_8)Pad\_8 = true;

else if (btnIn == Keys.Pad\_9)Pad\_9 = true;

else if (btnIn == Keys.Pad\_0)Pad\_0 = true;

else if (btnIn == Keys.Pad\_Enter)Pad\_Enter = true;

}

void KeyBoardState::UpdateState\_Up(UINT btnIn)

{

Keycode Keys;

if (btnIn == Keys.A)A = false;

else if (btnIn == Keys.B)B = false;

else if (btnIn == Keys.C)C = false;

else if (btnIn == Keys.D)D = false;

else if (btnIn == Keys.E)E = false;

else if (btnIn == Keys.F)F = false;

else if (btnIn == Keys.G)G = false;

else if (btnIn == Keys.H)H = false;

else if (btnIn == Keys.I)I = false;

else if (btnIn == Keys.J)J = false;

else if (btnIn == Keys.K)K = false;

else if (btnIn == Keys.L)L = false;

else if (btnIn == Keys.M)M = false;

else if (btnIn == Keys.N)N = false;

else if (btnIn == Keys.O)O = false;

else if (btnIn == Keys.P)P = false;

else if (btnIn == Keys.Q)Q = false;

else if (btnIn == Keys.R)R = false;

else if (btnIn == Keys.S)S = false;

else if (btnIn == Keys.T)T = false;

else if (btnIn == Keys.U)U = false;

else if (btnIn == Keys.V)V = false;

else if (btnIn == Keys.W)W = false;

else if (btnIn == Keys.X)X = false;

else if (btnIn == Keys.Y)Y = false;

else if (btnIn == Keys.Z)Z = false;

else if (btnIn == Keys.ESC)ESC = false;

else if (btnIn == Keys.Backspace)Backspace = false;

else if (btnIn == Keys.Tab)Tab = false;

else if (btnIn == Keys.Clear) Clear = false;

else if (btnIn == Keys.Enter)Enter = false;

else if (btnIn == Keys.Space) Space = false;

else if (btnIn == Keys.Shift)Shift = false;

else if (btnIn == Keys.Control)Control = false;

else if (btnIn == Keys.Alt)Alt = false;

else if (btnIn == Keys.CapsLock)CapsLock = false;

else if (btnIn == Keys.PageUp)PageUp = false;

else if (btnIn == Keys.PageDown) PageDown = false;

else if (btnIn == Keys.End)End = false;

else if (btnIn == Keys.Home) Home = false;

else if (btnIn == Keys.Left)Left = false;

else if (btnIn == Keys.Up)Up = false;

else if (btnIn == Keys.Right) Right = false;

else if (btnIn == Keys.Down)Down = false;

else if (btnIn == Keys.Insert)Insert = false;

else if (btnIn == Keys.Del)Del = false;

else if (btnIn == Keys.Help) Help = false;

else if (btnIn == Keys.Num\_1)Num\_1 = false;

else if (btnIn == Keys.Num\_2)Num\_2 = false;

else if (btnIn == Keys.Num\_3)Num\_3 = false;

else if (btnIn == Keys.Num\_4)Num\_4 = false;

else if (btnIn == Keys.Num\_5)Num\_5 = false;

else if (btnIn == Keys.Num\_6)Num\_6 = false;

else if (btnIn == Keys.Num\_7)Num\_7 = false;

else if (btnIn == Keys.Num\_8)Num\_8 = false;

else if (btnIn == Keys.Num\_9)Num\_9 = false;

else if (btnIn == Keys.Num\_0)Num\_0 = false;

else if (btnIn == Keys.Pad\_1)Pad\_1 = false;

else if (btnIn == Keys.Pad\_2)Pad\_2 = false;

else if (btnIn == Keys.Pad\_3)Pad\_3 = false;

else if (btnIn == Keys.Pad\_4)Pad\_4 = false;

else if (btnIn == Keys.Pad\_5)Pad\_5 = false;

else if (btnIn == Keys.Pad\_6)Pad\_6 = false;

else if (btnIn == Keys.Pad\_7)Pad\_7 = false;

else if (btnIn == Keys.Pad\_8)Pad\_8 = false;

else if (btnIn == Keys.Pad\_9)Pad\_9 = false;

else if (btnIn == Keys.Pad\_0)Pad\_0 = false;

else if (btnIn == Keys.Pad\_Enter)Pad\_Enter = false;

}

}

Keycode.h

#pragma once

using namespace std;

namespace game\_framework {

class Keycode

{

public:

Keycode();

