

**組別：第 18 組**

**題目：Forager**

**組員：賴威錩 學號 106590007**

**游少謙 學號 106590033**

**授課老師：陳偉凱**

台北科大 資工系 2019年 物件導向程式設計實習報告 第18組 題目 Forager

**目錄**

[一、簡介： 2](#_Toc11318896)

[1、動機 2](#_Toc11318897)

[2、分工 2](#_Toc11318898)

[二、遊戲介紹： 2](#_Toc11318899)

[1、遊戲說明 2](#_Toc11318900)

[2、遊戲圖形 2](#_Toc11318901)

[3、遊戲音效 2](#_Toc11318902)

[三、程式設計 2](#_Toc11318903)

[1、程式架構 2](#_Toc11318904)

[2、程式類別 2](#_Toc11318905)

[3、程式技術 2](#_Toc11318906)

[四、結語 2](#_Toc11318907)

[1、問題及解決辦法 2](#_Toc11318908)

[2、時間表 2](#_Toc11318909)

[3、貢獻比例 2](#_Toc11318910)

[4、自我檢核表 2](#_Toc11318911)

[5、收穫 2](#_Toc11318912)

[6、心得、感想 2](#_Toc11318913)

[7、對於本課程的建議 2](#_Toc11318914)

[附錄 2](#_Toc11318915)

**一、簡介：**

1、動機

我們選擇製作的這一個遊戲是一款孤島生存遊戲，我們選擇此遊戲的原因是因為我們兩個人對於孤島生存這種類型的遊戲非常感興趣，平常都是遊玩其他團隊製作的生存遊戲，想說來試著做做看這種高難度的遊戲類型，剛剛好就看到最近網路上有一個非常火熱的遊戲，Forager，所以就選擇這款遊戲，一方面是對於生存遊戲的熱情，另一方面是想要試著做做看比較複雜的遊戲，進而磨練自己的程式設計能力。

2、分工

(1). 賴威錩：

引入Direct2D提供顯示功能、分離CDDraw、開頭選單製作、開頭選單效果、存檔系統、存檔快照、採集效果、採集通知與效果、圖片類別、Atlas圖片、PNG支援、文字顯示、文字邊框、地圖架構、畫面攝影機、血量系統與碰撞系統、MFC內建選單移除與更換滑鼠圖示與夜晚效果等工作。

(2). 游少謙：

引入XAudio2提供音效功能、背包系統、採集系統、建築系統、選單系統、角色系統、採集工具系統、遊戲內資源系統與基於FAtlas所開發之動畫類別等工作。

**二、遊戲介紹：**

1、遊戲說明

(1). 主要目的:

Forager是一款孤島生存遊戲，玩家藉由操控一個人類探索未知的世界。

(2). 操作方式:

W:向上走

A:向左走

S:向下走

D:向右走

B:開啟背包

E:與建築物互動

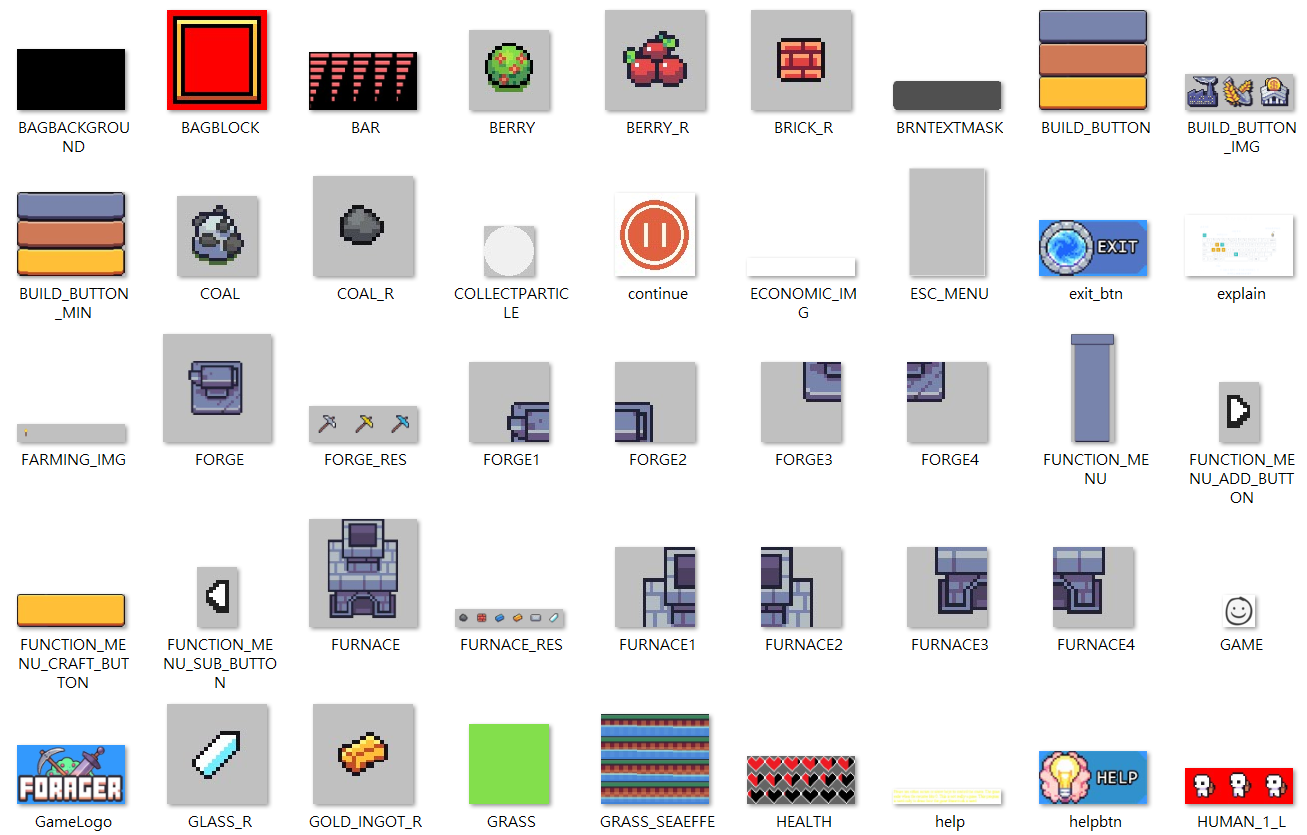
滑鼠左鍵:採集與操作選單

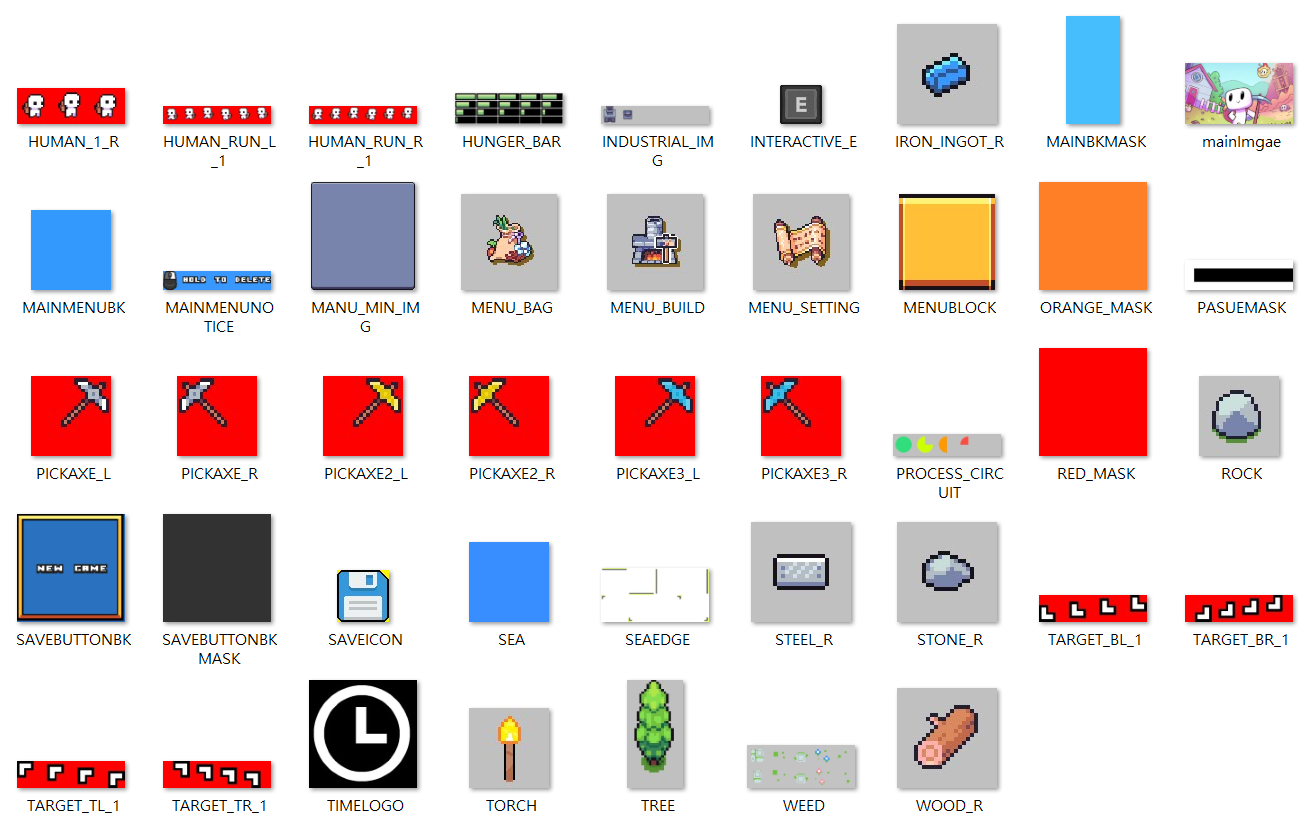
ESC:打開選單

(3). 遊戲方式:

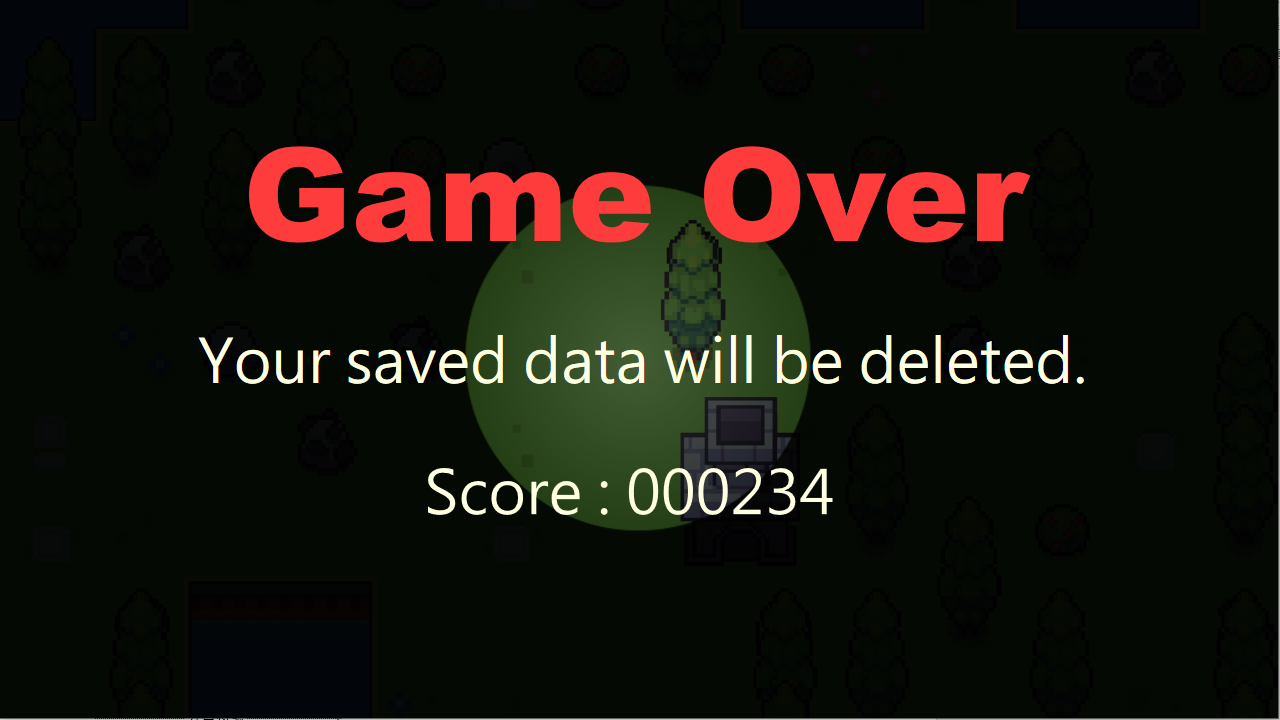
在遊戲開始時，玩家會出生在孤島的某個地方，基本配備就有十字鎬，所以就可以進行採集，遊戲運作的同時，角色的飢餓度也會緩慢下降，當飢餓度下降到0時，血量就會緩慢下降，當血量下降到0時，角色就會死亡，存檔就會被刪除，為了不要死亡，在探索孤島的同時，也要記得補充食物，案B打開包包使用滑鼠左鍵即可食用，在ESC選單上面由三個按鈕，分別是包包、建築、設定，包包是儲存角色採集或是合成的物品，建築是消耗包包內的資源來建造建築物，對著可互動的建築物案E即可互動，設定是調整遊戲相關設定以及離開至主選單並存檔，在主選單中，左鍵點擊空的存檔即可建立新存檔，左鍵點擊舊存檔即可進入舊存檔的遊戲，右鍵對舊存檔按住不動即可刪除存檔。

2、遊戲圖形







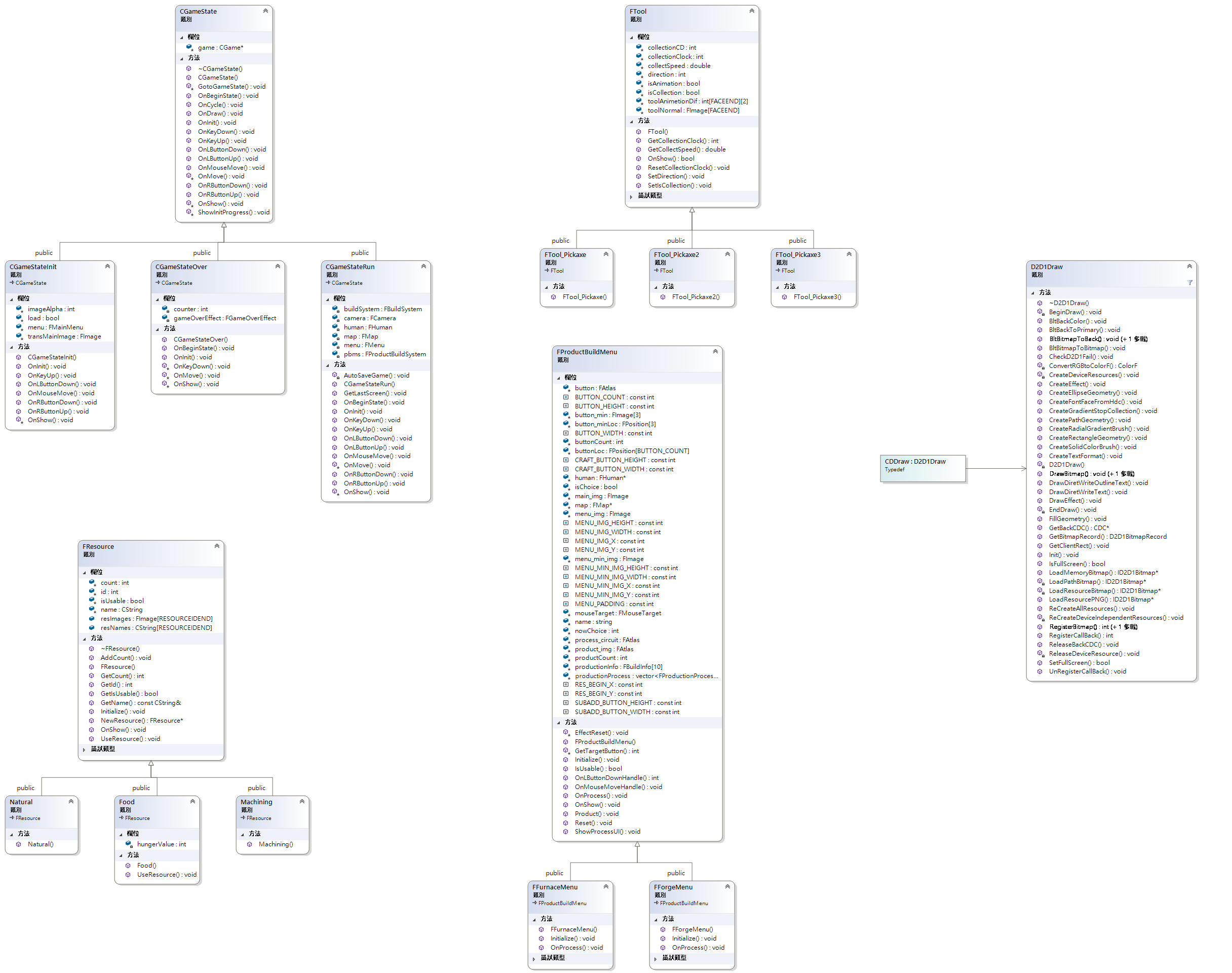


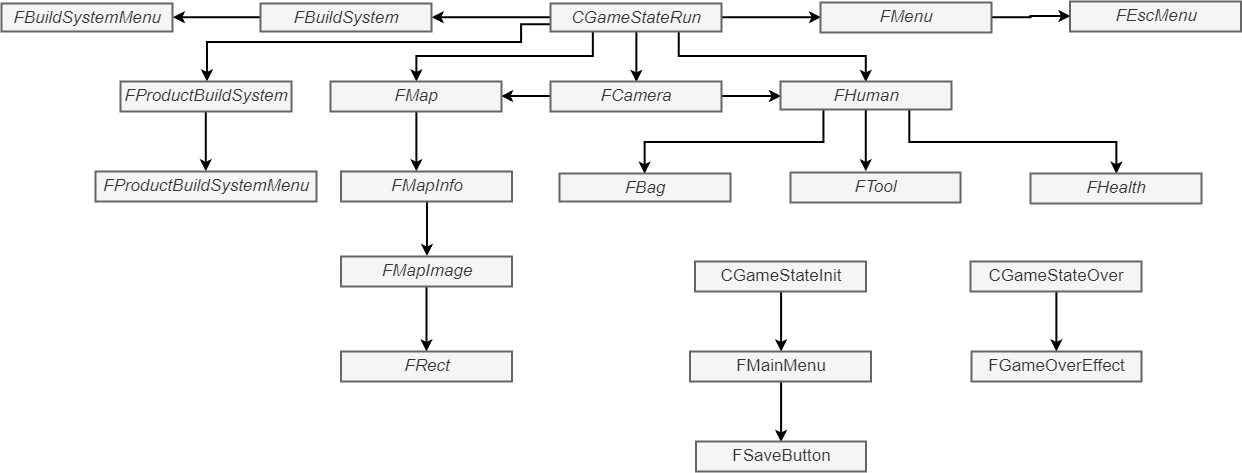
3、遊戲音效

|  |  |  |
| --- | --- | --- |
| **音效用途** | **音效名稱** | **音效來源** |
| 背景音樂 | Forager Original Soundtrack Wandering Around | https://youtu.be/xhsqSIk7hb0 |
| 採集音效 | Metal Strike on Wood (youtube 音效庫) | <https://www.youtube.com/audiolibrary/soundeffects?ar=3> |
| 受傷音效 | Minecraft Damage (Oof) Sound Effect (HD) | https://youtu.be/0T\_NR2KY8uI |
| 死亡音效 | Splat – Gaming Sound Effect (HD) | https://youtu.be/KUOfNtoJyLw |

**三、程式設計**

1、程式架構





2、程式類別

|  |  |  |  |
| --- | --- | --- | --- |
| **類別名稱** | **.h檔行數** | **.cpp檔行數** | **說明** |
| FAtlas | 29 | 49 | 可以顯示圖片部分區塊 |
| FBottomRightNotice | 42 | 87 | 採集資源提示 |
| FBottomRightNoticeEffect | 36 | 79 | 採集資源提示效果 |
| FCamera | 35 | 164 | 畫面顯示 |
| FCollectHighLightEffect | 33 | 62 | 資源採集高亮效果 |
| FCollectParticle | 30 | 73 | 資源採集粒子效果 |
| FGameOverEffect | 33 | 79 | 遊戲結束效果 |
| FGrassSeaEffect | 28 | 55 | 海浪效果等 |
| FHealth | 36 | 140 | 人物血量飢餓度管理 |
| FImage | 31 | 56 | 圖片處理類別 |
| FMainMenu | 65 | 227 | 開頭選單 |
| FMap | 69 | 354 | 地圖管理系統 |
| FMapImage | 54 | 60 | 地圖單元圖片 |
| FMapInfo | 52 | 52 | 地圖單元資訊 |
| FNightEffect | 51 | 179 | 夜間效果 |
| FOutlineText | 33 | 65 | 有邊框字體 |
| FPosition | 24 | 42 | 提供遊戲與螢幕座標轉換 |
| FRect | 24 | 25 | 提供碰撞檢測的矩形 |
| FSaveButton | 48 | 176 | 存檔按鈕 |
| FSaveImage | 37 | 75 | 存檔圖片 |
| FSaveSystem | 108 | 272 | 存檔系統 |
| FText | 31 | 39 | 文字顯示 |
| D2D1Draw | 130 | 635 | 圖形顯示 |
| D2D1FPSLimit | 33 | 58 | FPS 控制監控 |
| FAnimation | 36 | 67 | 以FAtlas為基礎撥放動畫 |
| FBag | 58 | 220 | 管理角色的包包 |
| FBuildInfo | 64 | 118 | 存放所有和成品與建築的資訊 |
| FBuildSystem | 77 | 179 | 建築系統的邏輯處理 |
| FBuildSystemMenu | 78 | 234 | 管理建築選單 |
| FCollectionBar | 35 | 72 | 採集進度條UI |
| FEscMenu | 62 | 231 | ESC選單 |
| FHuman | 91 | 353 | 角色移動與採集 |
| FMenu | 60 | 195 | 管理ESC選單,包包,建築界面 |
| FMouseTarget | 37 | 66 | 滑鼠Hover顯示 |
| FProductBuildMenu | 128 | 342 | 管理生產東西的建築選單 |
| FProductBuildSystem | 45 | 105 | 生產東西的建築邏輯處理 |
| FResource | 71 | 97 | 管理所有資源 |
| FTool | 66 | 106 | 管理角色拿的工具 |
| XAudio2Engine | 75 | 292 | 播放聲音 |
| **總行數** | **2075** | **5780** |  |

3、程式技術

Game framework 4.10 採用DirectDraw與GDI作為顯示核心架構。起初我們也延續此架構，但到第一次DEMO後發現，他們內建的功能，不足以勝任我們這個遊戲的顯示需求。例如圖片混和功能(顯示帶有透明度的圖片)，最早我們是透過CImage類別載入圖片文件，取得每個像素的RGB帶入ALPHA公式去進行圖片混和的工作，將低速的Set(Get)Pixel，換成指標直接存取，結果效能還是慘不忍睹。後來則是使用GDI內建的AlphaBlend進行這項任務，在內顯電腦工作良好，獨立顯示卡環境也是慘不忍睹。故在第二次DEMO前，大膽引入Direct2D來進行圖片顯示的重責大任。

我們本身並沒有DirectX家族的任何開發經驗，故在引入初期，花費了相當大的時間閱讀Windows開發人員中心的文章，了解Direct2D基本資源管理觀念與基本初始化工作等。為了維持課堂的公平性，我們堅持不引入別人在網路上已經封裝好寫好的CODE，僅使用微軟的內建函式庫來達成，所以參考Game framework 4.10內建的CDDraw類別，設計出D2D1Draw類別將Direct2D封裝。為了相容Game framework 4.10也花了一些功夫，把在CDDraw類別中的Public Function對應到D2D1Draw類別裡面實現。也就是把DirectDraw這個顯示的重責大任交給Direct2D進行處理。

也因為得到Direct2D硬體加速的幫忙之下，很多原本效能慘不忍睹的畫面效果都得以產生，例如圖片透明混和、旋轉、縮放與DirectWrite文字等顯示效果。雖然畫面效果良好，但也產生大量資源管理的問題，在管理這方面也花了很多時間。

在音效的部分，原本也是想使用Game framework 4.10內建的CAudio，但是同學使用時說好像會延遲，再加上CAudio沒有提供調整音量的Interface，所以我們就使用微軟內建的函式庫XAudio2為基礎，以提供音效處理功能。

一開始在撥放時都非常順利，但後來才發現一次把整個檔案的資料全部讀進來太佔記憶體了，所以後來選擇使用流式讀取(Streaming)方式來讀取檔案，就是開一個執行緒來對檔案進行分段讀取，進而節省記憶體空間。由於時間不足，所以我們只開發出讀取WAV檔的方法，並沒有支援壓縮格式(如:MP3)，也不能對正在撥放的音效進行處理。

**四、結語**

1、問題及解決辦法

(1). 當需要大量讀取圖片的像素時，使用CImage圖片類別內建的Set(Get)Pixel方法速度非常慢，可以使用GetBits取像素資料指標，直接操像素值。需要注意資料儲存的順序是BGRA。