~Keycode();

void testing();

const char ESC = 27;

const char Backspace = 8;

const char Tab = 9;

const char Clear = 12;

const char Enter = 13;

const char Space = 32;

const char Shift = 16;

const char Control = 17;

const char Alt = 18;

const char CapsLock = 20;

const char PageUp = 33;

const char PageDown = 34;

const char End = 35;

const char Home = 36;

const char Left = 37;

const char Up = 38;

const char Right = 39;

const char Down = 40;

const char Insert = 45;

const char Del = 46;

const char Help = 47;

const char A = 65;

const char B = 66;

const char C = 67;

const char D = 68;

const char E = 69;

const char F = 70;

const char G = 71;

const char H = 72;

const char I = 73;

const char J = 74;

const char K = 75;

const char L = 76;

const char M = 77;

const char N = 78;

const char O = 79;

const char P = 80;

const char Q = 81;

const char R = 82;

const char S = 83;

const char T = 84;

const char U = 85;

const char V = 86;

const char W = 87;

const char X = 88;

const char Y = 89;

const char Z = 90;

const char Num\_1 = 49;

const char Num\_2 = 50;

const char Num\_3 = 51;

const char Num\_4 = 52;

const char Num\_5 = 53;

const char Num\_6 = 54;

const char Num\_7 = 55;

const char Num\_8 = 56;

const char Num\_9 = 57;

const char Num\_0 = 48;

const char Pad\_1 = 97;

const char Pad\_2 = 98;

const char Pad\_3 = 99;

const char Pad\_4 = 100;

const char Pad\_5 = 101;

const char Pad\_6 = 102;

const char Pad\_7 = 103;

const char Pad\_8 = 104;

const char Pad\_9 = 105;

const char Pad\_0 = 96;

const char Pad\_Enter = 108;

};

}

Keycode.cpp

#include "stdafx.h"

#include "Keycode.h"

namespace game\_framework {

Keycode::Keycode()

{

}

Keycode::~Keycode()

{

}

}

CollisionSensor.h

#pragma once

using namespace std;

using namespace game\_framework;

namespace CollisionSensor\_namespace

{

bool BitmapPicture\_HitRectangle(BitmapPicture, BitmapPicture);

bool BitmapPicture\_HitRectangle(BitMapRectangle, BitMapRectangle);

bool PixelCollision(BitmapPicture \*, BitmapPicture \*, int);

bool PixelCollision(BitmapAnimation \*, BitmapAnimation \*, int);

}

CollisionSensor.cpp

#include "stdafx.h"

#include <ddraw.h>

#include <windows.h>

#include <stdio.h>

#include <sstream>

#include "gamelib.h"

#include "WKBitmap.h"

#include "BattlePlayer.h"

#include "CollisionSensor.h"

using namespace std;

using namespace game\_framework;

namespace CollisionSensor\_namespace

{

bool BitmapPicture\_HitRectangle(BitmapPicture Bitmap1, BitmapPicture Bitmap2)

{

if (Bitmap1.visable && Bitmap2.visable)

{

int x1 = Bitmap1.Rect.X\_int;

int x2 = Bitmap2.Rect.X\_int;

int y1 = Bitmap1.Rect.Y\_int;

int y2 = Bitmap2.Rect.Y\_int;

int w1 = Bitmap1.Rect.Width;

int w2 = Bitmap2.Rect.Width;

int h1 = Bitmap1.Rect.Height;

int h2 = Bitmap2.Rect.Height;

if (x1 >= x2 && x1 >= x2 + w2) {

return false;

}

else if (x1 <= x2 && x1 + w1 <= x2) {

return false;

}

else if (y1 >= y2 && y1 >= y2 + h2) {

return false;

}

else if (y1 <= y2 && y1 + h1 <= y2) {

return false;

}

return true;

}

else

{

return false;

}

}

bool BitmapPicture\_HitRectangle(BitMapRectangle Bitmap1, BitMapRectangle Bitmap2)

{

int x1 = (int)Bitmap1.X;

int x2 = (int)Bitmap2.X;

int y1 = (int)Bitmap1.Y;

int y2 = (int)Bitmap2.Y;

int w1 = (int)Bitmap1.Width;

int w2 = (int)Bitmap2.Width;

int h1 = (int)Bitmap1.Height;

int h2 = (int)Bitmap2.Height;

if (x1 >= x2 && x1 >= x2 + w2) {

return false;

}

else if (x1 <= x2 && x1 + w1 <= x2) {

return false;

}

else if (y1 >= y2 && y1 >= y2 + h2) {

return false;

}

else if (y1 <= y2 && y1 + h1 <= y2) {

return false;

}

else

{

return true;

}

}

bool PixelCollision(BitmapPicture \* Bitmap1, BitmapPicture \* Bitmap2, int accuracy)

{

if (BitmapPicture\_HitRectangle(\*Bitmap1, \*Bitmap2) == true)

{

if (Bitmap1->CanPixelCollision&&Bitmap2->CanPixelCollision&& Bitmap1->visable &&Bitmap2->visable &&accuracy > 0)

{

//決定碰撞四邊

int HitRight = 0;

int HitLeft = 0;

int HitTop = 0;

int Hitbottom = 0;

if (Bitmap1->Rect.X\_int + Bitmap1->Rect.Width > Bitmap2->Rect.X\_int + Bitmap2->Rect.Width)