(2). 需要顯示透明圖片時，可以透過GDI的AlphaBlend進行顯示工作，不需要自己計算現在螢幕像素與圖片像素的透明關係。

(3). 在Direct2D的環境下，可以透過DXGI交換鏈取得現在的緩衝，再利用緩衝內建的GetDC副程式取得HDC，來重現CDDraw的GetBackCDC副程式，但做這件事情要記得在初始化時，加入DXGI\_SWAP\_CHAIN\_FLAG\_GDI\_COMPATIBLE旗標，才能支援GDI環境。

(4). DirectWrite的FontFace初始化有點麻煩，這時候可以透過GDI的HFont來初始化FontFace。

(5). Direct2D讀取圖片需要使用Windows Imaging Component來運作。可以使用GDI圖片類別HBITMAP中的GetDIBits，取得圖片像素資料到陣列中，再透過Direct2D內建的從陣列的像素資料創建圖片資源的方式載入圖片。

(6). 某些Direct2D資源可以透過Callback的模式，將他放在其他類別儲存，在需要重新創建資源的時候，透過呼叫CallBack Function來達到管理的目地。

(7). 有些時候會發生GDI Object，不小心忘了釋放的問題。這時候可以透過多次執行測試那段程式碼，並觀察工作管理員內顯示的GDI物件，是否有爆掉的情形，舉例來說Windows10預設上限是10000個GDI物件，超過這個數字程式會有崩潰的現象。

(8). 使用XAudio2播放音效時，如果一次存取整個檔案會占用非常多的記憶體，所以我們使用流式讀取(Streaming)來進行檔案讀取，讓讀取變成一段一段分開讀取。

(9). 使用XAudio2播放音效時，每一段音效剛開始播放時，都會有一種喇叭爆音的聲音，只要在播放前先把聲音靜音，等播放一校段時間後，再將聲音提升為正常的聲音，就可以解決，但是缺點是會失去一小部分的聲音。

(10). 使用XAudio2播放音效時，如果沒有連接音頻播放設備，IXAudio2MasteringVoice就會有錯誤訊息，而且也不會創建資源，目前使用的辦法是，若發現IXAudio2MasteringVoice創建錯誤，就開一個執行緒，每隔一段時間就創建一次，若創建完成就退出執行緒，繼續播放音效;但是若原本已經成功創建，之後再把播放設備移除，就會發生有這個資源，但是聲音卻無法播放出去的問題，而且也不會產生錯誤訊息，解決辦法為，因為將音頻資料推送到緩衝區有設一個極限值，當IXAudio2MasteringVoice沒有作用時，音頻就不會播放，緩衝區的數量也會卡在極限值，所以就在這邊設一個檢查時間，若超過時間則停止播放音效，進行檢查IXAudio2MasteringVoice的工作。

2、時間表

|  |  |  |  |
| --- | --- | --- | --- |
| **週次** | **賴威錩(小時)** | **游少謙(小時)** | **說明** |
| 1 | 3 | 0 | 熟悉遊戲框架操作 |
| 2 | 4 | 1 | 熟悉遊戲框架操作 |
| 3 | 10 | 8 | 地圖雙向捲動 |
| 4 | 10 | 2.5 | 海不能走，樹也不能走 |
| 5 | 18 | 10 | 新增樹的class, (砍樹) |
| 6 | 15 | 14 | 人物動畫;圖片透明效果 |
| 7 | 16 | 10 | 改善卡頓問題;新增資源Hover顯示 |
| 8 | 10 | 12 | 血量顯示;十字鎬動畫 |
| 9 | 13 | 14 | 資源重生;地圖系統優化;簡易建築 |
| 10 | 22 | 12.5 | 包包與建築介面更新;地圖景物改善 |
| 11 | 18 | 15 | 存檔系統、建築系統 |
| 12 | 10 | 0 | 優化部分系統 |
| 13 | 13 | 11 | 解決掉偵問題;建築互動系統完善 |
| 14 | 13 | 10 | 鍛造台、白天晚上 |
| 15 | 10 | 10 | 火炬;火炬夜晚效果; |
| 16 | 9 | 12 | 音效; 關於我畫面 |
| 17 |  | | 實驗報告製作 |
| **合計** | **194** | **129** |  |

3、貢獻比例

**賴威錩：50%**

**游少謙：50%**

4、自我檢核表

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 項目 | 是否完成 | 無法完成的原因 |
| 1 | 解決Memory leak | ■ 已完成  未完成 |  |
| 2 | 自訂遊戲Icon | ■ 已完成  未完成 |  |
| 3 | 全螢幕啟動 | ■ 已完成  未完成 |  |
| 4 | 有About畫面 | ■ 已完成  未完成 |  |
| 5 | 初始畫面說明案件及滑鼠之用法語密技 | ■ 已完成  未完成 |  |
| 6 | 上傳setup/apk/source檔 | ■ 已完成  未完成 |  |
| 7 | Setup檔可正確執行 | ■ 已完成  未完成 |  |
| 8 | 報告字型、點數、對齊、行距、頁碼等格式正確 | ■ 已完成  未完成 |  |
| 9 | 報告封面、側邊格式正確 | ■ 已完成  未完成 |  |
| 10 | 報告附錄程式格式正確 | ■ 已完成  未完成 |  |

5、收穫

(1). 賴威錩：

學到遊戲的基本流程、類別必須封裝的好才能讓別人方便使用、一些MFC與WIN32API的觀念、使用GDI進行繪圖的方法、Direct2D的基本使用、DXGI概念與基本的使用、C++多執行續、運用之前學到的C++觀念與static修飾詞的多種用法等。

(2). 游少謙：

學到了許多更有邏輯的程式寫法，比上學期更熟悉了C++的語法，了解到visual studio 組態Debug, Release的區別，實用列舉(enum)，我以前學到一直都不知道他有甚麼用處，static, const的使用時機，了解到哪些地方可能會影響到程式碼的效能，如call by reference, 多重迴圈等等，XAudio2的基礎概念及運作方式和使用，C++的多執行緒。

6、心得、感想

(1). 賴威錩：

時光飛逝，又到了期末的時刻，這門課程與我當初想像得有些出入。我當初以為我可以在這門課程，更深入了解物件導向的精隨。但事實並非如此，程式設計的功力一定是有所長進，這點無庸置疑。但是，因為當初遊戲起步的時候，所撰寫的程式碼非常的C-style，不夠C++，也因此連帶影響後面程式碼。

這個遊戲所寫的Code，並不是非常優良。起因在於剛開始的時候，我們都沒有一個足夠的經驗與好的想法，去建構我們的程式基礎架構。導致在後面撰寫程式的時候，有些的方總是不如人意，用了一些比較不好的方式，這點是我覺得非常可惜的一件事情，但這是一件歷史共業，不是任何人造成的，我當然也能坦然接受。

遊戲框架提供一個快速的方式，讓使用者可以快速開發遊戲。但我認為應付基本的需求，他確實是可行的。但在一些場合上，他的表現差強人意，例如對圖片格式支援度僅支援BMP，不支持也是很流行的JPG與PNG等格式。還有本身沒有提供圖形旋轉、圖形透明度與文字輸出等工作。

起初我們也和大多數的同學一樣，使用GDI或是自幹副程式進行像素的處理來解決我們的問題，但他在內顯環境工作良好，獨立顯卡環境卻慘不忍睹。故我們採取微軟推行但推不太起來的的Direct2D來取代DirectDraw；DirectWrite來取代GDI文字處理的工作。也多虧了硬體加速的福音，他們的工作速度非常良好，同時支援抗鋸齒等特性。

也因為沒有相關的運用經驗，故對於引入這些東西，是非常困難的。但所謂天下無難事，只怕有心人。在花費了眾多的時間與精力後，成功的將其運用在這個專題上。隨然封裝Direct2D相關的Code寫的不是非常理想，但畢竟是初體驗。且因為沒有經過大量的測試，所以也有一些小問題。但至少他在大部分的時間可以正常工作，這點是非常讓人欣慰的。那透過瀏覽遊戲框架的Code，我也了解到說，一個基本的遊戲流程，大概是怎麼樣運作的，對沒有撰寫過遊戲的我來說，也是一件新奇的事物。

還有就是，MFC自帶的選單，實在是不太美觀。所以我也花了一些時間去了解一下。還有Windows透過消息和程式通訊等相關的東西。這也是很新奇的。

總之，這個學期我來說，算是有一個非常奇妙的體驗。當然花費大量的時間，真的是一件非常累人的事情，也導致常常發生睡眠不足，而產生遲到的情形。我想這是非常不好的事情，我沒有比別人聰明，但我敢花時間，但以後應該要多多衡量自己的能耐，以免斷送年輕的生命。

(2). 游少謙：

其實在這學期開始之前，我就已經非常期待這堂課。因為之前寫的東西都是在為我們打好基礎，所以功能上來說沒有很實用，終於有機會可以實用二年級上學期學習的物件導向程式設計，還可以寫出一款遊戲。而且我也找到我非常喜歡的組員，理所當然的我非常期待，一直期待著我們可以做出什麼樣的遊戲。

但是才剛開始，我就了解到這堂課不是我想像中的那麼美好，在和組員討論程式架構時，我聽得懵懵懂懂，然後也給不出什麼好的意見，這讓我深受打擊。一方面是感受到我與組員的差距，另一方面是對於想法貧窮的我感到憤怒，不過我還是想要做好，所以就一邊聽著組員的指導一步一步做。

在中期的時候，我的組員引入Direct2D來提供顯示功能，這東西在同學之間被說是黑科技，我也這麼認為，我相信寫出這個東西一定要有足夠的專注力、知識、耐力，我也非常佩服他。不過身為他的組員不免會被拿來比較，這讓我越來越消極。總覺得沒有我好像也沒什麼差，這讓我寫東西有一次沒一次的，甚至還Delay到Demo的進度，最後可能是組員也看我沒在做事，就叫我寫XAudio2用來播放音效。

但是還是拖了好幾個禮拜，最後是他一直催我，我才寫出一些可以動的東西，我相信他來寫應該兩三天就寫完了，不過他可能是想讓我體驗自己寫出一些東西的成就感吧，而且寫出一個可以動的東西真的是蠻爽的，雖然網路上教學看一看就寫得出來了，不過我真的很感謝我的組員，在我寫不好的時候糾正我，還教了我許多東西，也讓我學到很多東西。

7、對於本課程的建議

(1). 賴威錩：

建議遊戲框架應該要提供更友善的功能，如旋轉圖片、透明圖片、GDI文字應該要封裝與其他的圖片格式等，方便同學開發遊戲時，能夠更快的上手，與更有質感的遊戲。建議應該要將微軟已經拋棄的DirectDraw換掉，提供同學製作的遊戲更長的生命週期。

(2). 游少謙：

音效方面應該要增加可以調整音量的方法，讓學生在開發時可以做出更有層次更多變化的音效控管，再來就是我們組員提到的，旋轉圖片，透明功能，這樣學生在開發遊戲時，不僅會有更漂亮的遊戲畫面，學生做得好，教授也樂得開心。

附錄

/\* FileName: CGameStateInit.cpp \*/

﻿#include "stdafx.h"

#include "CGameStateInit.h"

#include "D2D1Draw.h"

namespace game\_framework

{

bool CGameStateInit::load = true;

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

{

}

void CGameStateInit::OnInit()

{

ShowInitProgress(25);

transMainImage.LoadBitmap(IDB\_MAINIMAGE);

transMainImage.SetTopLeft(0, 0, SIZE\_X, SIZE\_Y);

menu.Initialize();

}

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

UINT nFlags)

{

}

void CGameStateInit::OnMouseMove(UINT nFlags, CPoint point)

{

menu.MouseMoveHandle(point);

}

void CGameStateInit::OnLButtonDown(UINT nFlags, CPoint point)

{

if(!load && menu.OnLButtonDownHandle(point))

{

GotoGameState(GAME\_STATE\_RUN); // 切換至GAME\_STATE\_RUN

}

}

void CGameStateInit::OnShow()

{

static int step = 5;

static FLOAT scale = 1.5f;

if(load && imageAlpha <= 255 && imageAlpha >= 0)

{

imageAlpha += step;

if(scale > 1.0f)

scale += step \* 0.01f;

transMainImage.SetAlpha(imageAlpha);

transMainImage.SetScale(scale);

transMainImage.SetRotation(imageAlpha / 255.0f \* 30.0f);

}

if(imageAlpha <= 0)

load = false;

if(imageAlpha >= 255)

step = -10;

if(step < 0)

menu.OnShow();

if(load)

transMainImage.ShowBitmap();

/\*CPngImage image;

image.Load(IDB\_EXPLANPNG, AfxGetInstanceHandle());

CDC\* pDc = D2D1Draw::GetBackCDC();

CDC mDC;

mDC.CreateCompatibleDC(pDc);

mDC.SelectObject(image);

pDc->AlphaBlend(0, 0, 1280, 720, &mDC, 0, 0, 1280, 720, \_BLENDFUNCTION{ AC\_SRC\_OVER ,0, 255, AC\_SRC\_ALPHA });

D2D1Draw::ReleaseBackCDC();\*/

}

void CGameStateInit::OnRButtonDown(UINT nFlags, CPoint point)

{

menu.OnRButtonDownHandle(point);

}

void CGameStateInit::OnRButtonUp(UINT nFlags, CPoint point)

{

menu.OnRButtonUpHandle(point);

}

}

/\* FileName: CGameStateInit.h \*/

﻿#ifndef FORAGER\_CGAMESTATEINIT\_H

#define FORAGER\_CGAMESTATEINIT\_H

#include "Forager.h"

#include "FImage.h"

#include "CGameStdLib.h"

#include "FMainMenu.h"

namespace game\_framework

{

class CGameStateInit : public CGameState

{

public:

CGameStateInit(CGame\* g);

void OnInit();

void OnKeyUp(UINT, UINT, UINT);

void OnLButtonDown(UINT nFlags,

CPoint point);

void OnMouseMove(UINT nFlags, CPoint point);

void OnRButtonDown(UINT nFlags, CPoint point);

void OnRButtonUp(UINT nFlags, CPoint point);

protected:

FImage transMainImage;

FMainMenu menu;

int imageAlpha = 75;

static bool load;

void OnShow();

private:

};

}

#endif // !FORAGER\_CGAMESTATEINIT\_H

/\* FileName: CGameStateOver.cpp \*/

﻿#include "stdafx.h"

#include "CGameStateOver.h"

namespace game\_framework

{

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

{

}

void CGameStateOver::OnMove()

{

counter--;

}

void CGameStateOver::OnBeginState()

{

counter = 30 \* 10; // 5 seconds

FSaveSystem::DeleteSaveFile(FSaveSystem::saveID);

gameOverEffect.Initialize();

}

void CGameStateOver::OnInit()

{

ShowInitProgress(75);

}

void CGameStateOver::OnShow()

{

gameOverEffect.OnShow();

if(counter < 0)

game\_framework::CGame::Instance()->SetReset();

}

void CGameStateOver::OnKeyDown(UINT nChar, UINT nRepCnt,

UINT nFlags)

{

if(30 \* 10 - counter > 30 \* 2)

counter = 0;

}

}

/\* FileName: CGameStateOver.h \*/

﻿#ifndef FORAGER\_CGAMESTATEOVER\_H

#define FORAGER\_CGAMESTATEOVER\_H

#include "Forager.h"

#include "CGameStdLib.h"

#include "FGameOverEffect.h"

namespace game\_framework

{

class CGameStateOver : public CGameState

{

public:

CGameStateOver(CGame\* g);

void OnBeginState();

void OnInit();

protected:

void OnMove();

void OnShow();

void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);

private:

int counter;

FGameOverEffect gameOverEffect;

};

}

#endif // !FORAGER\_CGAMESTATEOVER\_H

/\* FileName: CGameStateRun.cpp \*/

﻿#include "stdafx.h"

#include "FGameOverEffect.h"

#include "CGameStateRun.h"

namespace game\_framework

{

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

{

}

void CGameStateRun::AutoSaveGame()

{

static int delay = 0;

delay++;

if(delay < 30 \* 10)

return;

else

delay = 0;

camera.ShowBackground();

camera.ShowForeground();

D2D1Draw::BltBackToPrimary();

FSaveSystem::SaveGame(&map, &human);

}

void CGameStateRun::OnBeginState()

{

const int ANIMATION\_SPEED = 15;

if(FSaveSystem::IsExistSaveFile(FSaveSystem::saveID))

FSaveSystem::LoadSaveGame(&map, &human);

else

FSaveSystem::ResetTime();

OnShow();

D2D1Draw::BltBackToPrimary();

FSaveSystem::SaveGame(&map, &human);

}

void CGameStateRun::OnMove() // 移動遊戲元素

{

if(human.nowMouseMode == FHuman::COLLECTION)

human.OnMove();

AutoSaveGame();

map.AutoCreateResource();

map.AutoCollectEffectReset();

map.AutoSetSeaEffect();

pbms.OnProcess();

if(human.humanHealth.GetGameOver())

{

camera.HideHuman();

D2D1Draw::BltBackToPrimary();

GetLastScreen();

GotoGameState(GAME\_STATE\_OVER);

}

}

void CGameStateRun::GetLastScreen()

{

CDC\* pCDC = D2D1Draw::GetBackCDC();

CDC memDC;

memDC.CreateCompatibleDC(pCDC);

CBitmap saveBmp;

saveBmp.CreateCompatibleBitmap(pCDC, SIZE\_X, SIZE\_Y);

CBitmap\* pOldBitmap = memDC.SelectObject(&saveBmp);

memDC.BitBlt(0, 0, SIZE\_X, SIZE\_Y, pCDC, 0, 0, SRCCOPY);

memDC.SelectObject(pOldBitmap);

HBITMAP hBmp = (HBITMAP)saveBmp;

tagBITMAP bm;

::GetObject(hBmp, sizeof(BITMAP), &bm);

tagBITMAPINFO bi;

memset(&bi, 0, sizeof(BITMAPINFO));

bi.bmiHeader = tagBITMAPINFOHEADER{ sizeof(bi.bmiHeader), bm.bmWidth, -bm.bmHeight, bm.bmPlanes, bm.bmBitsPixel, 0, 0, 0, 0, 0, 0 };

UINT pBuffLen = bm.bmWidth \* bm.bmHeight \* 4;

::GetDIBits(pCDC->m\_hDC, hBmp, 0, bm.bmHeight,

(void\*)(FGameOverEffect::pBuffer), &bi, DIB\_RGB\_COLORS);

for(UINT i = 0; i < pBuffLen; i += 4)

FGameOverEffect::pBuffer[i + 3] = 255;

saveBmp.DeleteObject();

memDC.DeleteDC();

D2D1Draw::ReleaseBackCDC();

}

void CGameStateRun::OnInit() // 遊戲的初值及圖形設定

{

ShowInitProgress(50);

FResource::Initialize();

map.Initialize();

human.Initialize(&map);

camera.Initialize(&map, &human);

buildSystem.Initialize(&map, &human);

pbms.Initialize(&human, &map);

menu.Initialize(&human, &buildSystem, &pbms);

}

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

UINT nFlags)

{

menu.OnKeyDownHandle(nChar);

human.SetMouseMode(menu.IsUsing());

human.OnKeyDownHandle(nChar);

}

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

UINT nFlags)

{

human.OnKeyUpHandle(nChar);

}

void CGameStateRun::OnLButtonDown(UINT nFlags, CPoint point)

{

if(human.nowMouseMode == FHuman::COLLECTION)

human.OnLButtonDownHandle(point);

else

menu.OnLButtonDownHandle(point);

}

void CGameStateRun::OnLButtonUp(UINT nFlags, CPoint point)

{

human.OnLButtonDownUp(point);

}

void CGameStateRun::OnMouseMove(UINT nFlags, CPoint point)

{

if(human.nowMouseMode == FHuman::COLLECTION)

human.OnMouseMoveHandle(point);

else

menu.OnMouseMoveHandle(point);

}

void CGameStateRun::OnShow()

{

camera.CalculatedPosition();

camera.ShowBackground();

// 滑鼠效果

human.MouseTargetShow();

human.ShowBar();

camera.ShowForeground();

// 顯示血量、右下角通知

human.humanHealth.OnShow();

human.bag.buttomRightNotice.OnShow();

pbms.ShowIteractiveE();

pbms.ShowProcessUI();

menu.OnShow();

FSaveSystem::OnShow();

if(menu.GetExitGameSignal())

{

camera.ShowBackground();

camera.ShowForeground();

D2D1Draw::BltBackToPrimary();

FSaveSystem::SaveGame(&map, &human);

game\_framework::CGame::Instance()->SetReset();

}

}

}

/\* FileName: CGameStateRun.h \*/

﻿#ifndef FORAGER\_CGAMESTATERUN\_H

#define FORAGER\_CGAMESTATERUN\_H

#include "FMap.h"

#include "FCamera.h"

#include "FHuman.h"

#include "FBuildSystem.h"

#include "FMenu.h"

#include "CGameStdLib.h"

#include "FSaveSystem.h"

namespace game\_framework

{

class CGameStateRun : public CGameState

{

public:

CGameStateRun(CGame\* g);

void OnBeginState();

void OnInit();

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

void GetLastScreen();

protected:

void OnMove();

void OnShow();

private:

void AutoSaveGame();

FCamera camera;

FMap map;

FHuman human;

FBuildSystem buildSystem;

FMenu menu;

FProductBuildSystem pbms;

};

}

#endif // !FORAGER\_CGAMESTATERUN\_H

/\* FileName: CGameStdLib.h \*/

﻿#ifndef FORAGER\_CGAMESTDLIB\_H

#define FORAGER\_CGAMESTDLIB\_H

/\* 請勿添加 \*/

#include "stdafx.h"

#include "Resource.h"

#include <mmsystem.h>

#include "audio.h"

#include "gamelib.h"

#include "D2D1Draw.h"

#endif // !FORAGER\_CGAMESTDLIB\_H

/\* FileName: D2D1Draw.cpp \*/

﻿#include "stdafx.h"

#include "D2D1Draw.h"

#include "gamelib.h"

namespace game\_framework

{

POINT D2D1Draw::screenSize = POINT{ 0, 0 };

bool D2D1Draw::isBeginDraw = false;

bool D2D1Draw::fullscreen = false;

FLOAT D2D1Draw::dpiX = 96.0f, D2D1Draw::dpiY = 96.0f;

ComPtr<ID3D11Device1> D2D1Draw::d3dDevice;

ComPtr<ID3D11DeviceContext1> D2D1Draw::d3dContext;

ComPtr<ID2D1Device> D2D1Draw::d2dDevice;

ComPtr<ID2D1Factory1> D2D1Draw::d2dFactory;

ComPtr<IDXGISwapChain1> D2D1Draw::d2dDXGISwapChain;

ComPtr<ID2D1Bitmap1> D2D1Draw::direct2DBackBuffer;

ComPtr<IDWriteFactory1> D2D1Draw::textFactory;

ComPtr<ID2D1DeviceContext> D2D1Draw::d2dDeviceContext;

ComPtr<IDXGISurface1> D2D1Draw::pDXGISwapChinSurface;

ComPtr<IDXGISurface> D2D1Draw::dxgiBackBuffer;

vector<D2D1BitmapRecord> D2D1Draw::bitmaps;

vector<D2D1\_RESET\_CALL> D2D1Draw::resetResourcesCall;

CDC D2D1Draw::cdc;

D2D1Draw D2D1Draw::d2d1draw;

// Destructor 釋放所有資源, 避免內存洩漏

D2D1Draw::~D2D1Draw()

{

ReleaseDeviceResource();

}

// 讀取 PNG 圖片

ID2D1Bitmap\* D2D1Draw::LoadResourcePNG(int IDB\_BITMAP,

COLORREF color)

{

CPngImage image;

image.Load(IDB\_BITMAP, AfxGetInstanceHandle());

// 透過 GDI 讀取資源圖片

HBITMAP hBmp = image;

HDC hdc = GetDC(NULL);

tagBITMAP bm;

::GetObject(hBmp, sizeof(BITMAP), &bm);

tagBITMAPINFO bi;

memset(&bi, 0, sizeof(BITMAPINFO));

bi.bmiHeader = tagBITMAPINFOHEADER{ sizeof(bi.bmiHeader), bm.bmWidth, -bm.bmHeight, bm.bmPlanes, bm.bmBitsPixel, 0, 0, 0, 0, 0, 0 };

// 將 BGRA 資料讀到陣列, 以便創建 Direct2D 圖片資源

unsigned char\* pBuff = new unsigned char[bm.bmWidth\*

bm.bmHeight \* 4];

UINT pBuffLen = bm.bmWidth \* bm.bmHeight \* 4;

::GetDIBits(hdc, hBmp, 0, bm.bmHeight, (void\*)pBuff, &bi,

DIB\_RGB\_COLORS);

// 去除要去除的顏色

for(UINT i = 0; i < pBuffLen; i += 4)

{

if(color != CLR\_INVALID &&

RGB(pBuff[i + 2], pBuff[i + 1], pBuff[i]) == color)

pBuff[i] = pBuff[i + 1] = pBuff[i + 2] = pBuff[i + 3] = 0;

else

{

// 預乘 alpha

pBuff[i] = int(pBuff[i] \* (pBuff[i + 3] / 255.0));

pBuff[i + 1] = int(pBuff[i + 1] \* (pBuff[i + 3] / 255.0));

pBuff[i + 2] = int(pBuff[i + 2] \* (pBuff[i + 3] / 255.0));

}

}

// 創建 Direct2D 圖片資源

ID2D1Bitmap\* pBitmap = NULL;

pBitmap = D2D1Draw::LoadMemoryBitmap(pBuff, bm.bmWidth,

bm.bmHeight);

// 釋放資源

delete[] pBuff;

DeleteObject(hBmp);

ReleaseDC(nullptr, hdc);

return pBitmap;

}

// 初始化, 設定優先權, 解析度等工作

void D2D1Draw::Init(int sx, int sy)

{

// CPU 優先

SetPriorityClass(GetCurrentProcess(), HIGH\_PRIORITY\_CLASS);

D2D1FPSLimit::fpsLimit.Start();

// 設定解析度

screenSize.x = sx;

screenSize.y = sy;

// 透過 MFC 的 OLE 初始化, 來初始化 COM library

CheckD2D1Fail(AfxOleInit() ? S\_OK : E\_FAIL, "AfxOleInit fail.");

// 重建所有資源

ReCreateAllResources();

// 設定全螢幕模式

SetFullScreen(D2D1\_FULLSCREEN);

}

// 設定全螢幕模式

bool D2D1Draw::SetFullScreen(bool isFullscreen)

{

fullscreen = isFullscreen;

const DXGI\_MODE\_DESC swapChainModeDesc = DXGI\_MODE\_DESC{ (UINT)screenSize.x, (UINT)screenSize.y, DXGI\_RATIONAL{0, 0}, DXGI\_FORMAT\_B8G8R8A8\_UNORM,

DXGI\_MODE\_SCANLINE\_ORDER\_UNSPECIFIED, DXGI\_MODE\_SCALING\_STRETCHED };

// 調整 DXGI 大小

CheckD2D1Fail(d2dDXGISwapChain->ResizeTarget(

&swapChainModeDesc),

"ResizeTarget fail.");

// DXGI 交換鏈, 設定全螢幕模式

HRESULT hr = d2dDXGISwapChain->SetFullscreenState(fullscreen,

NULL);

return hr == S\_OK ? true : false;

}

// 回傳現在是否為全螢幕

bool D2D1Draw::IsFullScreen()

{

return fullscreen;

}

// 取得 DXGI 交換鏈 Buffer 之 Surface 的 DC

CDC\* D2D1Draw::GetBackCDC()

{

// 如果 D2D 已經有繪圖的動作, 應該要先執行 EndDraw, 避免畫面不同步

if(isBeginDraw)

{

d2dDeviceContext->EndDraw();

d2dDeviceContext->BeginDraw();

}

// 取得 DXGI 交換鏈 Buffer 之 Surface

CheckD2D1Fail(d2dDXGISwapChain->GetBuffer(0,

\_\_uuidof(IDXGISurface1),

(void\*\*)pDXGISwapChinSurface.ReleaseAndGetAddressOf()),

"GetBuffer fail");

HDC hdc = NULL;

// 取得 DXGI 交換鏈 Buffer 之 Surface 的 HDC

CheckD2D1Fail(pDXGISwapChinSurface->GetDC(FALSE, &hdc),

"Get DC fail");

// 綁訂到 CDC

cdc.Attach(hdc);

return &cdc;

}

// 釋放 DXGI 交換鏈 Surface 之 DC

void D2D1Draw::ReleaseBackCDC()

{

// 解除 HDC 綁定

cdc.Detach();

// 釋放 DXGI 交換鏈 Buffer 之 HDC

CheckD2D1Fail(pDXGISwapChinSurface->ReleaseDC(NULL),

"Release DC fail.");

// 釋放交換鏈 Surface 暫存

pDXGISwapChinSurface.ReleaseAndGetAddressOf();

}

// 塗背景顏色

void D2D1Draw::BltBackColor(COLORREF color)

{

// Direct2D 繪圖起手式 BeginDraw

BeginDraw();

// 塗顏色到背景

d2dDeviceContext->Clear(ConvertRGBtoColorF(color));

}

// 渲染緩衝到主螢幕

void D2D1Draw::BltBackToPrimary()

{

// 渲染緩衝到主螢幕 (這個部分放在 EndDraw 一起做)

EndDraw();

}

// 渲染圖片到緩衝 (CMovingBitmap 相容)

void D2D1Draw::BltBitmapToBack(unsigned bmpID, int x, int y)

{

// 檢查圖片是否存在

CheckD2D1Fail((UINT)bmpID < bitmaps.size() ? S\_OK : E\_FAIL,

"Draw null bitmap.");

// 進行畫圖的動作

DrawBitmap(bitmaps[bmpID].pBitmap, x, y);

}

// 渲染縮放後圖片到緩衝 (CMovingBitmap 相容)

void D2D1Draw::BltBitmapToBack(unsigned bmpID, int x, int y,

double factor)

{

// 檢查圖片是否存在

CheckD2D1Fail((UINT)bmpID < bitmaps.size() ? S\_OK : E\_FAIL,

"Draw null bitmap.");

// 進行畫圖的動作

DrawBitmap(bitmaps[bmpID].pBitmap, x, y, 1.0f,

(int)(bitmaps[bmpID].pBitmap->GetSize().width \* factor),

(int)(bitmaps[bmpID].pBitmap->GetSize().height \* factor));

}

// 渲染圖片到圖片 (CMovingBitmap 相容) 警告:此功能尚未測試

void D2D1Draw::BltBitmapToBitmap(unsigned bmpID,

unsigned bmpTargetID, int x,

int y)

{

// 檢查圖片是否存在

CheckD2D1Fail((UINT)bmpID < bitmaps.size() ? S\_OK : E\_FAIL,

"Draw null bitmap.");

CheckD2D1Fail((UINT)bmpTargetID < bitmaps.size() ? S\_OK :

E\_FAIL,

"Draw null bitmap.");

// 取出圖片指標

ID2D1Bitmap\* bmp = bitmaps[bmpID].pBitmap;

ID2D1Bitmap\* bmpTarget = bitmaps[bmpTargetID].pBitmap;

// 創建 Direct2D Device Context 相容圖片渲染目標

ID2D1BitmapRenderTarget\* pBRT = NULL;

CheckD2D1Fail(d2dDeviceContext->CreateCompatibleRenderTarget(

bmpTarget->GetSize(), &pBRT), "Create BitmapRenderTarget fail");

pBRT->BeginDraw();

// 取得圖片矩形

D2D1\_RECT\_F dest = D2D1::RectF((FLOAT)x, (FLOAT)y,

bmp->GetSize().width,

bmp->GetSize().height);

// 把圖片畫到另一張圖

pBRT->DrawBitmap(bmp, dest);

// EndDraw 並檢查錯誤

CheckD2D1Fail(pBRT->EndDraw(), "BltBitmapToBitmap fail.");

// 釋放資源

D2D1\_SAFERELEASE(bitmaps[bmpTargetID].pBitmap);

pBRT->GetBitmap(&bitmaps[bmpTargetID].pBitmap);

pBRT->Release();

}

// 取得螢幕大小矩形

void D2D1Draw::GetClientRect(CRect& r)

{

r = CRect(0, 0, screenSize.x, screenSize.y);

}

// 檢查錯誤

void D2D1Draw::CheckD2D1Fail(HRESULT hr,

const char\* errorMessage)

{

if(hr == D2DERR\_RECREATE\_TARGET ||

hr == DXGI\_ERROR\_DEVICE\_REMOVED ||

hr == DXGI\_STATUS\_OCCLUDED)

{

// 如果是這些 error code 應該要重新創建資源

ReCreateAllResources();

}

else if(hr != S\_OK)

{

// 輸出是什麼錯誤訊息, 並終止程式

stringstream hex;

hex << std::hex << hr;

string message = "Direct2D error:\n\nHRESULT: 0x" + hex.str() +

"\n\nFile: " +

string(\_\_FILE\_\_)

+ "\nLine: " + to\_string(\_\_LINE\_\_) + "\n\nErrorMessage: " +

string(

errorMessage) + "\n\n";

LPTSTR errorText = NULL;

// 取得 Windows 錯誤訊息

FormatMessage(FORMAT\_MESSAGE\_FROM\_SYSTEM |

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER |

FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, hr, MAKELANGID(LANG\_ENGLISH, SUBLANG\_DEFAULT),

(LPTSTR)&errorText, 0,

NULL);

if(errorText != NULL)

{

message += "ErrorDescription: " + string(errorText);

LocalFree(errorText);

errorText = NULL;

}

AfxMessageBox(message.c\_str(), MB\_OK);

AfxDebugBreak();

exit(1);

}

}

// 註冊 Callback Function 回傳識別代號 (因為有些 Direct2D 資源, 在別的類別管理. 故重建資源時, 應該要呼叫這些 Callback 讓它們釋放並再創建資源)

int D2D1Draw::RegisterCallBack(D2D1\_RESET\_CALL callBackPtr)

{

// 放到 vector

for(unsigned int i = 0; i < resetResourcesCall.size(); i++)

if(resetResourcesCall[i] == nullptr)

{

resetResourcesCall[i] = callBackPtr;

return i;

}

resetResourcesCall.push\_back(callBackPtr);

return (int)resetResourcesCall.size() - 1;

}

// 取消註冊 Callback Function, 避免呼叫到已經被刪除之物件的 Member Function.

void D2D1Draw::UnRegisterCallBack(UINT callBackID)

{

// 檢查是否有這個 Callback Pointer.

CheckD2D1Fail(callBackID >= 0 &&

callBackID < resetResourcesCall.size() ? S\_OK : E\_FAIL,

"UnRegister CallBack out of range.");

resetResourcesCall[callBackID] = nullptr;

}

// 註冊 Bitmap (資源版本)(這些圖片資源會再需要的時候, 重新被創建, 並在離開時釋放)

int D2D1Draw::RegisterBitmap(int IDB\_BITMAP, COLORREF color)

{

for(unsigned int i = 0; i < bitmaps.size(); i++)

if(bitmaps[i].IDB\_BITMAP == IDB\_BITMAP)

return i;

bitmaps.push\_back(D2D1BitmapRecord{ IDB\_BITMAP, "", color, D2D1Draw::LoadResourceBitmap(IDB\_BITMAP, color) });

return (int)bitmaps.size() - 1;

}

// 註冊 Bitmap (路徑版本)(這些圖片資源會再需要的時候, 重新被創建, 並在離開時釋放)

int D2D1Draw::RegisterBitmap(const string& path, COLORREF color)

{

for(unsigned int i = 0; i < bitmaps.size(); i++)

if(bitmaps[i].path == path)

return i;

bitmaps.push\_back(D2D1BitmapRecord{ -1, path, color, LoadPathBitmap(path.c\_str(), color) });

return (int)bitmaps.size() - 1;

}

// 透過記憶體中圖片的 RGB 資料來讀取圖片

ID2D1Bitmap\* D2D1Draw::LoadMemoryBitmap(unsigned char\* pBuff,

int w, int h)

{

ID2D1Bitmap\* pBitmap = NULL;

D2D1\_BITMAP\_PROPERTIES bmpProp = D2D1::BitmapProperties(

D2D1::PixelFormat(

DXGI\_FORMAT\_B8G8R8A8\_UNORM,

D2D1\_ALPHA\_MODE\_PREMULTIPLIED), dpiX, dpiY);

CheckD2D1Fail(d2dDeviceContext->CreateBitmap(D2D1::SizeU(w, h),

pBuff, 4 \* w, bmpProp, &pBitmap), "Create D2D1 bitmap fail.");

return pBitmap;

}

// 創建 DirectWrite 的文字格式

void D2D1Draw::CreateTextFormat(WCHAR const\* fontFamilyName,

IDWriteFontCollection\* fontCollection,

DWRITE\_FONT\_WEIGHT fontWeight,

DWRITE\_FONT\_STYLE fontStyle, DWRITE\_FONT\_STRETCH fontStretch,

FLOAT fontSize,

WCHAR const\* localeName,

IDWriteTextFormat\*\* textFormat)

{

CheckD2D1Fail(textFactory->CreateTextFormat(fontFamilyName,

fontCollection,

fontWeight, fontStyle,

fontStretch, fontSize, localeName, textFormat),

"Create TextFormat fail.");

}

// 創建著色用筆刷

void D2D1Draw::CreateSolidColorBrush(D2D1\_COLOR\_F& color,

ID2D1SolidColorBrush\*\* solidColorBrush)

{

CheckD2D1Fail(d2dDeviceContext->CreateSolidColorBrush(color,

solidColorBrush),

"Create Brush fail.");;

}

// 創建內建特效

void D2D1Draw::CreateEffect(REFCLSID effectId,

ID2D1Effect\*\* effect)

{

CheckD2D1Fail(d2dDeviceContext->CreateEffect(effectId, effect),

"Create Effect fail.");

}

// 取得圖片資源紀錄

D2D1BitmapRecord D2D1Draw::GetBitmapRecord(int bmpID)

{

CheckD2D1Fail((UINT)bmpID < bitmaps.size() ? S\_OK : E\_FAIL,

"Find BitmapRecord fail.");

return bitmaps[bmpID];

}

// 透過 HDC 綁定的字體, 創建 DirectWrite 之 FontFace 字形物件

void D2D1Draw::CreateFontFaceFromHdc(HDC hdc,

IDWriteFontFace\*\* fontFace)

{

ComPtr<IDWriteGdiInterop> gdiInterop;

CheckD2D1Fail(textFactory->GetGdiInterop(

gdiInterop.ReleaseAndGetAddressOf()),

"Get GdiInterop fail.");

CheckD2D1Fail(gdiInterop->CreateFontFaceFromHdc(hdc, fontFace),

"Create Font Face fail.");

}

// 創建路徑幾何容器

void D2D1Draw::CreatePathGeometry(ID2D1PathGeometry\*\*

pathGeometry)

{

CheckD2D1Fail(d2dFactory->CreatePathGeometry(pathGeometry),

"Create Path Geometry fail.");

}

// 創建圓形幾何

void D2D1Draw::CreateEllipseGeometry(const D2D1\_ELLIPSE\*

ellipse,

ID2D1EllipseGeometry\*\* ellipseGeometry)

{

CheckD2D1Fail(d2dFactory->CreateEllipseGeometry(ellipse,

ellipseGeometry),

"Create Ellipse Geometry fail.");

}

// 創建矩形幾何

void D2D1Draw::CreateRectangleGeometry(const D2D1\_RECT\_F\* rect,

ID2D1RectangleGeometry\*\* rectangleGeometry)

{

CheckD2D1Fail(d2dFactory->CreateRectangleGeometry(rect,

rectangleGeometry),

"Create Rectangle Geometry fail.");

}

// 填滿幾何形狀

void D2D1Draw::FillGeometry(ID2D1Geometry\* geometry,

ID2D1Brush\* brush)

{

BeginDraw();

d2dDeviceContext->FillGeometry(geometry, brush);

}

// 創建筆刷用的梯度顏色

void D2D1Draw::CreateGradientStopCollection(const

D2D1\_GRADIENT\_STOP\*

gradientStops, UINT gradientStopsCount,

D2D1\_GAMMA colorInterpolationGamma, D2D1\_EXTEND\_MODE extendMode,

ID2D1GradientStopCollection\*\* gradientStopCollection)

{

CheckD2D1Fail(d2dDeviceContext->CreateGradientStopCollection(

gradientStops,

gradientStopsCount, colorInterpolationGamma,

extendMode, gradientStopCollection),

"Create Gradient Stop Collection fail");

}

// 創建放射狀梯度顏色筆刷

void D2D1Draw::CreateRadialGradientBrush(const

D2D1\_RADIAL\_GRADIENT\_BRUSH\_PROPERTIES&

radialGradientBrushProperties,

ID2D1GradientStopCollection\* gradientStopCollection,

ID2D1RadialGradientBrush\*\* radialGradientBrush)

{

CheckD2D1Fail(d2dDeviceContext->CreateRadialGradientBrush(

radialGradientBrushProperties, gradientStopCollection,

radialGradientBrush),

"Create Radial Gradient Brush fail");

}

// DiretWrite 寫字

void D2D1Draw::DrawDiretWriteText(const char\* text, int textLen,

IDWriteTextFormat\* format, int x, int y, ID2D1Brush\* brush)

{

// 轉成 wchar\_t

static wchar\_t textBuffer[2048] = { 0 };

::ZeroMemory(textBuffer, 2048);

swprintf(textBuffer, 2048, L"%hs", text);

// 取得 TextLayout 判讀文字佔據螢幕的大小

ComPtr<IDWriteTextLayout> pTextLayout;

CheckD2D1Fail(textFactory->CreateTextLayout(textBuffer, textLen,

format, 0.0f,

0.0f, &pTextLayout), "Create Textlayout fail.");

DWRITE\_TEXT\_METRICS textMetrics;

pTextLayout->GetMetrics(&textMetrics);

pTextLayout->SetMaxHeight(textMetrics.width);

pTextLayout->SetMaxWidth(textMetrics.height);

// 畫字

BeginDraw();

d2dDeviceContext->DrawTextLayout(D2D1\_POINT\_2F{ (FLOAT)x, (FLOAT)y },

pTextLayout.Get(), brush);

}

// DiretWrite 透過文字幾何繪製有邊框的寫字

void D2D1Draw::DrawDiretWriteOutlineText(const char\* text,

int textLen,

IDWriteTextFormat\* format, int x, int y,

ID2D1Brush\* brush

, ID2D1Brush\* outlineBrush, ID2D1PathGeometry\* pathGeometry,

FLOAT outlineWeight)

{

// 轉成 wchar\_t

static wchar\_t textBuffer[2048] = { 0 };

::ZeroMemory(textBuffer, 2048);

swprintf(textBuffer, 2048, L"%hs", text);

D2D1\_RECT\_F bounds;

BeginDraw();

// 取得邊界

pathGeometry->GetBounds(D2D1::Matrix3x2F::Identity(), &bounds);

d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Translation((

FLOAT)x - bounds.left, (FLOAT)y - bounds.top));

// 畫文字幾何

d2dDeviceContext->DrawGeometry(pathGeometry, outlineBrush,

outlineWeight);

d2dDeviceContext->FillGeometry(pathGeometry, brush);

d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Identity());

}

// 畫特效

void D2D1Draw::DrawEffect(ID2D1Effect\* effect, int x, int y,

int w, int h,

FLOAT scaleX, FLOAT scaleY, FLOAT degree)

{

// Direct2D 繪圖起手式

BeginDraw();

// 檢查有沒有縮放或是選轉, 如果有要設定轉換矩陣

D2D1\_MATRIX\_3X2\_F transform = D2D1::Matrix3x2F::Translation((

FLOAT)x, (FLOAT)y);

if(scaleX != 1.0f && scaleY != 1.0f && w != -1 && h != -1)

transform = transform \* D2D1::Matrix3x2F::Scale(D2D1::Size(

scaleX, scaleY),

D2D1::Point2F((FLOAT)(x + w / 2),

(FLOAT)(y + h / 2)));

if(degree != 0 && w != -1 && h != -1)

transform = transform \* D2D1::Matrix3x2F::Rotation(degree,

D2D1::Point2F((FLOAT)(x + w / 2), (FLOAT)(y + h / 2)));

d2dDeviceContext->SetTransform(transform);

d2dDeviceContext->DrawImage(effect);

d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Identity());

}

// 畫圖片

void D2D1Draw::DrawBitmap(ID2D1Bitmap\* bmpPtr, int dstX,

int dstY, FLOAT alpha,

int dstW, int dstH, D2D1\_RECT\_F\* srcR

, FLOAT scaleX, FLOAT scaleY, FLOAT degree)

{

// Direct2D 繪圖起手式

BeginDraw();

// 檢查有沒有縮放或是選轉, 如果有要設定轉換矩陣

D2D1\_MATRIX\_3X2\_F transform = D2D1::Matrix3x2F::Identity();

if(scaleX != 1.0f && scaleY != 1.0f && dstW != -1 && dstW != -1)

transform = transform \* D2D1::Matrix3x2F::Scale(D2D1::Size(

scaleX, scaleY),

D2D1::Point2F((FLOAT)(dstX + dstW / 2),

(FLOAT)(dstY + dstH / 2)));

if(degree != 0 && dstW != -1 && dstH != -1)

transform = transform \* D2D1::Matrix3x2F::Rotation(degree,

D2D1::Point2F((FLOAT)(dstX + dstW / 2),

(FLOAT)(dstY + dstH / 2)));

if((scaleX != 1.0f && scaleY != 1.0f && dstW != -1 &&

dstW != -1) ||

(degree != 0 && dstW != -1 && dstH != -1))

d2dDeviceContext->SetTransform(transform);

// 檢查是不是輸出指定的圖片範圍

if(srcR != NULL)

{

D2D1\_RECT\_F dest = D2D1::RectF((FLOAT)dstX, (FLOAT)dstY,

(FLOAT)(dstX + (dstW == dstH &&

dstW == -1 ? srcR->right - srcR->left : dstW)),

(FLOAT)(dstY + (dstW == dstH &&

dstH == -1 ? srcR->bottom - srcR->top : dstH)));

d2dDeviceContext->DrawBitmap(bmpPtr, dest, alpha,

D2D1\_BITMAP\_INTERPOLATION\_MODE\_LINEAR, srcR);

}

else

{

D2D1\_RECT\_F dest = D2D1::RectF((FLOAT)dstX, (FLOAT)dstY,

(FLOAT)(dstX + (dstW == dstH &&

dstW == -1 ? (int)bmpPtr->GetSize().width : dstW)),

(FLOAT)(dstY + (dstW == dstH &&

dstW == -1 ? (int)bmpPtr->GetSize().height : dstH)));

d2dDeviceContext->DrawBitmap(bmpPtr, dest, alpha);

}

// 如果設定轉換矩陣, 要把它變成預設的

if((scaleX != 1.0f && scaleY != 1.0f && dstW != -1 &&

dstW != -1) ||

(degree != 0 && dstW != -1 && dstH != -1))

d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Identity());

}

// 畫圖片

void D2D1Draw::DrawBitmap(int bmpID, int dstX, int dstY,

FLOAT alpha, int dstW,

int dstH, D2D1\_RECT\_F\* srcR

, FLOAT scaleX, FLOAT scaleY, FLOAT degree)

{

// 檢查圖片是否存在

CheckD2D1Fail((UINT)bmpID < bitmaps.size() ? S\_OK : E\_FAIL,

"Draw null bitmap.");

DrawBitmap(bitmaps[bmpID].pBitmap, dstX, dstY, alpha, dstW,

dstH, srcR, scaleX,

scaleY, degree);

}

// 轉換 COLORREF(BGR) 到 D2D1::ColorF(RGB) 的格式

D2D1::ColorF D2D1Draw::ConvertRGBtoColorF(COLORREF color)

{

return D2D1::ColorF(RGB(GetBValue(color), GetGValue(color),

GetRValue(color)));

}

// 讀取資源圖片並回傳指標

ID2D1Bitmap\* D2D1Draw::LoadResourceBitmap(int IDB\_BITMAP,

COLORREF color)

{

// 透過 GDI 讀取資源圖片

HBITMAP hBmp = ::LoadBitmap(AfxGetInstanceHandle(),

MAKEINTRESOURCE(IDB\_BITMAP));

HDC hdc = GetDC(NULL);

tagBITMAP bm;

::GetObject(hBmp, sizeof(BITMAP), &bm);

tagBITMAPINFO bi;

memset(&bi, 0, sizeof(BITMAPINFO));

bi.bmiHeader = tagBITMAPINFOHEADER{ sizeof(bi.bmiHeader), bm.bmWidth, -bm.bmHeight, bm.bmPlanes, bm.bmBitsPixel, 0, 0, 0, 0, 0, 0 };

// 將 BGRA 資料讀到陣列, 以便創建 Direct2D 圖片資源

unsigned char\* pBuff = new unsigned char[bm.bmWidth\*

bm.bmHeight \* 4];

UINT pBuffLen = bm.bmWidth \* bm.bmHeight \* 4;

::GetDIBits(hdc, hBmp, 0, bm.bmHeight, (void\*)pBuff, &bi,

DIB\_RGB\_COLORS);

// 去除要去除的顏色

for(UINT i = 0; i < pBuffLen; i += 4)

{

if(color != CLR\_INVALID &&

RGB(pBuff[i + 2], pBuff[i + 1], pBuff[i]) == color)

pBuff[i] = pBuff[i + 1] = pBuff[i + 2] = pBuff[i + 3] = 0;

else

pBuff[i + 3] = 255;

}

// 創建 Direct2D 圖片資源

ID2D1Bitmap\* pBitmap = NULL;

pBitmap = D2D1Draw::LoadMemoryBitmap(pBuff, bm.bmWidth,

bm.bmHeight);

// 釋放資源

delete[] pBuff;

DeleteObject(hBmp);

ReleaseDC(nullptr, hdc);

return pBitmap;

}

// 讀取路徑圖片並回傳指標

ID2D1Bitmap\* D2D1Draw::LoadPathBitmap(const char\* path,

COLORREF color)

{

// 透過 GDI 讀取圖片

HBITMAP hBmp = (HBITMAP)LoadImage(NULL, path, IMAGE\_BITMAP, 0,

0,

LR\_LOADFROMFILE);