{

HitRight = Bitmap2->Rect.X\_int + Bitmap2->Rect.Width;

}

else

{

HitRight = Bitmap1->Rect.X\_int + Bitmap1->Rect.Width;

}

if (Bitmap1->Rect.X\_int < Bitmap2->Rect.X\_int)

{

HitLeft = Bitmap2->Rect.X\_int;

}

else

{

HitLeft = Bitmap1->Rect.X\_int;

}

if (Bitmap1->Rect.Y\_int + Bitmap1->Rect.Height > Bitmap2->Rect.Y\_int + Bitmap2->Rect.Height)

{

Hitbottom = Bitmap2->Rect.Y\_int + Bitmap2->Rect.Height;

}

else

{

Hitbottom = Bitmap1->Rect.Y\_int + Bitmap1->Rect.Height;

}

if (Bitmap1->Rect.Y\_int < Bitmap2->Rect.Y\_int)

{

HitTop = Bitmap2->Rect.Y\_int;

}

else

{

HitTop = Bitmap1->Rect.Y\_int;

}

for (int i = 0; i < Hitbottom - HitTop; i += (accuracy))

{

for (int j = 0; j < HitRight - HitLeft; j += (accuracy))

{

if (Bitmap1->EffectRect[i + HitTop - Bitmap1->Rect.Y\_int][j + HitLeft - Bitmap1->Rect.X\_int] == true && Bitmap2->EffectRect[i + HitTop - Bitmap2->Rect.Y\_int][j + HitLeft - Bitmap2->Rect.X\_int] == true)

{

return true;

}

}

}

}

return false;

}

else

{

return false;

}

}

bool PixelCollision(BitmapAnimation \*Bitmap1, BitmapAnimation \*Bitmap2, int accuracy)

{

return PixelCollision(Bitmap1->DisplayBitmap, Bitmap2->DisplayBitmap, accuracy);

}

}

FunctionUser.h

#pragma once

#include <thread>

using namespace std;

using namespace game\_framework;

namespace FunctionUser\_namespace

{

void LoadingResource(void (LoadingFunction)(), thread \*mThread, bool \*started, bool \*finished);

void PlayEffect(BattlePlayer \*Master, string EffectName, CameraPosition Camera, double XR, double XL, double Y);

void PlayEffect(EffectSprite \*Effects, string EffectName, CameraPosition Camera, double XR, double XL, double Y,bool IsRight);

void DisableEffect(BattlePlayer \*Master, string EffectName);

void DisableEffect(EffectSprite \*Effects, string EffectName);

}

FunctionUser.cpp

#pragma once

#include "stdafx.h"

#include <ddraw.h>

#include <windows.h>

#include <stdio.h>

#include <sstream>

#include <thread>

#include "gamelib.h"

#include "WKBitmap.h"

#include "BattlePlayer.h"

#include "CollisionSensor.h"

#include "AttackObj.h"

#include "EffectSprite.h"

#include "FunctionUser.h"

using namespace std;

using namespace game\_framework;

namespace FunctionUser\_namespace

{

void LoadingResource(void (\*LoadingFunction)(), thread \*mThread, bool \*started, bool \*finished)

{

if (\*started == false && \*finished == false)

{

\*started = true;

\*mThread = thread(LoadingFunction);

}

if (\*finished == true && \*started == true)

{

mThread->join();

\*started = false;

}

}

void PlayEffect(BattlePlayer \*Master, string EffectName, CameraPosition Camera, double XR, double XL, double Y)

{

Master->Effects.BootEffect(&Master->Effects.Content[EffectName], Camera, XR, XL, Y, 0, 0, false, Master->IsRight);

}

void PlayEffect(EffectSprite \*Effects, string EffectName, CameraPosition Camera, double XR, double XL, double Y, bool IsRight)

{

Effects->BootEffect(&Effects->Content[EffectName], Camera, XR, XL, Y, 0, 0, false, IsRight);

}

void DisableEffect(BattlePlayer \* Master, string EffectName)

{

Master->Effects.Content[EffectName].loop = false;

Master->Effects.Content[EffectName].visable = false;

}

void DisableEffect(EffectSprite \* Effects, string EffectName)

{

Effects->Content[EffectName].loop = false;

Effects->Content[EffectName].visable = false;

}

}

TypeConverter.h

#pragma once

using namespace std;

namespace TypeConverter\_namespace

{

string IntToString(int);

int StringToInt(string);

}

TypeConverter.cpp

#include "stdafx.h"

#include <sstream>

#include "TypeConverter.h"

using namespace std;

namespace TypeConverter\_namespace

{

string IntToString(int intin)

{

stringstream SS = stringstream();

SS << intin;

string StepString;

SS >> StepString;

return StepString;

}

int StringToInt(string str)

{

return stoi(str);

}

}