HDC hdc = GetDC(NULL);

tagBITMAP bm;

::GetObject(hBmp, sizeof(BITMAP), &bm);

tagBITMAPINFO bi;

memset(&bi, 0, sizeof(BITMAPINFO));

bi.bmiHeader = tagBITMAPINFOHEADER{ sizeof(bi.bmiHeader), bm.bmWidth, -bm.bmHeight, bm.bmPlanes, bm.bmBitsPixel, 0, 0, 0, 0, 0, 0 };

// 將 BGRA 資料讀到陣列, 以便創建 Direct2D 圖片資源

unsigned char\* pBuff = new unsigned char[bm.bmWidth\*

bm.bmHeight \* 4];

UINT pBuffLen = bm.bmWidth \* bm.bmHeight \* 4;

::GetDIBits(hdc, hBmp, 0, bm.bmHeight, (void\*)pBuff, &bi,

DIB\_RGB\_COLORS);

// 去除要去除的顏色

for(UINT i = 0; i < pBuffLen; i += 4)

{

if(color != CLR\_INVALID &&

RGB(pBuff[i + 2], pBuff[i + 1], pBuff[i]) == color)

pBuff[i] = pBuff[i + 1] = pBuff[i + 2] = pBuff[i + 3] = 0;

else

pBuff[i + 3] = 255;

}

// 創建 Direct2D 圖片資源

ID2D1Bitmap\* pBitmap = NULL;

pBitmap = D2D1Draw::LoadMemoryBitmap(pBuff, bm.bmWidth,

bm.bmHeight);

// 釋放資源

delete[] pBuff;

DeleteObject(hBmp);

ReleaseDC(nullptr, hdc);

return pBitmap;

}

// 釋放設備資源與一些設備無關資源

void D2D1Draw::ReleaseDeviceResource()

{

isBeginDraw = false;

d3dDevice.ReleaseAndGetAddressOf();

d3dContext.ReleaseAndGetAddressOf();

d2dDevice.ReleaseAndGetAddressOf();

d2dFactory.ReleaseAndGetAddressOf();

d2dDXGISwapChain.ReleaseAndGetAddressOf();

direct2DBackBuffer.ReleaseAndGetAddressOf();

d2dDeviceContext.ReleaseAndGetAddressOf();

textFactory.ReleaseAndGetAddressOf();

dxgiBackBuffer.ReleaseAndGetAddressOf();

// 如果有 GetDC 沒有 ReleaseDC 會造成記憶體洩漏

if(pDXGISwapChinSurface.Get()) pDXGISwapChinSurface->ReleaseDC(

NULL);

pDXGISwapChinSurface.ReleaseAndGetAddressOf();

// 釋放圖片

for(unsigned int i = 0; i < bitmaps.size(); i++)

D2D1\_SAFERELEASE(bitmaps[i].pBitmap);

}

// 創建設備資源

void D2D1Draw::CreateDeviceResources()

{

// 設定 DPI

dpiX = dpiY = 96.0;

// 看是不是 DEBUG 模式要輸出 Direct2D DEBUG 資訊

D2D1\_FACTORY\_OPTIONS options = D2D1\_FACTORY\_OPTIONS{ D2D1\_DEBUG\_LEVEL\_NONE };

#if defined(DEBUG) || defined(\_DEBUG)

options.debugLevel = D2D1\_DEBUG\_LEVEL\_INFORMATION;

#endif

// 創建工廠

CheckD2D1Fail(D2D1CreateFactory(

D2D1\_FACTORY\_TYPE\_SINGLE\_THREADED,

\_\_uuidof(ID2D1Factory1), &options,

reinterpret\_cast<void\*\*>(d2dFactory.GetAddressOf())),

"Create D2D1 factory fail.");

CheckD2D1Fail(DWriteCreateFactory(DWRITE\_FACTORY\_TYPE\_SHARED,

\_\_uuidof(textFactory.Get()),

reinterpret\_cast<IUnknown\*\*>(textFactory.GetAddressOf())),

"Create DirectWritw factory fail.");

// 創建 D3D 資源以便創建 DXGI

D3D\_FEATURE\_LEVEL featureLevels[] = { D3D\_FEATURE\_LEVEL\_11\_1, D3D\_FEATURE\_LEVEL\_11\_0, D3D\_FEATURE\_LEVEL\_10\_1,

D3D\_FEATURE\_LEVEL\_10\_0, D3D\_FEATURE\_LEVEL\_9\_3, D3D\_FEATURE\_LEVEL\_9\_2, D3D\_FEATURE\_LEVEL\_9\_1

};

ComPtr<ID3D11Device> device;

ComPtr<ID3D11DeviceContext> context;

D3D\_FEATURE\_LEVEL returnedFeatureLevel;

CheckD2D1Fail(D3D11CreateDevice(nullptr,

D3D\_DRIVER\_TYPE\_HARDWARE, 0,

D3D11\_CREATE\_DEVICE\_BGRA\_SUPPORT, featureLevels,

ARRAYSIZE(featureLevels), D3D11\_SDK\_VERSION,

device.GetAddressOf(),

&returnedFeatureLevel,

context.GetAddressOf()), "Create D3D11 device fail.");

device->QueryInterface(\_\_uuidof(ID3D11Device1),

(void\*\*)d3dDevice.GetAddressOf());

context->QueryInterface(\_\_uuidof(ID3D11DeviceContext1),

(void\*\*)d3dContext.GetAddressOf());

ComPtr<IDXGIDevice> dxgiDevice;

d3dDevice->QueryInterface(\_\_uuidof(IDXGIDevice),

(void\*\*)dxgiDevice.GetAddressOf());

// GPU 優先

HRESULT hr = dxgiDevice->SetGPUThreadPriority(5);

// 創建 D2D 資源

CheckD2D1Fail(d2dFactory->CreateDevice(dxgiDevice.Get(),

d2dDevice.GetAddressOf()), "Create D2D1 device fail.");

CheckD2D1Fail(d2dDevice->CreateDeviceContext(

D2D1\_DEVICE\_CONTEXT\_OPTIONS\_NONE,

d2dDeviceContext.GetAddressOf()), "Create D2D1 DC error.");

// 創建 DXGI 交換鏈

ComPtr<IDXGIAdapter> dxgiAdapter;

dxgiDevice->GetAdapter(dxgiAdapter.GetAddressOf());

ComPtr<IDXGIFactory2> dxgiFactory;

dxgiAdapter->GetParent(IID\_PPV\_ARGS(

dxgiFactory.GetAddressOf()));

dxgiFactory->MakeWindowAssociation(NULL,

DXGI\_MWA\_NO\_WINDOW\_CHANGES | DXGI\_MWA\_NO\_ALT\_ENTER |

DXGI\_MWA\_NO\_PRINT\_SCREEN);

DXGI\_SWAP\_CHAIN\_DESC1 swapChainDesc = DXGI\_SWAP\_CHAIN\_DESC1

{

(UINT)screenSize.x, //UINT Width

(UINT)screenSize.y, //UINT Height

DXGI\_FORMAT\_B8G8R8A8\_UNORM, //DXGI\_FORMAT Format

false, //BOOL Stereo

DXGI\_SAMPLE\_DESC{ (UINT)1, (UINT)0 }, //DXGI\_SAMPLE\_DESC SampleDesc

DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT, //DXGI\_USAGE BufferUsage

2, //UINT BufferCount

DXGI\_SCALING\_STRETCH, //DXGI\_SCALING Scaling

DXGI\_SWAP\_EFFECT\_DISCARD, //DXGI\_SWAP\_EFFECT SwapEffect

DXGI\_ALPHA\_MODE\_UNSPECIFIED, //DXGI\_ALPHA\_MODE AlphaMode

DXGI\_SWAP\_CHAIN\_FLAG\_ALLOW\_MODE\_SWITCH | DXGI\_SWAP\_CHAIN\_FLAG\_GDI\_COMPATIBLE //UINT Flags

};

CheckD2D1Fail(dxgiFactory->CreateSwapChainForHwnd(

d3dDevice.Get(),

AfxGetMainWnd()->m\_hWnd, &swapChainDesc, nullptr,

nullptr, d2dDXGISwapChain.GetAddressOf()),

"Create D2D1 swap chain for hWnd fail");

d2dDXGISwapChain->GetBuffer(0,

IID\_PPV\_ARGS(dxgiBackBuffer.ReleaseAndGetAddressOf()));

// 綁定 Direct2D Device Context 到 DXGI 交換鏈

D2D1\_BITMAP\_PROPERTIES1 bitmapProperties =

D2D1::BitmapProperties1(

D2D1\_BITMAP\_OPTIONS\_TARGET | D2D1\_BITMAP\_OPTIONS\_CANNOT\_DRAW,

D2D1::PixelFormat(DXGI\_FORMAT\_B8G8R8A8\_UNORM,

D2D1\_ALPHA\_MODE\_IGNORE), dpiX,

dpiY);

d2dDeviceContext->CreateBitmapFromDxgiSurface(

dxgiBackBuffer.Get(),

&bitmapProperties,

direct2DBackBuffer.ReleaseAndGetAddressOf());

d2dDeviceContext->SetTarget(direct2DBackBuffer.Get());

}

// 重新創建資源

void D2D1Draw::ReCreateAllResources()

{

// 要變成視窗模式才可以釋放 DXGI

if(d2dDXGISwapChain.Get())

d2dDXGISwapChain->SetFullscreenState(false, nullptr);

ReleaseDeviceResource();

CreateDeviceResources();

ReCreateDeviceIndependentResources();

}

// 重新創建圖片資源

void D2D1Draw::ReCreateDeviceIndependentResources()

{

// 重讀圖片

for(unsigned int i = 0; i < bitmaps.size(); i++)

{

D2D1\_SAFERELEASE(bitmaps[i].pBitmap);

if(bitmaps[i].IDB\_BITMAP != -1)

bitmaps[i].pBitmap = LoadResourceBitmap(bitmaps[i].IDB\_BITMAP,

bitmaps[i].colorMask);

else if(bitmaps[i].path != "")

bitmaps[i].pBitmap = LoadPathBitmap(bitmaps[i].path.c\_str(),

bitmaps[i].colorMask);

}

// 呼叫 CALLBACK

for(unsigned int i = 0; i < resetResourcesCall.size(); i++)

if(resetResourcesCall[i]) resetResourcesCall[i]();

}

// Direct2D 繪圖起手式 BeginDraw

void D2D1Draw::BeginDraw()

{

// 一個遊戲迴圈只需 BeginDraw 一次

if(!isBeginDraw)

{

d2dDeviceContext->BeginDraw();

d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Identity());

isBeginDraw = true;

}

}

// Direct2D EndDraw 並將 DXGI 交換鏈緩衝貼到主螢幕

void D2D1Draw::EndDraw()

{

// 防呆

if(isBeginDraw)

{

CheckD2D1Fail(d2dDeviceContext->EndDraw(),

"D2D1 End draw fail.");

DXGI\_PRESENT\_PARAMETERS parameters = { 0 };

// DXGI 交換鏈緩衝貼到主螢幕, 疑似有垂直同步

HRESULT hr = d2dDXGISwapChain->Present1(2, 0, &parameters);

while(hr != S\_OK)

{

// 如果失敗要做錯誤處理

if(hr != DXGI\_STATUS\_OCCLUDED)

CheckD2D1Fail(hr, "D2D1 DXGI swap chain present1 fail.");

hr = d2dDXGISwapChain->Present1(0, DXGI\_PRESENT\_TEST,

&parameters);

if(hr == S\_OK)

SetFullScreen(fullscreen);

}

isBeginDraw = false;

}

}

}

/\* FileName: D2D1Draw.h \*/

﻿#pragma once

#ifndef D2D1DRAW\_H

#define D2D1DRAW\_H

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

// 作者 : WeiChang, Lai

// 目標 : Game framework 4.10 威力加強版

//

// 簡介 : 這個 class 會建立 Direct2D 等繪圖相關物件，以提供其他 class 使用

// 　　 一般的遊戲並不需直接操作這個物件，因此可以不管這個 class 的使用方法.

//

// 備註 : 可能僅適用於 Windows 10 系統.

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

#pragma comment(lib, "d2d1")

#pragma comment(lib, "d3d11")

#pragma comment(lib, "dxguid")

#pragma comment(lib, "dwrite")

#include <wrl.h>

#include <d3d11\_1.h>

#include <d2d1\_1.h>

#include <d2d1\_1helper.h>

#include <d2d1effects\_2.h>

#include <d2d1effecthelpers.h>

#include <dwrite\_1.h>

#include <vector>

#include <string>

#include <functional>

#include <sstream>

#include "D2D1FPSLimit.h"

#include <afxtoolbarimages.h>

using namespace std;

using namespace Microsoft::WRL;

#define D2D1\_RESET\_CALL function<void(void)>

#define D2D1\_SAFERELEASE(ptr) if(ptr) { ptr->Release(); ptr = NULL; }

#define D2D1\_FULLSCREEN false

namespace game\_framework

{

// 定義 bitmap 資源儲存結構

struct D2D1BitmapRecord

{

int IDB\_BITMAP = -1;

string path = "";

COLORREF colorMask = CLR\_INVALID;

ID2D1Bitmap\* pBitmap = NULL;

};

class D2D1Draw

{

public:

~D2D1Draw();

static void Init(int sx, int sy);

static bool SetFullScreen(bool isFullscreen);

static bool IsFullScreen();

static CDC\* GetBackCDC();

static void ReleaseBackCDC();

static void BltBackColor(COLORREF color);

static void BltBackToPrimary();

static void BltBitmapToBack(unsigned bmpID, int x, int y);

static void BltBitmapToBack(unsigned bmpID, int x, int y,

double factor);

static void BltBitmapToBitmap(unsigned bmpID,

unsigned bmpTargetID, int x,

int y);

static void GetClientRect(CRect& r);

static void CheckD2D1Fail(HRESULT hr, const char\* errorMessage);

static int RegisterCallBack(D2D1\_RESET\_CALL callBackPtr);

static void UnRegisterCallBack(UINT callBackID);

static int RegisterBitmap(int IDB\_BITMAP,

COLORREF color = CLR\_INVALID);

static int RegisterBitmap(const string& path,

COLORREF color = CLR\_INVALID);

static ID2D1Bitmap\* LoadMemoryBitmap(unsigned char\* pBuff,

int w, int h);

static ID2D1Bitmap\* LoadResourcePNG(int IDB\_BITMAP,

COLORREF color = CLR\_INVALID);

static void CreateTextFormat(WCHAR const\* fontFamilyName,

IDWriteFontCollection\* fontCollection,

DWRITE\_FONT\_WEIGHT fontWeight,

DWRITE\_FONT\_STYLE fontStyle, DWRITE\_FONT\_STRETCH fontStretch,

FLOAT fontSize,

WCHAR const\* localeName,

IDWriteTextFormat\*\* textFormat);

static void CreateSolidColorBrush(D2D1\_COLOR\_F& color,

ID2D1SolidColorBrush\*\* solidColorBrush);

static void CreateEffect(REFCLSID effectId,

ID2D1Effect\*\* effect);

static D2D1BitmapRecord GetBitmapRecord(int bmpID);

static void DrawDiretWriteText(const char\* text, int textLen,

IDWriteTextFormat\* format, int x, int y, ID2D1Brush\* brush);

static void DrawDiretWriteOutlineText(const char\* text,

int textLen,

IDWriteTextFormat\* format, int x, int y, ID2D1Brush\* brush

, ID2D1Brush\* outlineBrush, ID2D1PathGeometry\* pathGeometry,

FLOAT outlineWeight);

static void DrawEffect(ID2D1Effect\* effect, int x, int y,

int w =-1, int h =-1,

FLOAT scaleX =1, FLOAT scaleY =1,

FLOAT degree =0);

static void DrawBitmap(ID2D1Bitmap\* bmpPtr, int dstX, int dstY,

FLOAT alpha = 1,

int dstW = -1, int dstH = -1,

D2D1\_RECT\_F\* srcR = NULL

, FLOAT scaleX = 1, FLOAT scaleY = 1, FLOAT degree = 0);

static void DrawBitmap(int bmpID, int dstX, int dstY,

FLOAT alpha = 1,

int dstW = -1, int dstH = -1, D2D1\_RECT\_F\* srcR = NULL

, FLOAT scaleX = 1, FLOAT scaleY = 1, FLOAT degree = 0);

static void ReCreateAllResources();

static void CreateFontFaceFromHdc(HDC hdc,

IDWriteFontFace\*\* fontFace);

static void CreatePathGeometry(ID2D1PathGeometry\*\*

pathGeometry);

static void CreateEllipseGeometry(const D2D1\_ELLIPSE\* ellipse,

ID2D1EllipseGeometry\*\* ellipseGeometry);

static void CreateRectangleGeometry(const D2D1\_RECT\_F\* rect,

ID2D1RectangleGeometry\*\* rectangleGeometry);

static void FillGeometry(ID2D1Geometry\* pathGeometry,

ID2D1Brush\* brush);

static void CreateGradientStopCollection(const

D2D1\_GRADIENT\_STOP\*

gradientStops, UINT gradientStopsCount,

D2D1\_GAMMA colorInterpolationGamma, D2D1\_EXTEND\_MODE extendMode,

ID2D1GradientStopCollection\*\* gradientStopCollection);

static void CreateRadialGradientBrush(const

D2D1\_RADIAL\_GRADIENT\_BRUSH\_PROPERTIES&

radialGradientBrushProperties,

ID2D1GradientStopCollection\* gradientStopCollection,

ID2D1RadialGradientBrush\*\* radialGradientBrush);

private:

D2D1Draw()=default;

static D2D1::ColorF ConvertRGBtoColorF(COLORREF color);

static ID2D1Bitmap\* LoadResourceBitmap(int IDB\_BITMAP,

COLORREF color);

static ID2D1Bitmap\* LoadPathBitmap(const char\* path,

COLORREF color);

static void ReleaseDeviceResource();

static void CreateDeviceResources();

static void ReCreateDeviceIndependentResources();

\_\_forceinline static void BeginDraw();

\_\_forceinline static void EndDraw();

static bool isBeginDraw;

static bool fullscreen;

static POINT screenSize;

static FLOAT dpiX, dpiY;

static ComPtr<ID3D11Device1> d3dDevice;

static ComPtr<ID3D11DeviceContext1> d3dContext;

static ComPtr<ID2D1Device> d2dDevice;

static ComPtr<ID2D1Factory1> d2dFactory;

static ComPtr<IDXGISwapChain1> d2dDXGISwapChain;

static ComPtr<ID2D1Bitmap1> direct2DBackBuffer;

static ComPtr<ID2D1DeviceContext> d2dDeviceContext;

static ComPtr<IDWriteFactory1> textFactory;

static ComPtr<IDXGISurface1> pDXGISwapChinSurface;

static ComPtr<IDXGISurface> dxgiBackBuffer;

static vector<D2D1BitmapRecord> bitmaps;

static vector<D2D1\_RESET\_CALL> resetResourcesCall;

static CDC cdc;

static D2D1Draw d2d1draw;

};

typedef D2D1Draw CDDraw;

}

#endif // !D2D1DRAW\_H

/\* FileName: D2D1FPSLimit.cpp \*/

﻿#include "stdafx.h"

#include "D2D1FPSLimit.h"

using namespace std;

namespace game\_framework

{

D2D1FPSLimit D2D1FPSLimit::fpsLimit;

D2D1FPSLimit::D2D1FPSLimit()

{

startTime = 0;

lastTime = 0;

}

void D2D1FPSLimit::Start()

{

startTime = timeGetTime();

}

void D2D1FPSLimit::SetGameCycleTime()

{

gameCycleTime = GetCycleTime();

}

void D2D1FPSLimit::SetCurrentTime()

{

lastTime = timeGetTime();

}

DWORD D2D1FPSLimit::GetCycleTime()

{

DWORD currentTime = timeGetTime();

return currentTime - lastTime;

}

DWORD D2D1FPSLimit::GetFPS()

{

DWORD time = GetCycleTime();

return 1000 / (time<=0 ? 1000 : time);

}

void D2D1FPSLimit::FPSLimitDelay()

{

DWORD time = GetCycleTime();

if(time >= 0)

{

drawTime = time;

if(FPSDELAY > time && (FPSDELAY - time - 1) > 0)

{

Sleep(FPSDELAY - time - 1);

while(FPSDELAY > time) time = GetCycleTime();

}

frameTime = time;

}

}

}

/\* FileName: D2D1FPSLimit.h \*/

﻿#pragma once

#ifndef D2D1FPSLIMIT\_H

#define D2D1FPSLIMIT\_H

#define FPSDELAY 32

#include <Mmsystem.h>

using namespace std;

namespace game\_framework

{

class D2D1FPSLimit

{

public:

D2D1FPSLimit();

void Start();

void SetCurrentTime();

void SetGameCycleTime();

DWORD GetCycleTime();

DWORD GetFPS();

void FPSLimitDelay();

static D2D1FPSLimit fpsLimit;

DWORD frameTime;

DWORD drawTime;

DWORD gameCycleTime;

private:

DWORD startTime;

DWORD lastTime;

};

}

#endif // !D2D1FPSLIMIT\_H

/\* FileName: FAnimation.cpp \*/

﻿#include "stdafx.h"

#include "FAnimation.h"

namespace game\_framework

{

//讀取圖片及設定動畫初值

void FAnimation::LoadBitmap(int resID, int row, int col, int w,

int h,

int maxCount, COLORREF RGBMask)

{

this->img.LoadBitmap(resID, row, col, w, h, RGBMask);

this->maxCount = maxCount;

this->count = 0;

this->delay\_count = 10;

this->delay\_counter = this->delay\_count;

this->x = this->y = 0;

}

//回傳現在撥放動畫的索引

int FAnimation::GetCurrentBitmapNumber()

{

return this->count;

}

//回傳現在是否為最後一個動畫

bool FAnimation::IsFinalBitmap()

{

return this->count == this->maxCount - 1;

}

//動畫更新

void FAnimation::OnMove()

{

if(--delay\_counter <= 0)

{

delay\_counter = delay\_count;

count++;

if(count == maxCount)

count = 0;

}

}

//播放動畫

void FAnimation::OnShow()

{

this->img.ShowPartBitmap(count);

}

//重置動畫

void FAnimation::Reset()

{

this->count = 0;

this->delay\_counter = this->delay\_count;

}

//設定動畫速度(越大越慢)

void FAnimation::SetDelayCount(int delayCount)

{

this->delay\_count = delayCount;

this->delay\_counter = this->delay\_count;

}

//設定動畫的螢幕位置

void FAnimation::SetTopLeft(int nx, int ny, int wDst, int hDst)

{

this->x = nx;

this->y = ny;

this->img.SetTopLeft(nx, ny, wDst, hDst);

}

//設定動畫透明度

void FAnimation::SetAlpha(int alpha)

{

this->img.SetAlpha(alpha);

}

}

/\* FileName: FAnimation.h \*/

﻿#pragma once

#ifndef FORAGER\_FANIMATION\_H

#define FORAGER\_FANIMATION\_H

#include "CGameStdLib.h"

#include "FAtlas.h"

using namespace std;

namespace game\_framework

{

class FAnimation

{

public:

void LoadBitmap(int resID, int row, int col, int w, int h,

int maxCount,

COLORREF RGBMask = CLR\_INVALID);

int GetCurrentBitmapNumber(); // 取得正在撥放的bitmap是第幾個bitmap

bool IsFinalBitmap(); // 回傳正在撥放的bitmap是否為最後一個bitmap

void OnMove(); // 依頻率更換bitmap

void OnShow(); // 將動畫貼到螢幕

void Reset(); // 重設播放順序回到第一張圖形

void SetDelayCount(

int); // 設定動畫播放速度的常數(越大越慢)

void SetTopLeft(int, int, int wDst = -1,

int hDst = -1); // 將動畫的左上角座標移至 (x,y)

void SetAlpha(int alpha = 255);

private:

FAtlas img;

int count;

int maxCount;

int delay\_counter; // 延緩動畫播放速度的計數器

int delay\_count; // 動畫播放速度的常數

int x, y; // 動畫的座標

};

}

#endif // !FORAGER\_FANIMATION\_H

/\* FileName: FAtlas.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FAtlas.h"

using namespace std;

namespace game\_framework

{

//讀取圖片及設定圖片數值(row:行數; col:列數; w:圖片的單位寬度; h:圖片的單位高度)

void FAtlas::LoadBitmap(int resID, int row, int col, int w,

int h,

COLORREF RGBMask)

{

this->bmpID = D2D1Draw::RegisterBitmap(resID, RGBMask);

this->row = row;

this->col = col;

this->w = w;

this->h = h;

for(int x = 0; x < row; x++)

for(int y = 0; y < col; y++)

srcPosition.push\_back(D2D1::RectF((FLOAT)(y \* w),

(FLOAT)(x \* h),

(FLOAT)(y \* w) + (w), (FLOAT)(x \* h) + (h)));

}

//設定圖片螢幕位置

void FAtlas::SetTopLeft(int xDst, int yDst, int wDst, int hDst)

{

this->xDst = xDst;

this->yDst = yDst;

this->wDst = wDst;

this->hDst = hDst;

useDestSize = !((wDst == -1) && (wDst == hDst));

}

//設定圖片透明度

void FAtlas::SetAlpha(int alpha)

{

if(alpha > 255) alpha = 255;

if(alpha < 0) alpha = 0;

this->alpha = alpha;

}

//印出圖片(參數為第幾個圖片)

void FAtlas::ShowPartBitmap(int partID)

{

if(partID >= 0 && partID < (INT)srcPosition.size())

if(useDestSize)

D2D1Draw::DrawBitmap(bmpID, xDst, yDst, FLOAT(alpha / 255.0),

wDst, hDst,

&srcPosition[partID]);

else

D2D1Draw::DrawBitmap(bmpID, xDst, yDst, FLOAT(alpha / 255.0),

-1, -1,

&srcPosition[partID]);

else

D2D1Draw::CheckD2D1Fail(E\_FAIL, "Part Bitmap out of range.");

}

}

/\* FileName: FAtlas.h \*/

﻿#pragma once

#ifndef FORAGER\_FATLAS\_H

#define FORAGER\_FATLAS\_H

#include "CGameStdLib.h"

#include <vector>

using namespace std;

namespace game\_framework

{

class FAtlas

{

public:

FAtlas() = default;

void LoadBitmap(int resID, int row, int col, int w, int h,

COLORREF RGBMask = CLR\_INVALID);

void ShowPartBitmap(int partID);

void SetTopLeft(int xDst, int yDst, int wDst = -1,

int hDst = -1);

void SetAlpha(int alpha = 255);

private:

int bmpID = 0, alpha = 255, xDst = 0, yDst = 0, wDst = 0,

hDst = 0, row = 0,

col = 0, w = 0, h = 0;

bool useDestSize = false;

vector<D2D1\_RECT\_F> srcPosition;

};

}

#endif // !FORAGER\_FATLAS\_H

/\* FileName: FBag.cpp \*/

﻿#include "stdafx.h"

#include "FBag.h"

#include "FHuman.h"

namespace game\_framework

{

//載入包包圖片;設定包包座標;初始化物件

void FBag::Initialize(FHuman\* human)

{

this->data.reserve(BAG\_MAX);

for(int i = 0; i < BAG\_MAX; i++)

{

this->bagBlocksLoc[i] = FPosition(START\_LOC\_X +

(i%COUNT\_PER\_ROW) \*

DIS\_BETWEEN\_BLOCK,

START\_LOC\_Y + (i / COUNT\_PER\_ROW)

\*DIS\_BETWEEN\_BLOCK);//計算包包格子座標

}

this->bagBlock.LoadBitmap(IDB\_BAGBLOCK, RGB(255, 0, 0));

this->buttomRightNotice.Initialize();

this->mouseTarget.Initialize();

this->human = human;

}

//滑鼠移動事件

void FBag::OnMouseMoveHandle(const CPoint& point)

{

int targetBlock = GetTargetBlock(point);

if(targetBlock != -1)

{

mouseTarget.SetTopLeftWidth(bagBlocksLoc[targetBlock],

BAG\_BLOCK\_WIDTH,

BAG\_BLOCK\_HEIGHT, -10, -10);

mouseTarget.SetPointing(true);

return;

}

mouseTarget.SetPointing(false);

}

//滑鼠左鍵事件

void FBag::OnLButtonDownHandle(const CPoint& point)

{

int targetBlock = GetTargetBlock(point);

if(targetBlock != -1 && int(data.size()) > targetBlock)

{

if(data[targetBlock]->GetIsUsable())

{

data[targetBlock]->UseResource(this->human);

data[targetBlock]->AddCount(-1);

Refresh();

}

}

}

//回傳座標point上的格子索引

int FBag::GetTargetBlock(const CPoint& point)

{

for(int i = 0; i < BAG\_MAX; i++)

{

if(point.x >= bagBlocksLoc[i].\_pX &&

point.x < bagBlocksLoc[i].\_pX + BAG\_BLOCK\_WIDTH &&

point.y >= bagBlocksLoc[i].\_pY &&

point.y < bagBlocksLoc[i].\_pY + BAG\_BLOCK\_WIDTH)

return i;

}

return -1;

}

//將個數為0的物品移除

void FBag::Refresh()

{

for(vector<FResource\*>::iterator it = data.begin();

it != data.end();)

{

if((\*it)->GetCount() <= 0)

{

delete \*it;

it = data.erase(it);

}

else

++it;

}

}

//將物品放入包包

void FBag::GetRes(FResource\* res, int count)

{

if(data.size() <= BAG\_MAX)

{

if(data.size() > 0)

{

for(auto& it : data)

{

if(it->GetId() == res->GetId())

{

it->AddCount(count);

buttomRightNotice.AddNotice(&(\*it));

delete res;

return;

}

}

}

res->AddCount(count);

buttomRightNotice.AddNotice(res);

data.push\_back(res);

}

else

delete res;

}

//顯示包包介面

void FBag::ShowBag()

{

//顯示包包格子

for(int i = 0; i < BAG\_MAX; i++)

{

bagBlock.SetTopLeft(bagBlocksLoc[i].\_pX, bagBlocksLoc[i].\_pY);

bagBlock.ShowBitmap();

}

//顯示包包物品

for(int i = 0; i< int(data.size()); i++)

{

static FText text(L"Consolas", 20, RGB(255, 255, 255));

string temp\_str("x" + to\_string(data[i]->GetCount()));

data[i]->OnShow(bagBlocksLoc[i]);

text.ShowText(temp\_str.c\_str(), bagBlocksLoc[i].\_pX + 55,

bagBlocksLoc[i].\_pY + 55);

}

mouseTarget.OnShow();

}

//包包狀態重置

void FBag::Reset()

{

mouseTarget.SetPointing(false);

}

//移除包包物品

void FBag::ReduceRes(vector<FResReq> req, int productCount)

{

for(int i = 0; i < int(req.size()); i++)

{

for(auto& it : data)

{

if(it->GetId() == req[i].id)

{

it->AddCount(-req[i].count \* productCount);

break;

}

}

}

Refresh();

}

//判斷包包是否滿了

bool FBag::IsFull(int blockID, int count)

{

for(auto& it : data)

{

if(it->GetId() == blockID)

return false;

}

if(data.size() < BAG\_MAX)

return false;

else

return true;

}

//回傳包包內此id的資源個數

int FBag::GetBagResCount(int id)

{

if(data.size() != 0)

{

for(auto& it : data)

{

if(it->GetId() == id)

return it->GetCount();

}

}

return 0;

}

//判斷包包內資源是否足夠去建築或合成

bool FBag::IsEnoughToBuild(vector<FResReq> req,

int productCount)

{

bool isFinded;

if(data.size() != 0)

{

for(int i = 0; i < int(req.size()); i++)

{

isFinded = false;

for(auto& it : data)

{

if(it->GetId() == req[i].id)

{

isFinded = true;

if(it->GetCount() < req[i].count \* productCount)

return false;

break;

}

}

if(!isFinded)

return false;

}

return true;

}

return false;

}

//destructor 釋放資源

FBag::~FBag()

{

for(int i = 0; i < int(data.size()); i++)

delete data[i];

}

//回傳包包的vector(存檔用)

vector<FResource\*>\* FBag::GetBagData()

{

return &this->data;

}

}

/\* FileName: FBag.h \*/

﻿#ifndef FORAGER\_FBAG\_H

#define FORAGER\_FBAG\_H

#include <vector>

#include <string>

#include "FResource.h"

#include "FImage.h"

#include "CGameStdLib.h"

#include "FMouseTarget.h"

#include "FBottomRightNotice.h"

#include "FBuildInfo.h"

using namespace std;

namespace game\_framework

{

class FHuman;

class FBag

{

public:

static const int BAG\_MAX =

20; //包包最大格數

static const int START\_LOC\_X =

100; //第一格包包起始螢幕座標x

static const int START\_LOC\_Y =

200; //第一格包包起始螢幕座標y

static const int COUNT\_PER\_ROW =

10; //每排數量

static const int DIS\_BETWEEN\_BLOCK =

100; //每格距離

static const int BAG\_BLOCK\_WIDTH =

120; //每格寬度(mouseTarget)

static const int BAG\_BLOCK\_HEIGHT =

120; //每格高度(mouseTarget)

void Initialize(FHuman\* human);

void OnMouseMoveHandle(const CPoint& point);

void OnLButtonDownHandle(const CPoint& point);

void GetRes(FResource\* res, int count);

void ShowBag();

void Reset();

void ReduceRes(vector<FResReq> req, int productCount = 1);

bool IsFull(int blockID, int count);

bool IsEnoughToBuild(vector<FResReq> req, int productCount = 1);

int GetBagResCount(int id);

FBottomRightNotice buttomRightNotice;

FImage bagBlock;

~FBag();

/\* 存檔用 \*/

vector<FResource\*>\* GetBagData();

/\* 存檔用 \*/

private:

FPosition bagBlocksLoc[BAG\_MAX];

vector<FResource\*> data;

FMouseTarget mouseTarget;

int GetTargetBlock(const CPoint& point);

FHuman\* human;

void Refresh();

};

}

#endif // !FORAGER\_FBAG\_H

/\* FileName: FBottomRightNotice.cpp \*/

﻿#include "stdafx.h"

#include "FBottomRightNotice.h"

using namespace std;

namespace game\_framework

{

//載入圖片及初始化特效

void FBottomRightNotice::Initialize()

{

maskImage.LoadBitmap(IDB\_BRNTEXTMASK, RGB(255, 255, 255));

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

maskEffect[i].Initialize(&maskImage);

for(UINT i = 0; i < FResource::ResourceID::RESOURCEIDEND; i++)

noticesEffect[i].Initialize(&FResource::resImages[i]);

}

//當獲得物品時, 加入提醒

void FBottomRightNotice::AddNotice(FResource\* res)

{

//提醒最大數為5, 超過則刪除最前面的

if(notices.size() > 5)

DelNotice();

notices.push\_back(res->GetId());

times.push\_back(clock());

name.push\_back(string(res->GetName()));

}

//刪除最前面的提醒

void FBottomRightNotice::DelNotice()

{

if(!notices.empty())

{

maskEffect[0].ResetScale();

noticesEffect[notices[0]].ResetScale();

notices.erase(notices.begin());

times.erase(times.begin());

name.erase(name.begin());

}

}

//顯示

void FBottomRightNotice::OnShow()

{

for(unsigned int i = 0; i < notices.size(); i++)

{

clock\_t temp = clock();

//若時間超過兩秒則刪除提醒

if(temp - times[i] > 2000)

DelNotice();

else

{

//前0.3秒要使用特效

useEffect = temp - times[i] < 300 ? true : false;

ShowNotice(i, FPosition(SIZE\_X - 190, SIZE\_Y - (70 \* (i + 1))));

}

}

}

//顯示提醒

void FBottomRightNotice::ShowNotice(int id,

const FPosition& leftLoc)

{

//使用特效

if(useEffect)

{

maskEffect[id].SetTopLeft(leftLoc.\_pX + 40, leftLoc.\_pY + 15);

maskEffect[id].OnShow();

noticesEffect[notices[id]].SetTopLeft(leftLoc.\_pX - 5,

leftLoc.\_pY - 15);

noticesEffect[notices[id]].OnShow();

}

else

{

maskImage.SetTopLeft(leftLoc.\_pX + 40, leftLoc.\_pY + 15);

maskImage.SetAlpha(120);

maskImage.ShowBitmap();

ShowText(FPosition(leftLoc.\_pX + 40, leftLoc.\_pY + 15),

name[id]);

FResource::resImages[notices[id]].SetTopLeft(leftLoc.\_pX - 5,

leftLoc.\_pY - 15);

FResource::resImages[notices[id]].ShowBitmap();

}

}

//顯示資源名稱

void FBottomRightNotice::ShowText(const FPosition& leftLoc,

const string& message)

{

static FOutlineText text("Microsoft JhengHei", 26, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f);

static FOutlineText textS("Microsoft JhengHei", 18, RGB(255,

255, 255), RGB(0,

0, 0), 3.5f, FW\_HEAVY);

if(message.find(' ') == string::npos)

text.ShowText(message.c\_str(), leftLoc.\_pX + 50,

leftLoc.\_pY + 7);

else

textS.ShowText(message.c\_str(), leftLoc.\_pX + 40,

leftLoc.\_pY + 14);

}

}

/\* FileName: FBottomRightNotice.h \*/

﻿#pragma once

#ifndef FORAGER\_FBOTTOMRIGHTNOTICE\_H

#define FORAGER\_FBOTTOMRIGHTNOTICE\_H

#include "FBottomRightNoticeEffect.h"

#include "FPosition.h"

#include "FImage.h"

#include "CGameStdLib.h"

#include "FResource.h"

#include "FText.h"

#include "FOutlineText.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

namespace game\_framework

{

class FBottomRightNotice

{

public:

void Initialize();

void OnShow();

void AddNotice(FResource\* res);

void ShowNotice(int id, const FPosition& leftLoc);

void ShowText(const FPosition& leftLoc, const string& message);

void DelNotice();

private:

bool useEffect = false;

vector<int> notices;

vector<clock\_t> times;

vector<string> name;

FImage maskImage;

FBottomRightNoticeEffect maskEffect[5];

FBottomRightNoticeEffect

noticesEffect[FResource::ResourceID::RESOURCEIDEND];

};

}

#endif // !FORAGER\_FBOTTOMRIGHTNOTICE\_H

/\* FileName: FBottomRightNoticeEffect.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FBottomRightNoticeEffect.h"

using namespace std;

namespace game\_framework

{

//初始化

void FBottomRightNoticeEffect::Initialize(FImage\* image)

{

ID2D1Bitmap\* bmpTemp = image->GetBmpRes();

this->srcImage = image;

this->w = (int)bmpTemp->GetSize().width;

this->h = (int)bmpTemp->GetSize().height;

D2DCallBack();

}

//設定特效螢幕位置

void FBottomRightNoticeEffect::SetTopLeft(int x, int y)

{

this->x = x;

this->y = y;

//特效第一段, 放大圖片

if(first)

{

if(this->scale < 1.2f)

this->scale += 0.2f;

else

first = false;

}

//特效第二段, 縮小圖片

else

{

if(this->scale > 1)

{

this->scale -= 0.04f;

if(this->scale < 1)

this->scale = 1;

}

}

}

//顯示特效

void FBottomRightNoticeEffect::OnShow()

{

D2D1Draw::DrawEffect(alphaEffect.Get(), this->x, this->y, w, h,

scale, scale);

}

//重置縮放比例

void FBottomRightNoticeEffect::ResetScale()

{

this->scale = 0.4f;

this->first = true;

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FBottomRightNoticeEffect::D2DCallBack()

{

if(!srcImage)

return;

D2D1Draw::CreateEffect(CLSID\_D2D1GammaTransfer,

gammaEffect.ReleaseAndGetAddressOf());

D2D1Draw::CreateEffect(CLSID\_D2D1Opacity,

alphaEffect.ReleaseAndGetAddressOf());

gammaEffect->SetInput(0, srcImage->GetBmpRes());

gammaEffect->SetValue(D2D1\_GAMMATRANSFER\_PROP\_RED\_EXPONENT,

0.1f);

gammaEffect->SetValue(D2D1\_GAMMATRANSFER\_PROP\_GREEN\_EXPONENT,

0.1f);

gammaEffect->SetValue(D2D1\_GAMMATRANSFER\_PROP\_BLUE\_EXPONENT,

0.1f);

gammaEffect->GetOutput(

gammaEffectImage.ReleaseAndGetAddressOf());

alphaEffect->SetInput(0, gammaEffectImage.Get());

alphaEffect->SetValue(D2D1\_OPACITY\_PROP\_OPACITY, 0.9f);

}

//destructor 解除註冊 Callback Function

FBottomRightNoticeEffect::~FBottomRightNoticeEffect()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

//constructor 註冊 Callback Function

FBottomRightNoticeEffect::FBottomRightNoticeEffect()

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FBottomRightNoticeEffect::D2DCallBack, this));

}

}

/\* FileName: FBottomRightNoticeEffect.h \*/

﻿#pragma once

#ifndef FORAGER\_FBOTTOMRIGHTNOTICEEFFECT\_H

#define FORAGER\_FBOTTOMRIGHTNOTICEEFFECT\_H

#include "FImage.h"

#include "CGameStdLib.h"

using namespace std;

namespace game\_framework

{

class FBottomRightNoticeEffect

{

public:

FBottomRightNoticeEffect();

~FBottomRightNoticeEffect();

void Initialize(FImage\* image);

void OnShow();

void SetTopLeft(int x, int y);

void D2DCallBack();

void ResetScale();

private:

FLOAT scale = 0.4f;

bool first = true;

int x = 0, y = 0, w = 0, h = 0;

int callBackID;

ComPtr<ID2D1Effect> gammaEffect;

ComPtr<ID2D1Effect> alphaEffect;

ComPtr<ID2D1Image> gammaEffectImage;

FImage\* srcImage = nullptr;

};

}

#endif // !FORAGER\_FBOTTOMRIGHTNOTICEEFFECT\_H

/\* FileName: FBuildInfo.cpp \*/

﻿#include "stdafx.h"

#include "FBuildInfo.h"

using namespace std;

namespace game\_framework

{

vector<FResReq> FBuildInfo::productionResReq[PRODUCTIONNAMEEND];

string FBuildInfo::productionName[PRODUCTIONNAMEEND] =

{

"FURNACE",

"FORGE",

"COAL",

"BRICK",

"GOLD IGNOT",

"IRON IGNOT",

"STEEL",

"GLASS",

"CROSS",

"CROSS2",

"CROSS3",

"TORCH"

};

int FBuildInfo::thisResId[PRODUCTIONNAMEEND] =

{

FResource::NOTHING,

FResource::NOTHING,

FResource::COAL\_R,

FResource::BRICK\_R,

FResource::GOLD\_IGNOT\_R,

FResource::IRON\_IGNOT\_R,

FResource::STEEL\_R,

FResource::GLASS\_R,

FResource::NOTHING,

FResource::NOTHING,

FResource::NOTHING,

FResource::NOTHING

};

//constructor

FResReq::FResReq(int id, int count)

{

this->id = id;

this->count = count;

}

//初始化合成品合成需求

void FBuildInfo::productionInfoInitialize()

{

productionResReq[FURNACE] = { {FResource::ResourceID::STONE\_R, 2} };

productionResReq[FORGE] = { {FResource::ResourceID::STONE\_R, 2}, {FResource::ResourceID::GOLD\_IGNOT\_R, 2} };

productionResReq[COAL] = { {FResource::ResourceID::WOOD\_R, 2} };

productionResReq[BRICK] = { {FResource::ResourceID::STONE\_R, 2}, {FResource::ResourceID::COAL\_R, 2} };

productionResReq[GOLD\_IGNOT] = { {FResource::ResourceID::STONE\_R, 2} };

productionResReq[IRON\_IGNOT] = { {FResource::ResourceID::STONE\_R, 2} };

productionResReq[STEEL] = { {FResource::ResourceID::IRON\_IGNOT\_R, 2}, {FResource::ResourceID::GOLD\_IGNOT\_R, 2}, {FResource::ResourceID::COAL\_R, 2} };

productionResReq[GLASS] = { {FResource::ResourceID::STONE\_R, 2} };

productionResReq[PICKAXE] = { {FResource::ResourceID::WOOD\_R, 1} };

productionResReq[PICKAXE2] = { {FResource::ResourceID::WOOD\_R, 1} };

productionResReq[PICKAXE3] = { {FResource::ResourceID::WOOD\_R, 1} };

productionResReq[TORCH] = { {FResource::ResourceID::WOOD\_R, 1} };

}

//給予合成品對應的名字及合成需求

void FBuildInfo::Initialize(int name)

{

if(name == NOTHING)

{

this->name = "???????";

return;

}

this->name = productionName[name];

this->req = productionResReq[name];

this->id = name;

}

//回傳合成品名字

const string& FBuildInfo::GetName()

{

return name;

}

//回傳合成需求種類數量

int FBuildInfo::GetResTypeCount()

{

return (int)req.size();

}

//回傳特定合成需求數量

int FBuildInfo::GetResCount(int i)

{

return req[i].count;

}

//回傳特定合成需求id

int FBuildInfo::GetResId(int i)

{

return req[i].id;

}

//回傳合成需求

vector<FResReq> FBuildInfo::GetReq()

{

return req;

}

//回傳特定合成需求圖片

FImage FBuildInfo::GetResImg(int i)

{

return FResource::resImages[req[i].id];

}

//回傳合成品id

int FBuildInfo::GetId()

{

return id;

}

//合成品圖片

FImage FBuildInfo::GetThisResImg()

{

return FResource::resImages[thisResId[id]];

}

//回傳合成品的資源id

int FBuildInfo::GetThisResId()

{

return thisResId[id];

}

}

/\* FileName: FBuildInfo.h \*/

﻿#ifndef FORAGER\_FBUILDINFO\_H

#define FORAGER\_FBUILDINFO\_H

#include "CGameStdLib.h"

#include "FResource.h"

#include "FImage.h"

#include <string>

#include <vector>

namespace game\_framework

{

class FResReq

{

public:

FResReq(int id, int count);

int id;

int count;

};

class FBuildInfo

{

public:

enum ProductionName

{

FURNACE,

FORGE,

COAL,

BRICK,

GOLD\_IGNOT,

IRON\_IGNOT,

STEEL,

GLASS,

PICKAXE,

PICKAXE2,

PICKAXE3,

TORCH,

PRODUCTIONNAMEEND,

NOTHING

};

static void productionInfoInitialize();

static int thisResId[PRODUCTIONNAMEEND];

void Initialize(int name);

const string& GetName();

int GetResTypeCount();

int GetResCount(int i);

int GetResId(int i);

int GetId();

int GetThisResId();

vector<FResReq> GetReq();;

FImage GetResImg(int i);

FImage GetThisResImg();

private:

string name;

vector<FResReq> req;

int id;

static vector<FResReq> productionResReq[PRODUCTIONNAMEEND];

static string productionName[PRODUCTIONNAMEEND];

};

}

#endif // !FORAGER\_FBUILDINFO\_H

/\* FileName: FBuildSystem.cpp \*/

﻿#include "stdafx.h"

#include "FBuildSystem.h"

using namespace std;

namespace game\_framework

{

//初始化建築系統

void FBuildSystem::Initialize(FMap\* map, FHuman\* human)

{

FBuildInfo::productionInfoInitialize();

this->nowChoiceBuild = NOTHING;

this->map = map;

this->human = human;

this->mode = MENU;

buildSystemMenu.Initialize(human);

redMask.LoadBitmap(IDB\_RED\_MASK);

redMask.SetAlpha(100);

isBuildable = false;

building\_img[INDUSTRIAL][FURNACE].LoadBitmap(IDB\_FURNACE,

RGB(192, 192, 192));

building\_img[INDUSTRIAL][FORGE].LoadBitmap(IDB\_FORGE, RGB(192,

192, 192));

building\_img[FARMING][TORCH].LoadBitmap(IDB\_TORCH, RGB(192, 192,

192));

buildXYDif[FARMING][TORCH] = FPosition(0, -20);

}

//對map上的targetBlockMapLoc位置建築

void FBuildSystem::Build(const FPosition& targetBlockMapLoc,

int buildingExtraX,

int buildingExtraY)

{

int targetBlockMapLocX = targetBlockMapLoc.\_pX;

int targetBlockMapLocY = targetBlockMapLoc.\_pY;

int imgPart = 0;

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

{

for(int j = 0; j <= buildingExtraX; j++, imgPart++)

map->m\_mapBlock[targetBlockMapLocY + i][targetBlockMapLocX +

j].pFrontImageInfo

= &FMapInfo::imageInfo[GetBuildMapInfoType() +

imgPart];

}

Reset();

}

//滑鼠左鍵事件

void FBuildSystem::OnLButtonDownHandle(const CPoint& point)

{

//模式為建築

if(mode == BUILD)

{

FPosition mouseLoc(FHuman::Loc.\_pX + (point.x - SIZE\_X / 2) \*

100 / 80,

FHuman::Loc.\_pY + (point.y - SIZE\_Y / 2) \* 100 / 80);

FPosition blockMapLoc(

mouseLoc.ConvertMouseToBlockMapPosition());

//若所選的位置可建築

if(IsBuildable(blockMapLoc,

FMapInfo::imageInfo[GetBuildMapInfoType()].buildingExtraX,

FMapInfo::imageInfo[GetBuildMapInfoType()].buildingExtraY))

{

FBuildInfo buildInfoTemp = buildSystemMenu.GetBuildInfo();

Build(blockMapLoc,

FMapInfo::imageInfo[GetBuildMapInfoType()].buildingExtraX,

FMapInfo::imageInfo[GetBuildMapInfoType()].buildingExtraY);

human->bag.ReduceRes(buildInfoTemp.GetReq());

}

}

//模式為選單

else if(mode == MENU)

{

int targetBuild = buildSystemMenu.GetTargetBuild(point);

//若所選的位置有按鈕

if(targetBuild != -1)

{

int choiceButton = buildSystemMenu.GetChoiceButton();

//若選的建築按鈕上有建築

if(IsHaveBuild(choiceButton, targetBuild))

{

FBuildInfo buildInfoTemp = buildSystemMenu.GetBuildInfo();

if(human->bag.IsEnoughToBuild(buildInfoTemp.GetReq()))

{

this->mode = BUILD;

this->nowChoiceBuild = targetBuild;

OnMouseMoveHandle(point);

}

}

}

}

}

//滑鼠移動事件

void FBuildSystem::OnMouseMoveHandle(const CPoint& point)

{

//模式為建築

if(this->mode == BUILD)

{

FPosition mouseLoc(FHuman::Loc.\_pX + (point.x - SIZE\_X / 2) \*

100 / 80,

FHuman::Loc.\_pY + (point.y - SIZE\_Y / 2) \* 100 / 80);

FPosition blockMapLoc(

mouseLoc.ConvertMouseToBlockMapPosition());

FPosition blockScreenTopLeft =

human->BlockMapLocToScreenTopLeftLoc(

blockMapLoc);

//所選建築跟著滑鼠動

building\_img[buildSystemMenu.GetChoiceButton()][this->nowChoiceBuild].SetTopLeft(

blockScreenTopLeft.\_pX +

buildXYDif[buildSystemMenu.GetChoiceButton()][this->nowChoiceBuild].\_pX,

blockScreenTopLeft.\_pY +

buildXYDif[buildSystemMenu.GetChoiceButton()][this->nowChoiceBuild].\_pY);

building\_img[buildSystemMenu.GetChoiceButton()][this->nowChoiceBuild].SetAlpha(

100);

//若滑鼠位置不可建築, 則顯示紅色遮罩

if(!IsBuildable(blockMapLoc,

FMapInfo::imageInfo[GetBuildMapInfoType()].buildingExtraX,

FMapInfo::imageInfo[GetBuildMapInfoType()].buildingExtraY))

{

redMask.SetTopLeft(blockScreenTopLeft.\_pX,

blockScreenTopLeft.\_pY,

FMapInfo::imageInfo[GetBuildMapInfoType()].imgWidth,

FMapInfo::imageInfo[GetBuildMapInfoType()].imgHeight);

this->isBuildable = false;

}

else

this->isBuildable = true;

}

//模式為選單

else if(this->mode == MENU)

buildSystemMenu.OnMouseMoveHandle(point);

}

//顯示建築系統

void FBuildSystem::ShowBuildSystem()

{

//模式為建築

if(this->mode == BUILD)

{

building\_img[buildSystemMenu.GetChoiceButton()][this->nowChoiceBuild].ShowBitmap();

if(!this->isBuildable)

redMask.ShowBitmap();

}

//模式為選單

else if(this->mode == MENU)

buildSystemMenu.ShomMenu();

}

//將建築轉換為MapInfoType

int FBuildSystem::GetBuildMapInfoType()

{

if(buildSystemMenu.GetChoiceButton() == INDUSTRIAL)

{

if(this->nowChoiceBuild == FURNACE)

return FMapInfo::Type::FURNACE\_1;

else if(this->nowChoiceBuild == FORGE)

return FMapInfo::Type::FORGE\_1;

}

else if(buildSystemMenu.GetChoiceButton() == FARMING)

{

if(this->nowChoiceBuild == TORCH)

return FMapInfo::Type::TORCH;

}

else if(buildSystemMenu.GetChoiceButton() == ECONOMIC)

{

}

return -1;

}

//判斷選的位置上是否有建築

bool FBuildSystem::IsHaveBuild(int choiceButton,

int targetBuild)

{

return ((choiceButton == INDUSTRIAL &&

targetBuild < INDUSTRIALEND)||

(choiceButton == FARMING && targetBuild < FARMINGEND)||

(choiceButton == ECONOMIC && targetBuild < ECONOMICEND));

}

//重置建築系統狀態

void FBuildSystem::Reset()

{

this->nowChoiceBuild = NOTHING;

this->mode = MENU;

buildSystemMenu.Reset();

}

//判斷此位置是否可建築

bool FBuildSystem::IsBuildable(const FPosition&

targetBlockMapLoc,

int buildingExtraX, int buildingExtraY)

{

int targetBlockMapLocX = targetBlockMapLoc.\_pX;

int targetBlockMapLocY = targetBlockMapLoc.\_pY;

FPosition humanMapLoc = human->Loc.ConvertHumanToMapPosition();

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

{

for(int j = 0; j <= buildingExtraY; j++)

{

if(!(map->m\_mapBlock[targetBlockMapLocY + j][targetBlockMapLocX

+

i].IsBuildable()) ||

(humanMapLoc.\_pX == (targetBlockMapLocX + i) &&

humanMapLoc.\_pY == (targetBlockMapLocY + j)))

return false;

}

}

return true;

}

}

/\* FileName: FBuildSystem.h \*/

﻿#ifndef FORAGER\_FBUILDSYSTEM\_H

#define FORAGER\_FBUILDSYSTEM\_H

#include "CGameStdLib.h"

#include "FMap.h"

#include "FHuman.h"

#include "FAtlas.h"

#include "FImage.h"

#include "FBuildSystemMenu.h"

#include "FBuildInfo.h"

#include "FPosition.h"

namespace game\_framework

{

class FMap;

class FBuildSystem

{

public:

enum BuildButton

{

INDUSTRIAL,

FARMING,

ECONOMIC,

BUILDBUTTONEEND,

NOTHING

};

enum Mode

{

BUILD,

MENU,

MODEEND

};

enum Industrial

{

FURNACE,

FORGE,

INDUSTRIALEND

};

enum Farming

{

TORCH,

FARMINGEND

};

enum Economic

{

ECONOMICEND

};

FBuildSystem() = default;

void Initialize(FMap\* map, FHuman\* human);

void OnLButtonDownHandle(const CPoint& point);

void OnMouseMoveHandle(const CPoint& point);

void ShowBuildSystem();

void Build(const FPosition& targetBlockMapLoc,

int buildingExtraX,

int buildingExtraY);

void Reset();

private:

int GetBuildMapInfoType();

bool IsBuildable(const FPosition& targetBlockMapLoc,

int buildingExtraX,

int buildingExtraY);

bool IsHaveBuild(int choiceButton, int targetBuild);

FBuildSystemMenu buildSystemMenu;

FAtlas industrial\_img;

FAtlas Farming\_img;

FAtlas Economic\_img;

FImage building\_img[FBuildSystemMenu::BuildButton::BUILDBUTTONEEND][FBuildSystemMenu::BUILD\_TYPE\_COUNT];

FImage redMask;

bool isBuildable;

int nowChoiceBuild;

int mode;

FPosition buildXYDif[FBuildSystemMenu::BuildButton::BUILDBUTTONEEND][FBuildSystemMenu::BUILD\_TYPE\_COUNT];

FMap\* map;

FHuman\* human;

};

}

#endif // !FORAGER\_FBUILDSYSTEM\_H

/\* FileName: FBuildSystemMenu.cpp \*/

﻿#include "stdafx.h"

#include "FBuildSystemMenu.h"

using namespace std;

namespace game\_framework

{

//初始化建築系統選單,

void FBuildSystemMenu::Initialize(FHuman\* human)

{

//選單按鈕位置初始化

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

{

this->buttonLoc[i] = FPosition(BUTTON\_X,

BUTTON\_Y + (i\*(BUTTON\_HEIGHT + DIS\_BETWEEN\_BUTTON)));

for(int j = 0; j < BUILD\_TYPE\_COUNT; j++)

this->button\_minLoc[i][j] = FPosition(BUTTON\_X + 20,

BUTTON\_Y + ((i + 1)\*(BUTTON\_HEIGHT + DIS\_BETWEEN\_BUTTON)) + (j\*

(BUTTON\_MIN\_HEIGHT + DIS\_BETWEEN\_BUTTON\_MIN)));

}

button.LoadBitmap(IDB\_BUILD\_BUTTON, BUILDBUTTONEEND, 1, 195, 60,

RGB(192, 192,

192));

button\_img.LoadBitmap(IDB\_BUILD\_BUTTON\_IMG, 1, BUILDBUTTONEEND,

80, 80, RGB(192,

192, 192));

button\_min.LoadBitmap(IDB\_BUILD\_BUTTON\_MIN, BUILDBUTTONEEND, 1,

155, 40,

RGB(192, 192, 192));

menu\_building\_img[INDUSTRIAL].LoadBitmap(IDB\_INDUSTRIAL\_IMG, 1,

FBuildSystemMenu::BUILD\_TYPE\_COUNT, 160, 160, RGB(192, 192,

192));

menu\_building\_img[FARMING].LoadBitmap(IDB\_FARMING\_IMG, 1,

FBuildSystemMenu::BUILD\_TYPE\_COUNT, 160, 160, RGB(192, 192,

192));

menu\_building\_img[ECONOMIC].LoadBitmap(IDB\_ECONOMIC\_IMG, 1,

FBuildSystemMenu::BUILD\_TYPE\_COUNT, 160, 160, RGB(255, 255,

255));

backGroundImage.LoadBitmap(IDB\_BAGBACKGROUND);

button\_Name[INDUSTRIAL] = "INDUSTRIAL";

button\_Name[FARMING] = "FARMING";

button\_Name[ECONOMIC] = "ECONOMIC";

//INDUSTRIAL建築需求初始化

buildInfo[INDUSTRIAL][0].Initialize(FBuildInfo::FURNACE);

buildInfo[INDUSTRIAL][1].Initialize(FBuildInfo::FORGE);

buildInfo[INDUSTRIAL][2].Initialize(FBuildInfo::NOTHING);

buildInfo[INDUSTRIAL][4].Initialize(FBuildInfo::NOTHING);

buildInfo[INDUSTRIAL][3].Initialize(FBuildInfo::NOTHING);

buildInfo[INDUSTRIAL][5].Initialize(FBuildInfo::NOTHING);

//FARMING建築需求初始化

buildInfo[FARMING][0].Initialize(FBuildInfo::TORCH);

buildInfo[FARMING][1].Initialize(FBuildInfo::NOTHING);

buildInfo[FARMING][2].Initialize(FBuildInfo::NOTHING);

buildInfo[FARMING][3].Initialize(FBuildInfo::NOTHING);

buildInfo[FARMING][4].Initialize(FBuildInfo::NOTHING);

buildInfo[FARMING][5].Initialize(FBuildInfo::NOTHING);

//ECONOMIC建築需求初始化

buildInfo[ECONOMIC][0].Initialize(FBuildInfo::NOTHING);

buildInfo[ECONOMIC][1].Initialize(FBuildInfo::NOTHING);

buildInfo[ECONOMIC][2].Initialize(FBuildInfo::NOTHING);

buildInfo[ECONOMIC][3].Initialize(FBuildInfo::NOTHING);

buildInfo[ECONOMIC][4].Initialize(FBuildInfo::NOTHING);

buildInfo[ECONOMIC][5].Initialize(FBuildInfo::NOTHING);

backGroundImage.SetTopLeft(BUTTON\_X - 20, 0);

backGroundImage.SetAlpha(BACKGROUND\_ALPHA);

choiceButton = NOTHING;

mouseTarget.Initialize();

isPointingBuild = false;

this->human = human;

}

//顯示選單

void FBuildSystemMenu::ShomMenu()

{

//先印背景圖片

backGroundImage.ShowBitmap();

//印大按鈕

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

{

button.SetTopLeft(buttonLoc[i].\_pX, buttonLoc[i].\_pY);

button\_img.SetTopLeft(buttonLoc[i].\_pX - 10,

buttonLoc[i].\_pY - 10);

button.ShowPartBitmap(i);

button\_img.ShowPartBitmap(i);

static FOutlineText text("Microsoft JhengHei", 20, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f);

string temp\_str(button\_Name[i]);

text.ShowText(temp\_str.c\_str(), buttonLoc[i].\_pX + 70,

buttonLoc[i].\_pY + 20);

}

//若有選擇大按鈕, 則印小按鈕

if(this->choiceButton != NOTHING)

{

for(int i = 0; i < BUILD\_TYPE\_COUNT; i++)

{

button\_min.SetTopLeft(button\_minLoc[choiceButton][i].\_pX,

button\_minLoc[choiceButton][i].\_pY);

button\_min.ShowPartBitmap(choiceButton);

menu\_building\_img[choiceButton].SetTopLeft(

button\_minLoc[choiceButton][i].\_pX,

button\_minLoc[choiceButton][i].\_pY,

BUTTON\_MIN\_HEIGHT, BUTTON\_MIN\_HEIGHT);

menu\_building\_img[GetChoiceButton()].SetAlpha(255);

menu\_building\_img[choiceButton].ShowPartBitmap(i);

static FOutlineText text("Microsoft JhengHei", 20, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f);

string temp\_str(buildInfo[choiceButton][i].GetName());

text.ShowText(temp\_str.c\_str(),

button\_minLoc[choiceButton][i].\_pX + 50,

button\_minLoc[choiceButton][i].\_pY + 12);

}

//若滑鼠有放在小按鈕上, 則顯示建築需求

if(isPointingBuild)

{

FPosition resReqLoc(

button\_minLoc[choiceButton][nowTargetButton].\_pX -

RESREQ\_WIDTH - 20,

button\_minLoc[choiceButton][nowTargetButton].\_pY);

button.SetTopLeft(resReqLoc.\_pX, resReqLoc.\_pY, RESREQ\_WIDTH,

100 + (buildInfo[choiceButton][nowTargetButton].GetResTypeCount()

- 1) \* 70);

button.ShowPartBitmap(choiceButton);

for(int i = 0;

i < buildInfo[choiceButton][nowTargetButton].GetResTypeCount();

i++)

{

FImage resTemp =

buildInfo[choiceButton][nowTargetButton].GetResImg(i);

resTemp.SetTopLeft(resReqLoc.\_pX, resReqLoc.\_pY + (70 \* i));

resTemp.ShowBitmap();

static FOutlineText text("Microsoft JhengHei", 20, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f);

string temp\_str(to\_string(human->bag.GetBagResCount(

buildInfo[choiceButton][nowTargetButton].GetResId(

i))) + "/" + to\_string(

buildInfo[choiceButton][nowTargetButton].GetResCount(i)));

text.ShowText(temp\_str.c\_str(), resReqLoc.\_pX + 100,

resReqLoc.\_pY + 40 + 70 \* i);

}

}

}

mouseTarget.OnShow();

}

//回傳point上的按鈕索引

int FBuildSystemMenu::GetTargetBuild(const CPoint& point)

{

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

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + BUTTON\_WIDTH &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + BUTTON\_HEIGHT)

{

SetChoiceButton(i);

return -1;

}

}

for(int i = 0; i < BUILD\_TYPE\_COUNT; i++)

{

if(point.x >= button\_minLoc[this->choiceButton][i].\_pX &&

point.x < button\_minLoc[this->choiceButton][i].\_pX +

BUTTON\_WIDTH

&& point.y >= button\_minLoc[this->choiceButton][i].\_pY &&

point.y < button\_minLoc[this->choiceButton][i].\_pY +

BUTTON\_HEIGHT)

return i;

}

SetChoiceButton(NOTHING);

return -1;

}

//滑鼠移動事件

void FBuildSystemMenu::OnMouseMoveHandle(const CPoint& point)

{

int targetButtonType;

int targetButton = GetTargetButton(point, &targetButtonType);

//若有選到按鈕

if(targetButton != -1)

{

//若選到大按鈕

if(targetButtonType == BIG)

{

mouseTarget.SetTopLeftWidth(buttonLoc[targetButton],

BUTTON\_WIDTH + 40,

BUTTON\_HEIGHT + 40, -20, -20);

mouseTarget.SetPointing(true);

this->isPointingBuild = false;

}

//若選到小按鈕

else if(targetButtonType == SMALL)

{

mouseTarget.SetTopLeftWidth(

button\_minLoc[choiceButton][targetButton],

BUTTON\_MIN\_WIDTH + 40, BUTTON\_MIN\_HEIGHT + 40, -20, -20);

mouseTarget.SetPointing(true);

if(buildInfo[choiceButton][targetButton].GetResTypeCount() != 0)

{

this->isPointingBuild = true;

nowTargetButton = targetButton;

return;

}

}

this->isPointingBuild = false;

return;

}

this->isPointingBuild = false;

mouseTarget.SetPointing(false);

}

//重置建築選單狀態

void FBuildSystemMenu::Reset()

{

this->choiceButton = NOTHING;

mouseTarget.SetPointing(false);

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

this->buttonLoc[i].\_pY = BUTTON\_Y + (i\*(BUTTON\_HEIGHT +

DIS\_BETWEEN\_BUTTON));

}

//回傳所選建築的建築需求

FBuildInfo FBuildSystemMenu::GetBuildInfo()

{

return buildInfo[choiceButton][nowTargetButton];

}

//回傳目前選的建築種類索引

int FBuildSystemMenu::GetChoiceButton()

{

return this->choiceButton;

}

//設定選擇按鈕(人性化需求 如:點擊旁邊, 下拉事選單收起)

void FBuildSystemMenu::SetChoiceButton(int choiceButton)

{

this->choiceButton = choiceButton;

ChangeButtonLoc();

}

//改變按鈕位置(下拉式選單)

void FBuildSystemMenu::ChangeButtonLoc()

{

if(choiceButton == NOTHING) //還原

Reset();

else

{

int distance = (BUTTON\_MIN\_HEIGHT + DIS\_BETWEEN\_BUTTON\_MIN)\*

BUILD\_TYPE\_COUNT;

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

this->buttonLoc[i].\_pY = BUTTON\_Y + (i\*(BUTTON\_HEIGHT +

DIS\_BETWEEN\_BUTTON));

for(int i = this->choiceButton + 1; i < BUILDBUTTONEEND; i++)

this->buttonLoc[i].\_pY = BUTTON\_Y + (i\*(BUTTON\_HEIGHT +

DIS\_BETWEEN\_BUTTON)) +

distance;

}

}

//回傳point位置按鈕索引

int FBuildSystemMenu::GetTargetButton(const CPoint& point,

int\* targetButtonType)

{

//判斷大按鈕

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

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + BUTTON\_WIDTH &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + BUTTON\_HEIGHT)

{

\*targetButtonType = BIG;

return i;

}

}

//判斷小按鈕

for(int i = 0; i < BUILD\_TYPE\_COUNT; i++)

{

if(point.x >= button\_minLoc[this->choiceButton][i].\_pX &&

point.x < button\_minLoc[this->choiceButton][i].\_pX +

BUTTON\_WIDTH

&& point.y >= button\_minLoc[this->choiceButton][i].\_pY &&

point.y < button\_minLoc[this->choiceButton][i].\_pY +

BUTTON\_HEIGHT)

{

\*targetButtonType = SMALL;

return i;

}

}

return -1;

}

}

/\* FileName: FBuildSystemMenu.h \*/

﻿#ifndef FORAGER\_FBUILDSYSTEMMENU\_H

#define FORAGER\_FBUILDSYSTEMMENU\_H

#include "CGameStdLib.h"

#include "FAtlas.h"

#include "FImage.h"

#include "FMouseTarget.h"

#include "FPosition.h"

#include "FText.h"

#include "FHuman.h"

#include "FBuildInfo.h"

#include "FResource.h"

#include <string>

namespace game\_framework

{

class FMap;

class FBuildSystemMenu

{

public:

enum BuildButton

{

INDUSTRIAL,

FARMING,

ECONOMIC,

BUILDBUTTONEEND,

NOTHING

};

enum ButtonType

{

BIG,

SMALL,

BUTTONTYPEEND

};

static const int DIS\_BETWEEN\_BUTTON = 20;

static const int DIS\_BETWEEN\_BUTTON\_MIN = 10;

static const int BACKGROUND\_ALPHA = 80; //背景透明度

static const int BUTTON\_WIDTH = 195; //按鈕寬度

static const int BUTTON\_HEIGHT = 60; //按鈕高度

static const int BUTTON\_MIN\_WIDTH = 155; //小按鈕寬度

static const int BUTTON\_MIN\_HEIGHT = 40; //小按鈕高度

static const int BUTTON\_X = 1065; //按鈕起始X

static const int BUTTON\_Y = 60; //按鈕起始Y

static const int BUILD\_TYPE\_COUNT =

6; //每種建築種類數量

static const int RESREQ\_WIDTH = 165; //資源提示選單寬度

void Initialize(FHuman\* human);

void ShomMenu();

void OnMouseMoveHandle(const CPoint& point);

void SetChoiceButton(int choiceButton);

void Reset();

FBuildInfo GetBuildInfo();

int GetTargetBuild(const CPoint& point);

int GetChoiceButton();

string button\_Name[BUILDBUTTONEEND];

private:

void ChangeButtonLoc();

int GetTargetButton(const CPoint& point, int\* targetButtonType);

FImage backGroundImage; //背景圖片

FAtlas button;

FAtlas button\_min;

FAtlas button\_img;

FAtlas menu\_building\_img[BUILDBUTTONEEND];

FBuildInfo buildInfo[BUILDBUTTONEEND][BUILD\_TYPE\_COUNT];

FPosition buttonLoc[BUILDBUTTONEEND];

FPosition button\_minLoc[BUILDBUTTONEEND][BUILD\_TYPE\_COUNT];

FMouseTarget mouseTarget;

int choiceButton;

int nowTargetButton;

bool isPointingBuild;

FHuman\* human;

};

}

#endif // !FORAGER\_FBUILDSYSTEMMENU\_H

/\* FileName: FCamera.cpp \*/

﻿#include "stdafx.h"

#include "FCamera.h"

namespace game\_framework

{

//初始化

void FCamera::Initialize(FMap\* map, FHuman\* human)

{

this->map = map;

this->human = human;

this->seaEdge.LoadBitmap(IDB\_SEAEDGE, 2, 4, 88, 88, RGB(255,

255, 255));

this->nightEffect.Init();

}

//計算所有座標

void FCamera::CalculatedPosition()

{

this->humanMapLoc = human->Loc.ConvertHumanToMapPosition();

this->humanScreenLoc =

human->Loc.ConvertHumanToScreenPosition();

this->mapx\_left = humanMapLoc.\_pX - (FMap::PROTECT\_X / 2 + 2);

this->mapx\_right = humanMapLoc.\_pX + (FMap::PROTECT\_X / 2 + 2);

this->mapy\_up = humanMapLoc.\_pY - (FMap::PROTECT\_Y / 2 + 2);

this->mapy\_down = humanMapLoc.\_pY + (FMap::PROTECT\_Y / 2 + 2);

}

//印背景

void FCamera::ShowBackground()

{

//印草

for(int x = mapx\_left; x <= mapx\_right; x++)

for(int y = mapy\_up; y <= mapy\_down; y++)

{

FMapInfo\* block = map->GetDangerMapInfo(x, y);

FMapImage\* backBlock = block->pBackImageInfo;

if(backBlock->type != FMapInfo::GRASS)

continue;

if(backBlock)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

backBlock->SetImageTopLeft(sx, sy);

backBlock->ShowBitmap(false, block->weedType);

}

}

//印海

for(int x = mapx\_left; x <= mapx\_right; x++)

for(int y = mapy\_up; y <= mapy\_down; y++)

{

FMapInfo\* block = map->GetDangerMapInfo(x, y);

FMapImage\* backBlock = block->pBackImageInfo;

if(backBlock->type != FMapInfo::SEA)

continue;

if(backBlock)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

backBlock->SetImageTopLeft(sx, sy);

backBlock->ShowBitmap(false, block->weedType);

if(block->useSeaEffect)

block->seaEffect.OnShowAt((x % 2)? (4 - (x % 5)) : x % 5, sx,

sy);

ShowEdge(sx, sy, block->edgeFlag);

}

}

}

//印海的邊緣(有海的地方, 附近的草會多一條邊緣)

void FCamera::ShowEdge(int sx, int sy, int edgeFlag)

{

seaEdge.SetTopLeft(sx - 4, sy - 4);

if(edgeFlag & SEAEDGE\_U)

seaEdge.ShowPartBitmap(0);

if(edgeFlag & SEAEDGE\_D)

seaEdge.ShowPartBitmap(1);

if(edgeFlag & SEAEDGE\_L)

seaEdge.ShowPartBitmap(2);

if(edgeFlag & SEAEDGE\_R)

seaEdge.ShowPartBitmap(3);

if(edgeFlag & SEAEDGE\_UL)

seaEdge.ShowPartBitmap(4);

if(edgeFlag & SEAEDGE\_DL)

seaEdge.ShowPartBitmap(5);

if(edgeFlag & SEAEDGE\_UR)

seaEdge.ShowPartBitmap(6);

if(edgeFlag & SEAEDGE\_DR)

seaEdge.ShowPartBitmap(7);

}

//印前景

void FCamera::ShowForeground()

{

//所有前景資源

for(int x = mapx\_left; x <= mapx\_right; x++)

for(int y = mapy\_up; y <= humanMapLoc.\_pY; y++)

{

FMapInfo\* block = map->GetDangerMapInfo(x, y);

FMapImage\* frontBlock = block->pFrontImageInfo;

if(frontBlock)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

frontBlock->SetImageTopLeft(sx, sy);

frontBlock->ShowBitmap(block->useEffect);

if(frontBlock->type == FMapInfo::Type::TORCH)

nightEffect.RegisterTorch(sx, sy);

}

}

//印人類

human->OnShow(humanScreenLoc.\_pX - (humanScreenLoc.\_pX +

correctX),

humanScreenLoc.\_pY - (humanScreenLoc.\_pY + correctY));

//有些資源的高度超過兩格, 為了把人類擋住, 必須再印一次

for(int x = mapx\_left; x <= mapx\_right; x++)

for(int y = humanMapLoc.\_pY; y <= mapy\_down; y++)

{

FMapInfo\* block = map->GetDangerMapInfo(x, y);

FMapImage\* frontBlock = block->pFrontImageInfo;

if(frontBlock)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

if(y == humanMapLoc.\_pY && -correctY - sy > 40)

continue;

frontBlock->SetImageTopLeft(sx, sy);

frontBlock->ShowBitmap(block->useEffect);

if(frontBlock->type == FMapInfo::Type::TORCH)

nightEffect.RegisterTorch(sx, sy);

}

}

//晚上效果

nightEffect.OnShow();

}

//隱藏人類(死亡時人類會消失)

void FCamera::HideHuman()

{

CalculatedPosition();

FCamera::ShowBackground();

for(int x = mapx\_left; x <= mapx\_right; x++)

for(int y = mapy\_up; y <= humanMapLoc.\_pY; y++)

{

FMapInfo\* block = map->GetDangerMapInfo(x, y);

FMapImage\* frontBlock = block->pFrontImageInfo;

if(frontBlock)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

frontBlock->SetImageTopLeft(sx, sy);

frontBlock->ShowBitmap(block->useEffect);

}

}

for(int x = mapx\_left; x <= mapx\_right; x++)

for(int y = humanMapLoc.\_pY; y <= mapy\_down; y++)

{

FMapInfo\* block = map->GetDangerMapInfo(x, y);

FMapImage\* frontBlock = block->pFrontImageInfo;

if(frontBlock)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

if(y == humanMapLoc.\_pY && -correctY - sy > 40)

continue;

frontBlock->SetImageTopLeft(sx, sy);

frontBlock->ShowBitmap(block->useEffect);

if(frontBlock->type == FMapInfo::Type::TORCH)

nightEffect.RegisterTorch(sx, sy);

}

}

nightEffect.OnShow();

}

}

/\* FileName: FCamera.h \*/

﻿#ifndef FORAGER\_FCAMERA\_H

#define FORAGER\_FCAMERA\_H

#include "FMap.h"

#include "FHuman.h"

#include "FAtlas.h"

#include "FNightEffect.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FCamera

{

public:

void Initialize(FMap\* map, FHuman\* human);

void ShowBackground();

void ShowForeground();

void CalculatedPosition();

void HideHuman();

private:

void ShowEdge(int sx, int sy, int edgeFlag);

FMap\* map;

FHuman\* human;

FAtlas seaEdge;

int correctX = -SIZE\_X / 2, correctY = -SIZE\_Y / 2;

int mapx\_left = 0, mapx\_right = 0, mapy\_up = 0, mapy\_down = 0;

FPosition humanMapLoc;

FPosition humanScreenLoc;

FNightEffect nightEffect;

};

}

#endif // !FORAGER\_FCAMERA\_H

/\* FileName: FCollectHighLightEffect.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FCollectHighLightEffect.h"

using namespace std;

namespace game\_framework

{

// 初始化採集特效

void FCollectHighLightEffect::Initialize(FImage\* image)

{

ID2D1Bitmap\* bmpTemp = image->GetBmpRes();

this->srcImage = image;

this->w = (int)bmpTemp->GetSize().width;

this->h = (int)bmpTemp->GetSize().height;

D2DCallBack();

}

// 設定特效位置

void FCollectHighLightEffect::SetTopLeft(int x, int y)

{

this->x = x;

this->y = y;

if(this->scale > 1)

{

this->scale -= 0.02f;

if(this->scale < 1)

this->scale = 1;

}

}

// 顯示特效

void FCollectHighLightEffect::OnShow()

{

D2D1Draw::DrawEffect(gammaEffect.Get(), this->x, this->y, w, h,

scale, scale);

}

// 重置特效比例

void FCollectHighLightEffect::ResetScale()

{

this->scale = 1.2f;

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FCollectHighLightEffect::D2DCallBack()

{

if(!srcImage)

return;

D2D1Draw::CreateEffect(CLSID\_D2D1GammaTransfer,

gammaEffect.ReleaseAndGetAddressOf());

gammaEffect->SetInput(0, srcImage->GetBmpRes());

gammaEffect->SetValue(D2D1\_GAMMATRANSFER\_PROP\_RED\_EXPONENT,

0.1f);

gammaEffect->SetValue(D2D1\_GAMMATRANSFER\_PROP\_GREEN\_EXPONENT,

0.1f);

gammaEffect->SetValue(D2D1\_GAMMATRANSFER\_PROP\_BLUE\_EXPONENT,

0.1f);

}

// destructor 解除註冊 Callback Function

FCollectHighLightEffect::~FCollectHighLightEffect()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

// constructor 註冊 Callback Function

FCollectHighLightEffect::FCollectHighLightEffect()

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FCollectHighLightEffect::D2DCallBack, this));

}

}

/\* FileName: FCollectHighLightEffect.h \*/

﻿#pragma once

#ifndef FORAGER\_FCOLLECTHIGHLIGHTEFFECT\_H

#define FORAGER\_FCOLLECTHIGHLIGHTEFFECT\_H

#include "FImage.h"

#include "CGameStdLib.h"

using namespace std;

namespace game\_framework

{

class FCollectHighLightEffect

{

public:

FCollectHighLightEffect();

~FCollectHighLightEffect();

void Initialize(FImage\* image);

void SetTopLeft(int x, int y);

void OnShow();

void D2DCallBack();

void ResetScale();

private:

FLOAT scale = 1.22f;

int x = 0, y = 0, w = 0, h = 0;

int callBackID;

ComPtr<ID2D1Effect> gammaEffect;

FImage\* srcImage = nullptr;

};

}

#endif // !FORAGER\_FCOLLECTHIGHLIGHTEFFECT\_H

/\* FileName: FCollectionBar.cpp \*/

﻿#include "StdAfx.h"

#include "FCollectionBar.h"

namespace game\_framework

{

//初始化採集特效及讀取圖片

void FCollectionBar::Initialize()

{

this->nowCount = 0;

this->collectCount = 0;

this->blockMapLoc = FPosition(0, 0);

this->dx = 18;

this->dy = 65;

this->imgWidth = 80;

this->imgHeight = 80;

this->collectionBarTime = 100;

this->collectionBarClock = 0;

this->isCollection = false;

this->bar.LoadBitmap(IDB\_BAR, 8, 5, 42, 14, RGB(100, 100, 100));

}

//顯示採集條

void FCollectionBar::OnShow(int sx, int sy)

{

if(this->isCollection)

{

bar.SetTopLeft(sx + dx + (imgWidth -80)/2,

sy + dy + (imgHeight - 80) + 5);

bar.ShowPartBitmap(int(nowCount));

if(collectionBarClock > 0)

collectionBarClock--;

else

Remove();

}

}

//採集條減少

void FCollectionBar::Collect(const FPosition& blockMapLoc,

int collectCount,

int imgWidth, int imgHeight)

{

this->isCollection = true;

this->collectionBarClock = this->collectionBarTime;

//當轉換目標或超過時間則重置採集條

if(this->reset || this->blockMapLoc.\_pX != blockMapLoc.\_pX ||

this->blockMapLoc.\_pY != blockMapLoc.\_pY)

{

this->blockMapLoc.\_pX = blockMapLoc.\_pX;

this->blockMapLoc.\_pY = blockMapLoc.\_pY;

this->collectCount = collectCount;

this->imgWidth = imgWidth;

this->imgHeight = imgHeight;

this->nowCount = (36.0 / this->collectCount);

this->reset = false;

}

else

this->nowCount += (36.0 /this->collectCount);

}

//判斷採集條是否採集完畢

bool FCollectionBar::IsZero(bool debugMode)

{

if(debugMode || this->nowCount >= 36)

{

this->isCollection = false;

this->reset = true;

return true;

}

return false;

}

//採集條不顯示

void FCollectionBar::Remove()

{

this->isCollection = false;

}

}

/\* FileName: FCollectionBar.h \*/

﻿#pragma once

#ifndef FORAGER\_FCOLLECTIONBAR\_H

#define FORAGER\_FCOLLECTIONBAR\_H

#include "FAtlas.h"

#include "FPosition.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FCollectionBar

{

public:

void Initialize();

void OnShow(int sx, int sy);

void Collect(const FPosition& BlockScreenTopLeftLoc,

int collectCount,

int imgWidth, int imgHeight);

bool IsZero(bool debugMode);

void Remove();

FPosition blockMapLoc;

private:

double nowCount;

int collectCount;

int dx, dy;//修正

int imgWidth;

int imgHeight;

bool isCollection, reset = false;

FAtlas bar;

int collectionBarTime;

int collectionBarClock;

};

}

#endif // !FORAGER\_FCOLLECTIONBAR\_H

/\* FileName: FCollectParticle.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FCollectParticle.h"

#include <cmath>

using namespace std;

namespace game\_framework

{

//初始化採集粒子特效位置及讀取圖片

void FCollectParticle::Initialize()

{

int fix = 10;

route[0] = FPosition(40, 25);

route[1] = FPosition(25 + fix, 25 + fix);

route[2] = FPosition(25, 40);

route[3] = FPosition(25 + fix, 55 - fix);

route[4] = FPosition(40, 55);

route[5] = FPosition(55 - fix, 55 - fix);

route[6] = FPosition(55, 40);

route[7] = FPosition(55 - fix, 25 + fix);

directional[0] = FPosition(0, -1);

directional[1] = FPosition(-1, -1);

directional[2] = FPosition(-1, 0);

directional[3] = FPosition(-1, 1);

directional[4] = FPosition(0, 1);

directional[5] = FPosition(1, 1);

directional[6] = FPosition(1, 0);

directional[7] = FPosition(1, -1);

for(int i = 0; i < 8; i++)

route[i] = FPosition(route[i].\_pX - 25, route[i].\_pY - 25);

baseImage.LoadBitmap(IDB\_COLLECTPARTICLE, RGB(195, 195, 195));

w = 50;

h = 50;

showCount = 60;

load = true;

}

//顯示採集粒子特效

void FCollectParticle::OnShow(int sx, int sy)

{

if(showCount < 60)

{

showCount += 2;

if(scale > 0)

scale += step;

else if(scale - 0.01f < 0)

scale = 0;

int offset = showCount;

for(int i = 0; i < 8; i++)

{

D2D1Draw::DrawBitmap(baseImage.GetBmpRes(),

sx + route[i].\_pX + directional[i].\_pX \* offset \* extendSize +

offsetX,

sy + route[i].\_pY + directional[i].\_pY \* offset \* extendSize +

offsetY,

1.0f - showCount \* 0.01f, w, h, NULL, scale, scale);

}

}

}

//設定粒子特效位置及縮放

void FCollectParticle::SetCollectParticle(const int& extendSize)

{

showCount = offsetX = offsetY = 0;

scale = 0.9f;

step = -0.08f;

this->extendSize = extendSize;

if(extendSize > 1)

{

offsetX = offsetY = 40 \* (extendSize - 1);

scale \*= extendSize \* extendSize \* 0.6f;

step \*= extendSize \* extendSize \* 0.6f;

}

}

}

/\* FileName: FCollectParticle.h \*/

﻿#pragma once

#ifndef FORAGER\_FCOLLECTPARTICLE\_H

#define FORAGER\_FCOLLECTPARTICLE\_H

#include "CGameStdLib.h"

#include "FPosition.h"

#include "FImage.h"

using namespace std;

namespace game\_framework

{

class FCollectParticle

{

public:

void Initialize();

void SetCollectParticle(const int& extendSize);

void OnShow(int sx, int sy);

private:

int showCount = 60, w = 50, h = 50, offsetX = 0, offsetY = 0,

extendSize = 1;

FLOAT scale = 0.9f, step = -0.08f;

FImage baseImage;

FPosition route[8];

FPosition directional[8];

bool load = false;

};

}

#endif // !FORAGER\_FCOLLECTPARTICLE\_H

/\* FileName: FEscMenu.cpp \*/

﻿#include "StdAfx.h"

#include "FEscMenu.h"

namespace game\_framework

{

//讀取圖片及設定座標

void FEscMenu::Initialize(FHuman\* human)

{

this->menu\_img.LoadBitmap(IDB\_ESC\_MENU, RGB(192, 192, 192));

this->backGroundImage.LoadBitmap(IDB\_BAGBACKGROUND);

this->orangeMask.LoadBitmap(IDB\_ORANGE\_MASK);

XAudio2Engine::instance.GetAudioVolume(&audioVolume);

audioVolume \*= 10;

subButton.LoadBitmap(IDB\_FUNCTION\_MENU\_SUB\_BUTTON, RGB(192, 192,

192));

addButton.LoadBitmap(IDB\_FUNCTION\_MENU\_ADD\_BUTTON, RGB(192, 192,

192));

healthDebug.LoadBitmap(IDB\_HEART, RGB(100, 100, 100));

collectDebug.LoadBitmap(IDB\_PICKAXE\_L, RGB(255, 255, 255));

buttonLoc[LEAVE] = FPosition(MENU\_IMG\_X + MENU\_PADDING,

MENU\_IMG\_Y + MENU\_IMG\_HEIGHT - LEAVE\_BUTTON\_HEIGHT -

MENU\_PADDING);

buttionTextDif[LEAVE] = FPosition(10, 15);

buttonLoc[SUBAUDIOVOLUME] = FPosition(MENU\_IMG\_X + MENU\_PADDING,

MENU\_IMG\_Y + MENU\_PADDING + 80);

buttonLoc[ADDAUDIOVOLUME] = FPosition(MENU\_IMG\_X +

MENU\_IMG\_WIDTH - MENU\_PADDING

- 40, MENU\_IMG\_Y + MENU\_PADDING + 80);

buttonLoc[HEALTHDEBUGMODE] = FPosition(MENU\_IMG\_X + MENU\_PADDING

+60,

MENU\_IMG\_Y + MENU\_IMG\_HEIGHT - 390);

buttonLoc[COLLECTDEBUGMODE] = FPosition(MENU\_IMG\_X +

MENU\_PADDING + 240,

MENU\_IMG\_Y + MENU\_IMG\_HEIGHT -400);

this->subButton.SetTopLeft(buttonLoc[SUBAUDIOVOLUME].\_pX,

buttonLoc[SUBAUDIOVOLUME].\_pY, 40, 60);

this->addButton.SetTopLeft(buttonLoc[ADDAUDIOVOLUME].\_pX,

buttonLoc[ADDAUDIOVOLUME].\_pY, 40, 60);

this->healthDebug.SetTopLeft(buttonLoc[HEALTHDEBUGMODE].\_pX,

buttonLoc[HEALTHDEBUGMODE].\_pY, 34, 30);

this->collectDebug.SetTopLeft(buttonLoc[COLLECTDEBUGMODE].\_pX,

buttonLoc[COLLECTDEBUGMODE].\_pY, 80, 80);

this->orangeMask.SetTopLeft(buttonLoc[LEAVE].\_pX,

buttonLoc[LEAVE].\_pY,

LEAVE\_BUTTON\_WIDTH, LEAVE\_BUTTON\_HEIGHT);

this->orangeMask.SetAlpha(100);

this->menu\_img.SetTopLeft(MENU\_IMG\_X, MENU\_IMG\_Y);

this->backGroundImage.SetTopLeft(MENU\_IMG\_X, MENU\_IMG\_Y,

MENU\_IMG\_WIDTH,

MENU\_IMG\_HEIGHT);

this->backGroundImage.SetAlpha(100);

this->isPointing = false;

this->isUseHealthDebug = false;

this->isUseCollectDebug = false;

this->human = human;

}

//顯示ESC選單

void FEscMenu::OnShow()

{

backGroundImage.ShowBitmap();

menu\_img.ShowBitmap();

subButton.ShowBitmap();

addButton.ShowBitmap();

healthDebug.ShowBitmap();

collectDebug.ShowBitmap();

if(isPointing)

orangeMask.ShowBitmap();

ShowText();

}

//顯示文字

void FEscMenu::ShowText()

{

string allText[BUTTONSEND] = { "SAVE AND EXIT TO MENU" };

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

{

static FText text(L"Consolas", 30, RGB(255, 255, 255));

text.ShowText(allText[i].c\_str(),

buttonLoc[i].\_pX+ buttionTextDif[i].\_pX,

buttonLoc[i].\_pY+buttionTextDif[i].\_pY);

}

if(XAudio2Engine::instance.IsPlayable())

{

static FText text(L"Consolas", 50, RGB(255, 255, 255));

text.ShowText(to\_string(int(audioVolume)).c\_str(),

MENU\_IMG\_X + MENU\_IMG\_WIDTH / 2 - 20,

MENU\_IMG\_Y + MENU\_PADDING + 77);

}

else

{

static FText text(L"Consolas", 50, RGB(255, 255, 255));

text.ShowText("X", MENU\_IMG\_X + MENU\_IMG\_WIDTH / 2 - 20,

MENU\_IMG\_Y + MENU\_PADDING + 77);

audioVolume = 10.0f;

}

if(isUseHealthDebug)

{

static FText text(L"Consolas", 30, RGB(255, 255, 255));

text.ShowText("(IS USING)", buttonLoc[HEALTHDEBUGMODE].\_pX - 60,

buttonLoc[HEALTHDEBUGMODE].\_pY + 41);

}

if(isUseCollectDebug)

{

static FText text(L"Consolas", 30, RGB(255, 255, 255));

text.ShowText("(IS USING)", buttonLoc[COLLECTDEBUGMODE].\_pX -40,

buttonLoc[COLLECTDEBUGMODE].\_pY + 50);

}

static FText text1(L"Consolas", 40, RGB(255, 255, 255));

text1.ShowText("Music Volume", MENU\_IMG\_X + MENU\_PADDING + 50,

MENU\_IMG\_Y + MENU\_PADDING + 40);

}

//判斷是否有按下離開按鈕

bool FEscMenu::GetExitGameSignal()

{

return exitSignal;

}

//若按下按鈕則設置訊號

void FEscMenu::SaveAndExitToMenu()

{

exitSignal = true;

}

//重置ESC選單狀態

void FEscMenu::Reset()

{

isPointing = false;

}

//滑鼠左鍵事件

void FEscMenu::OnLButtonDownHandle(const CPoint& point)

{

int targetButton = GetTargetButton(point);

if(targetButton != -1)

{

//若按下的是離開按鈕

if(targetButton == LEAVE)

SaveAndExitToMenu();

if(targetButton == ADDAUDIOVOLUME)

{

if(XAudio2Engine::instance.IsPlayable())

{

if(audioVolume < 50)

{

if(audioVolume < 10)

audioVolume++;

else if(audioVolume >= 10)

audioVolume += 10.0f;

XAudio2Engine::instance.SetAudioVolume(audioVolume / 10);

}

}

}

if(targetButton == SUBAUDIOVOLUME)

{

if(XAudio2Engine::instance.IsPlayable())

{

if(audioVolume > 0)

{

if(audioVolume <= 10)

audioVolume--;

else if(audioVolume > 10)

audioVolume -= 10.0f;

XAudio2Engine::instance.SetAudioVolume(audioVolume / 10);

}

}

}

if(targetButton == HEALTHDEBUGMODE)

{

isUseHealthDebug = !isUseHealthDebug;

human->humanHealth.SetDebugMode(isUseHealthDebug);

}

if(targetButton == COLLECTDEBUGMODE)

{

isUseCollectDebug = !isUseCollectDebug;

human->SetDebugMode(isUseCollectDebug);

}

return;

}

}

//滑鼠移動事件

void FEscMenu::OnMouseMoveHandle(const CPoint& point)

{

int targetButton = GetTargetButton(point);

if(targetButton != -1)

{

//若碰到的是離開按鈕則顯示遮罩

if(targetButton == LEAVE)

isPointing = true;

else if(targetButton == ADDAUDIOVOLUME)

addButton.SetScale(1.3f);

else if(targetButton == SUBAUDIOVOLUME)

subButton.SetScale(1.3f);

else if(targetButton == HEALTHDEBUGMODE)

healthDebug.SetScale(1.2f);

else if(targetButton == COLLECTDEBUGMODE)

collectDebug.SetScale(1.2f);

return;

}

addButton.SetScale(1.0f);

subButton.SetScale(1.0f);

if(isUseHealthDebug)

healthDebug.SetScale(1.3f);

else

healthDebug.SetScale(1.0f);

if(isUseCollectDebug)

collectDebug.SetScale(1.3f);

else

collectDebug.SetScale(1.0f);

isPointing = false;

}

//回傳point位置按鈕索引

int FEscMenu::GetTargetButton(const CPoint& point)

{

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

{

if(i == LEAVE)

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + LEAVE\_BUTTON\_WIDTH &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + LEAVE\_BUTTON\_HEIGHT)

return i;

}

else if(i == ADDAUDIOVOLUME || i == SUBAUDIOVOLUME)

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + 40 &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + 60)

return i;

}

else if(i == HEALTHDEBUGMODE)

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + 34 &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + 30)

return i;

}

else if(i == COLLECTDEBUGMODE)

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + 80 &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + 80)

return i;

}

}

return -1;

}

}

/\* FileName: FEscMenu.h \*/

﻿#pragma once

#ifndef FORAGER\_FESCMENU\_H

#define FORAGER\_FESCMENU\_H

#include "FImage.h"

#include "FMouseTarget.h"

#include "FHuman.h"

#include "FPosition.h"

#include "FText.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FEscMenu

{

public:

enum Buttons

{

LEAVE,

ADDAUDIOVOLUME,

SUBAUDIOVOLUME,

HEALTHDEBUGMODE,

COLLECTDEBUGMODE,

BUTTONSEND

};

static const int MENU\_IMG\_WIDTH = 400; //選單寬度

static const int MENU\_IMG\_HEIGHT =

571; //選單高度

static const int MENU\_IMG\_X = SIZE\_X / 2 -

MENU\_IMG\_WIDTH/2; //選單起始座標x

static const int MENU\_IMG\_Y =

110; //選單起始座標y

static const int MENU\_PADDING = 20; //選單內距

static const int LEAVE\_BUTTON\_WIDTH = MENU\_IMG\_WIDTH - 2 \*

MENU\_PADDING; //離開按鈕寬度

static const int LEAVE\_BUTTON\_HEIGHT =

75; //離開按鈕高度

void Initialize(FHuman\* human);

void OnShow();

void Reset();

bool GetExitGameSignal();

void OnLButtonDownHandle(const CPoint& point);

void OnMouseMoveHandle(const CPoint& point);

private:

FImage backGroundImage; //背景圖片

FImage orangeMask;

FImage menu\_img;

FImage addButton;

FImage subButton;

FImage healthDebug;

FImage collectDebug;

FPosition buttonLoc[BUTTONSEND];

FPosition buttionTextDif[BUTTONSEND]; //字位置誤差

FHuman\* human;

int GetTargetButton(const CPoint& point);

float audioVolume;

bool isPointing;

void ShowText();

void SaveAndExitToMenu();

bool exitSignal = false;

bool isUseHealthDebug;

bool isUseCollectDebug;

};

}

#endif // !FORAGER\_FESCMENU\_H

/\* FileName: FGameOverEffect.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FText.h"

#include "FOutlineText.h"

#include "FGameOverEffect.h"

#include "FSaveSystem.h"

using namespace std;

namespace game\_framework

{

unsigned char FGameOverEffect::pBuffer[SIZE\_X \* SIZE\_Y \* 4] = {0};

// 讀取圖片及設定位置

void FGameOverEffect::Initialize()

{

load = true;

x = 0;

y = 0;

mask.LoadBitmap(IDB\_RED\_MASK);

D2DCallBack();

}

// 顯示遊戲結束特效

void FGameOverEffect::OnShow()

{

static FText gameOverText(L"Arial", 130, RGB(255, 60, 60),

DWRITE\_FONT\_WEIGHT\_ULTRA\_BLACK);

static FText gameOverNoticeText(L"Microsoft JhengHei", 60,

RGB(255, 255, 225));

static FOutlineText gameOverScore("Microsoft JhengHei", 60,

RGB(255, 255, 225),

RGB(100, 100, 100), 0.f);

if(load)

{

// 縮放

if(scale > 1.2)

scale -= 0.05f;

D2D1Draw::DrawEffect(alphaEffect.Get(), this->x, this->y, w, h,

scale, scale,

0);

}

// 透明度

if(alpha > 0)

alpha -= 5;

else if(alpha < 0)

alpha = 0;

// 紅色遮罩

mask.SetTopLeft(0, 0, SIZE\_X, SIZE\_Y);

mask.SetAlpha(alpha);

mask.ShowBitmap();

// 顯示GAMEOVER文字

gameOverText.ShowText("Game Over", SIZE\_X / 2 - 395,

SIZE\_Y / 2 - 260);

gameOverNoticeText.ShowText("Your saved data will be deleted.",

SIZE\_X / 2 - 440, SIZE\_Y / 2 - 40);

int time = FSaveSystem::GetGameTime();

int hour = (time / 60) / 60;

int minute = (time / 60) - (hour \* 60);

int sec = time - (minute \* 60) - (hour \* 60 \* 60);

string temp\_str(to\_string((hour / 10) % 100) + to\_string(

hour % 10) + to\_string((minute / 10) % 100) + to\_string(

minute % 10) + to\_string((sec / 10) % 100) + to\_string(

sec % 10));

gameOverScore.ShowText(string("Score : " + temp\_str).c\_str(),

SIZE\_X / 2 - 210,

SIZE\_Y / 2 + 110);

}

// constructor 註冊 Callback Function

FGameOverEffect::FGameOverEffect()

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FGameOverEffect::D2DCallBack, this));

}

// destructor 解除註冊 Callback Function

FGameOverEffect::~FGameOverEffect()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FGameOverEffect::D2DCallBack()

{

if(load)

{

w = SIZE\_X;

h = SIZE\_Y;

\*(d2dBmpPtr.ReleaseAndGetAddressOf()) =

D2D1Draw::LoadMemoryBitmap(pBuffer,

SIZE\_X, SIZE\_Y);

D2D1Draw::CreateEffect(CLSID\_D2D1Opacity,

alphaEffect.ReleaseAndGetAddressOf());

alphaEffect->SetInput(0, d2dBmpPtr.Get());

alphaEffect->SetValue(D2D1\_OPACITY\_PROP\_OPACITY, 0.4f);

}

}

}

/\* FileName: FGameOverEffect.h \*/

﻿#pragma once

#ifndef FORAGER\_FGAMEOVEREFFECT\_H

#define FORAGER\_FGAMEOVEREFFECT\_H

#include "CGameStdLib.h"

#include "FImage.h"

using namespace std;

namespace game\_framework

{

class FGameOverEffect

{

public:

FGameOverEffect();

~FGameOverEffect();

void Initialize();

void OnShow();

void D2DCallBack();

static unsigned char pBuffer[SIZE\_X \* SIZE\_Y \* 4];

private:

int callBackID = 0, x = 0, y = 0, w = 0, h = 0, alpha = 150;

FLOAT scale = 1.5F;

ComPtr<ID2D1Bitmap> d2dBmpPtr = NULL;

ComPtr<ID2D1Effect> alphaEffect;

bool load = false;

FImage mask;

};

}

#endif // !FORAGER\_FGAMEOVEREFFECT\_H

/\* FileName: FGrassSeaEffect.cpp \*/

﻿#include "stdafx.h"

#include "Resource.h"

#include "FGrassSeaEffect.h"

using namespace std;

namespace game\_framework

{

int FGrassSeaEffect::seaEffectCount = 0;

FAtlas FGrassSeaEffect::imgBase;

//讀取圖片

void FGrassSeaEffect::Initialize()

{

FGrassSeaEffect::seaEffectCount = 0;

FGrassSeaEffect::imgBase.LoadBitmap(IDB\_GRASSSEAEFFECT, 4, 3,

80, 50);

}

//更新特效

void FGrassSeaEffect::NextEffect()

{

seaEffectCount++;

if(seaEffectCount > 20)

seaEffectCount = 3;

}

//回傳要顯示的圖片ID

int FGrassSeaEffect::SafeImgID(int imgID)

{

//海浪動畫速度

int offset = 3;

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

{

imgID += offset;

if(imgID > 9)

{

offset = -(offset);

imgID = 8;

}

else if(imgID < 0)

{

offset = -(offset);

imgID = 1;

}

}

return imgID;

}

//顯示

void FGrassSeaEffect::OnShowAt(int imgID, int x, int y)

{

imgBase.SetTopLeft(x, y);

imgBase.ShowPartBitmap(SafeImgID(imgID));

}

}

/\* FileName: FGrassSeaEffect.h \*/

﻿#pragma once

#ifndef FORAGER\_FGRASSSEAEFFECT\_H

#define FORAGER\_FGRASSSEAEFFECT\_H

#include "CGameStdLib.h"

#include "FAtlas.h"

using namespace std;

namespace game\_framework

{

class FGrassSeaEffect

{

public:

FGrassSeaEffect() = default;

static void Initialize();

static void NextEffect();

int SafeImgID(int imgID);

void OnShowAt(int imgID, int x, int y);

private:

static FAtlas imgBase;

static int seaEffectCount;

};

}

#endif // !FORAGER\_FGRASSSEAEFFECT\_H

/\* FileName: FHealth.cpp \*/

﻿#include "StdAfx.h"

#include "FHealth.h"

namespace game\_framework

{

//讀取圖片及設定座標

void FHealth::Initialize()

{

hpFront.LoadBitmap(IDB\_HEALTH\_FRONT, 3, 6, 34, 30, RGB(100, 100,

100));

hungerBar.LoadBitmap(IDB\_HUNGER\_BAR, 4, 5, 74, 26);

hungerBar.SetTopLeft(10, 50);

heartPosition[0] = FPosition(10, 10);

heartPosition[1] = FPosition(50, 10);

heartPosition[2] = FPosition(90, 10);

}

//扣血

void FHealth::SubBlood(int num)

{

#if defined(\_\_DDEBUG) || defined(\_DEBUG)

return; // debug 不扣血

#endif

damage += num;

if(heart == 0 && damage >= 16)

{

SetGameOver();

damage = 16;

//死亡音效

XAudio2Engine::instance.Play(".\\audio\\dead.wav");

return;

}

else if(damage > 16)

{

damage = 0;

heart--;

//扣血音效

XAudio2Engine::instance.Play(".\\audio\\getDamage.wav");

}

// 血量警示音效

if(heart == 0 && damage < 16)

{

if(damage >= 2 && damage < 6)

XAudio2Engine::instance.Play(".\\audio\\ekgsound0.wav");

else if(damage >= 6 && damage < 10)

XAudio2Engine::instance.Play(".\\audio\\ekgsound1.wav");

else if(damage >= 10)

XAudio2Engine::instance.Play(".\\audio\\ekgsound2.wav");

}

}

// 設定測試模式

void FHealth::SetDebugMode(bool mode)

{

this->debugMode = mode;

}

//回血(有飽食度時會回血)

void FHealth::AddBlood(int num)

{

damage -= num;

if(damage < 0 && heart < 2)

{

damage = 16;

heart++;

}

else if(damage < 0 && heart == 2)

damage = 0;

}

//設定GAMEOVER訊號

void FHealth::SetGameOver()

{

gameOver = true;

}

//回傳GAMEOVER訊號

bool FHealth::GetGameOver()

{

bool result = gameOver;

gameOver = false;

return result;

}

//飽食度控制

void FHealth::HungerHandle()

{

if(debugMode) return;

if(hpDelay >= 4)

{

zoom += step;

if(zoom >= 8 || zoom <= 0) step = -step;

hpDelay = 0;

}

if(hungerDelay >= 30)

{

if(hunger < 17)

{

hunger++;

AddBlood(2);

}

else if(hunger >= 17)

SubBlood();

hungerDelay = 0;

}

hpDelay++;

hungerDelay++;

}

//降低飽食度

void FHealth::SubHunger(int num)

{

this->hunger -= num;

if(this->hunger < 0)

this->hunger = 0;

}

//顯示血量及飽食度

void FHealth::OnShow()

{

this->HungerHandle();

for(int i = 0; i < 3; i++)

{

if(i == heart)

continue;

hpFront.SetTopLeft(heartPosition[i].\_pX, heartPosition[i].\_pY);

if(i < heart)

hpFront.ShowPartBitmap(0);

else

hpFront.ShowPartBitmap(16);

}

hpFront.SetTopLeft(heartPosition[heart].\_pX - zoom / 2,

heartPosition[heart].\_pY - zoom / 2, 34 + zoom, 30 + zoom);

hpFront.ShowPartBitmap(damage);

hungerBar.ShowPartBitmap(hunger);

}

//存檔用(讀取存檔資料)

void FHealth::SetSaveData(int hunger, int heart, int damage)

{

this->hunger = hunger;

this->heart = heart;

this->damage = damage;

}

//存檔用(資料存入檔案)

void FHealth::GetSaveData(int& hunger, int& heart, int& damage)

{

hunger = this->hunger;

heart = this->heart;

damage = this->damage;

}

}

/\* FileName: FHealth.h \*/

﻿#pragma once

#ifndef FORAGER\_FHEALTH\_H

#define FORAGER\_FHEALTH\_H

#include "FAtlas.h"

#include "FPosition.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FHealth

{

public:

void Initialize();

void SubBlood(int num = 1);

void AddBlood(int num = 1);

void OnShow();

bool GetGameOver();

void SetGameOver();

void HungerHandle();

void SubHunger(int num = 1);

void SetDebugMode(bool mode);

/\* 存檔用 \*/

void SetSaveData(int hunger, int heart, int damage);

void GetSaveData(int& hunger, int& heart, int& damage);

private:

int hunger = 0, heart = 0, damage = 0, hpDelay = 0,

hungerDelay = 0, zoom = 0,

step = 4;

FAtlas hpFront, hungerBar;

FPosition heartPosition[3];

bool gameOver = false, debugMode = false;

};

}

#endif // !FORAGER\_FHEALTH\_H

/\* FileName: FHuman.cpp \*/

﻿#include "stdafx.h"

#include "FHuman.h"

#include "FMap.h"

namespace game\_framework

{

FPosition FHuman::Loc;

//讀取動畫圖片及設定座標及設定初值

void FHuman::Initialize(FMap\* map)

{

//讀取動畫

human[STOP][FACERIGHT].LoadBitmap(IDB\_HUMAN\_R, 80, 240, 80, 80,

3, RGB(255, 0,

0));

human[STOP][FACERIGHT].SetDelayCount(5);

human[STOP][FACELEFT].LoadBitmap(IDB\_HUMAN\_L, 80, 240, 80, 80,

3, RGB(255, 0,

0));

human[STOP][FACELEFT].SetDelayCount(5);

human[RUN][FACERIGHT].LoadBitmap(IDB\_HUMAN\_RUN\_R, 80, 480, 80,

80, 6, RGB(255,

0, 0));

human[RUN][FACERIGHT].SetDelayCount(5);

human[RUN][FACELEFT].LoadBitmap(IDB\_HUMAN\_RUN\_L, 80, 480, 80,

80, 6, RGB(255, 0,

0));

human[RUN][FACELEFT].SetDelayCount(5);

//工具初始化

tool = nullptr;

nowTool = FTool::PICKAXE;

ChangeTool();

//人類位置

Loc = FPosition(FMap::MAPX\_SIZE / 2 \* 100,

FMap::MAPY\_SIZE / 2 \* 100);

//變數初始化

nowInteractiveBuild = FProductBuildSystem::NOTHING;

direction = FACERIGHT;

animationMode = STOP;

nowMouseMode = COLLECTION;

collectionRange = 1;

isMovingLeft = false;

isMovingRight = false;

isMovingUp = false;

isMovingDown = false;

isPointing = false;

isInteractable = false;

debugMode = false;

//包包初始化

bag.Initialize(this);

//採集條初始化

bar.Initialize();

//血條初始化

humanHealth.Initialize();

//採集粒子特效初始化

collectParticle.Initialize();

//滑鼠目標初始化

mouseTarget.Initialize();

//獲得地圖指標

this->map = map;

}

//更新

void FHuman::OnMove()

{

static const int STEP\_SIZE = 20;

int i = 0;

//若有按下方向鍵

if(isMovingLeft || isMovingRight || isMovingUp || isMovingDown)

{

animationMode = RUN;

human[RUN][direction].OnMove();

if(isMovingLeft)

if(map->IsNoEntry(FPosition(Loc.\_pX - STEP\_SIZE, Loc.\_pY)))

{

for(i = 1; !map->IsNoEntry(FPosition(Loc.\_pX - i, Loc.\_pY));

i++);

Loc.\_pX -= (--i);

}

else

Loc.\_pX -= (STEP\_SIZE);

if(isMovingRight)

if(map->IsNoEntry(FPosition(Loc.\_pX + STEP\_SIZE, Loc.\_pY)))

{

for(i = 1; !map->IsNoEntry(FPosition(Loc.\_pX + i, Loc.\_pY));

i++);

Loc.\_pX += (--i);

}

else

Loc.\_pX += (STEP\_SIZE);

if(isMovingUp)

if(map->IsNoEntry(FPosition(Loc.\_pX, Loc.\_pY - STEP\_SIZE)))

{

for(i = 1; !map->IsNoEntry(FPosition(Loc.\_pX, Loc.\_pY - i));

i++);

Loc.\_pY -= (--i);

}

else

Loc.\_pY -= (STEP\_SIZE);

if(isMovingDown)

if(map->IsNoEntry(FPosition(Loc.\_pX, Loc.\_pY + STEP\_SIZE)))

{

for(i = 1; !map->IsNoEntry(FPosition(Loc.\_pX, Loc.\_pY + i));

i++);

Loc.\_pY += (--i);

}

else

Loc.\_pY += (STEP\_SIZE);

}

else

{

animationMode = STOP;//設定模式為STOP

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

{

human[RUN][i].Reset();//重製RUN動畫

}

}

MouseTargetPositionRefresh();

}

//顯示人類

void FHuman::OnShow(int sx, int sy)

{

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

human[animationMode][i].OnMove();

human[animationMode][direction].SetTopLeft(sx - 40, sy - 64);

human[animationMode][direction].OnShow();

if(tool->OnShow(sx, sy))

OnLButtonDownHandle(mousePointTemp);

}

//鍵盤按下事件處理

void FHuman::OnKeyDownHandle(UINT nChar)

{

const char KEY\_LEFT = 0x41;

const char KEY\_UP = 0x57;

const char KEY\_RIGHT = 0x44;

const char KEY\_DOWN = 0x53;

const char KEY\_B = 0x42;

const char KEY\_I = 0x49;

const char KEY\_E = 0x45;

const char KEY\_ESC = 0x1b;

if(nowMouseMode == COLLECTION)

{

if(nChar == KEY\_LEFT)

isMovingLeft = true;

if(nChar == KEY\_RIGHT)

isMovingRight = true;

if(nChar == KEY\_UP)

isMovingUp = true;

if(nChar == KEY\_DOWN)

isMovingDown = true;

}

else

StopMotion();

}

//鍵盤起來事件處理

void FHuman::OnKeyUpHandle(UINT nChar)

{

const char KEY\_LEFT = 0x41;

const char KEY\_UP = 0x57;

const char KEY\_RIGHT = 0x44;

const char KEY\_DOWN = 0x53;

if(nowMouseMode == COLLECTION)

{

if(nChar == KEY\_LEFT)

isMovingLeft = false;

if(nChar == KEY\_RIGHT)

isMovingRight = false;

if(nChar == KEY\_UP)

isMovingUp = false;

if(nChar == KEY\_DOWN)

isMovingDown = false;

}

}

//滑鼠左鍵事件處理

void FHuman::OnLButtonDownHandle(const CPoint& point)

{

if(nowMouseMode == COLLECTION)

{

FPosition mouseLoc(Loc.\_pX + (point.x - SIZE\_X / 2) \* 100 / 80,

Loc.\_pY + (point.y - SIZE\_Y / 2) \* 100 / 80);

FPosition blockMapLoc(

mouseLoc.ConvertMouseToBlockMapPosition());

tool->SetIsCollection(true);

//當採集冷卻結束

if(tool->GetCollectionClock() == 0)

{

//冷卻重置

tool->ResetCollectionClock();

//若採集目標有在採集範圍內

if(IsInRange(blockMapLoc))

{

const FMapImage\* block = map->GetRealBlock(blockMapLoc);

//若採集目標可採集

if(block && block->isCollectible)

{

//播放音效

XAudio2Engine::instance.Play("audio\\collect.wav");

//若包包沒有滿

if(!bag.IsFull(block->resource, 1))

{

FPosition realBlockLoc = map->GetRealBlockLoc(blockMapLoc);

int blockX = realBlockLoc.\_pX;

int blockY = realBlockLoc.\_pY;

int extraX =

map->m\_mapBlock[blockY][blockX].pFrontImageInfo->buildingExtraX;

int extraY =

map->m\_mapBlock[blockY][blockX].pFrontImageInfo->buildingExtraY;

map->CollectEffectReset();

collectParticle.SetCollectParticle(extraX + 1);

//被採集的資源整個都要特效

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

for(int j = 0; j <= extraX; j++)

map->SetCollectEffect(&map->m\_mapBlock[blockY + i][blockX + j]);

//採集條減少

bar.Collect(FPosition(blockX, blockY),

int(block->collectCount \* tool->GetCollectSpeed()),

block->imgWidth,

block->imgHeight);

//若採集條為0或使用Debug模式

if(bar.IsZero(debugMode))

//包包獲得資源

bag.GetRes(map->CollectResource(blockMapLoc), 1);

}

}

}

}

}

}

//滑鼠左鍵起來事件處理

void FHuman::OnLButtonDownUp(const CPoint& point)

{

if(nowMouseMode == COLLECTION)

tool->SetIsCollection(false);

}

//滑鼠移動事件處理

void FHuman::OnMouseMoveHandle(const CPoint& point)

{

mousePointTemp = point;

//更新滑鼠座標

int mouseX = mousePointTemp.x;

int mouseY = mousePointTemp.y;

if(nowMouseMode == COLLECTION)

{

//改變人類及工具面對方向

if(mouseX >= SIZE\_X / 2)

direction = FACERIGHT;

else if(mouseX < SIZE\_X / 2)

direction = FACELEFT;

tool->SetDirection(direction);

}

}

//判斷資源是否在採集範圍內

bool FHuman::IsInRange(const FPosition& blockLoc)

{

FPosition humanMapLoc(Loc.ConvertHumanToMapPosition());

int dx = blockLoc.\_pX - int(humanMapLoc.\_pX);

int dy = blockLoc.\_pY - int(humanMapLoc.\_pY);

return (dx >= -1 && dx <= 1) && (dy >= -1 && dy <= 1);

}

//滑鼠座標更新

void FHuman::MouseTargetPositionRefresh()

{

int mouseX = mousePointTemp.x;

int mouseY = mousePointTemp.y;

if(nowMouseMode == COLLECTION)

{

FPosition mouseLoc(Loc.\_pX + (mouseX - SIZE\_X / 2) \* 100 / 80,

Loc.\_pY + (mouseY - SIZE\_Y / 2) \* 100 / 80);

FPosition blockMapLoc(

mouseLoc.ConvertMouseToBlockMapPosition());

if(IsInRange(blockMapLoc))

{

const FMapImage\* block = map->GetRealBlock(blockMapLoc);

if(block && block->isCollectible)

{

FPosition realBlockLoc = map->GetRealBlockLoc(blockMapLoc);

FPosition humanScreenLoc = Loc.ConvertHumanToScreenPosition();

FPosition blockScreenTopLeftLoc =

realBlockLoc.ConvertMapToScreenTopLeftPosition();

FPosition targetBlockScreenTopLeftLoc = FPosition(

blockScreenTopLeftLoc.\_pX -

(humanScreenLoc.\_pX - (SIZE\_X / 2)),

blockScreenTopLeftLoc.\_pY - (humanScreenLoc.\_pY -

(SIZE\_Y / 2)));

mouseTarget.SetTopLeftWidth(targetBlockScreenTopLeftLoc,

block->imgWidth,

block->imgHeight);

mouseTarget.SetPointing(true);

//若滑鼠目標可互動

if(block->isInteractable)

{

FProductBuildSystem::interactiveE.SetTopLeft(

targetBlockScreenTopLeftLoc.\_pX +

block->imgWidth / 2 - 21,

targetBlockScreenTopLeftLoc.\_pY + block->imgHeight / 2 - 19);

FProductBuildSystem::nowInteractive =

FProductBuildSystem::ConverToPB(

block->type);

FProductBuildSystem::nowInteractiveMapLoc = FPosition(

realBlockLoc.\_pX,

realBlockLoc.\_pY);

isInteractable = true;

}

else

isInteractable = false;

return;

}

}

}

isInteractable = false;

mouseTarget.SetPointing(false);

}

//停止動作

void FHuman::StopMotion()

{

isMovingLeft = false;

isMovingRight = false;

isMovingUp = false;

isMovingDown = false;

tool->SetIsCollection(false);

animationMode = STOP;

mouseTarget.SetPointing(false);

isInteractable = false;

}

//變換工具

void FHuman::ChangeTool()

{

delete tool;

if(nowTool == FTool::PICKAXE)

tool = new FTool\_Pickaxe();

else if(nowTool == FTool::PICKAXE2)

tool = new FTool\_Pickaxe2();

else if(nowTool == FTool::PICKAXE3)

tool = new FTool\_Pickaxe3();

tool->SetDirection(direction);

}

//將地圖座標轉換為螢幕座標

FPosition FHuman::BlockMapLocToScreenTopLeftLoc(

const FPosition& blockMapLoc)

{

FPosition blockLoc = FPosition(blockMapLoc.\_pX,

blockMapLoc.\_pY).ConvertMapToScreenTopLeftPosition();

FPosition humanScreenLoc = Loc.ConvertHumanToScreenPosition();

return FPosition(blockLoc.\_pX - (humanScreenLoc.\_pX + correctX),

blockLoc.\_pY - (humanScreenLoc.\_pY + correctY));

}

//destructor 刪除工具

FHuman::~FHuman()

{

delete tool;

}

//顯示滑鼠目標

void FHuman::MouseTargetShow()

{

mouseTarget.OnShow();

}

//顯示採集條

void FHuman::ShowBar()

{

FPosition blockLoc = FPosition(bar.blockMapLoc.\_pX,

bar.blockMapLoc.\_pY).ConvertMapToScreenTopLeftPosition();

FPosition humanScreenLoc = Loc.ConvertHumanToScreenPosition();

int sx = blockLoc.\_pX - (humanScreenLoc.\_pX + correctX);

int sy = blockLoc.\_pY - (humanScreenLoc.\_pY + correctY);

collectParticle.OnShow(sx, sy);

bar.OnShow(sx, sy);

}

//改變滑鼠模式(多載)

void FHuman::SetMouseMode(bool isUsingMenu)

{

if(isUsingMenu)

this->nowMouseMode = MENU;

else

this->nowMouseMode = COLLECTION;

}

//改變滑鼠模式(多載)

void FHuman::SetMouseMode(int mouseMode)

{

this->nowMouseMode = mouseMode;

}

//設定Debug模式

void FHuman::SetDebugMode(bool mode)

{

this->debugMode = mode;

}

}

/\* FileName: FHuman.h \*/

﻿#ifndef FORAGER\_FHUMAN\_H

#define FORAGER\_FHUMAN\_H

#include "FPosition.h"

#include "FBag.h"

#include "FHealth.h"

#include "FCollectionBar.h"

#include "FMouseTarget.h"

#include "FCollectParticle.h"

#include "FProductBuildSystem.h"

#include "FAnimation.h"

#include "FTool.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FMap;

class FHuman

{

public:

enum MouseMode

{

COLLECTION = 0, //採集

MENU,

INTERACTIVE,

MOUSEMODEEND

};

enum DirectionMode

{

FACERIGHT,

FACELEFT,

FACEEND

};

enum AnimationMode

{

STOP,

RUN,

ANIMATIONMODEEND

};

FHuman() = default;

void Initialize(FMap\* map);

void OnMove();

void OnShow(int sx, int sy);

void OnKeyDownHandle(UINT nChar);

void OnKeyUpHandle(UINT nChar);

void OnLButtonDownHandle(const CPoint& point);

void OnLButtonDownUp(const CPoint& point);

void OnMouseMoveHandle(const CPoint& point);

void MouseTargetShow();

void ShowBar();

void SetMouseMode(bool isUsingMenu);

void SetMouseMode(int mouseMode);

void SetDebugMode(bool mode);

bool isInteractable;

FPosition BlockMapLocToScreenTopLeftLoc(const FPosition&

blockLoc);

int nowMouseMode;

static FPosition Loc;

FBag bag;

FCollectionBar bar;

FHealth humanHealth;

FCollectParticle collectParticle;

int nowTool;

void ChangeTool();

~FHuman();

private:

int collectionRange;//採集範圍

int direction;//面對方向

int animationMode;

int nowInteractiveBuild;

int correctX = -SIZE\_X / 2, correctY = -SIZE\_Y / 2;

bool isMovingDown;

bool isMovingLeft;

bool isMovingRight;

bool isMovingUp;

bool isPointing;

bool debugMode;

FTool\* tool;

FAnimation human[ANIMATIONMODEEND][FACEEND];

FMouseTarget mouseTarget;

FMap\* map;

CPoint mousePointTemp; //暫存滑鼠座標

bool IsInRange(const FPosition&

blockMapLoc);//是否在採集範圍

void MouseTargetPositionRefresh();

void StopMotion();

};

}

#endif // !FORAGER\_FHUMAN\_H

/\* FileName: FImage.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FImage.h"

using namespace std;

namespace game\_framework

{

// 讀取圖片

void FImage::LoadBitmap(int resID, COLORREF RGBMask)

{

this->bmpID = D2D1Draw::RegisterBitmap(resID, RGBMask);

}

// 取得 D2D1 圖片資源, 用於某些需要直接存取的時機

ID2D1Bitmap\* FImage::GetBmpRes()

{

return D2D1Draw::GetBitmapRecord(bmpID).pBitmap;

}

// 顯示圖片

void FImage::ShowBitmap()

{

if(useDestSize)

D2D1Draw::DrawBitmap(bmpID, xDst, yDst, FLOAT(alpha / 255.0),

wDst, hDst, NULL,

scale, scale, degree);

else

D2D1Draw::DrawBitmap(bmpID, xDst, yDst, FLOAT(alpha / 255.0));

}

// 設定圖片位置

void FImage::SetTopLeft(int xDst, int yDst, int wDst, int hDst)

{

this->xDst = xDst;

this->yDst = yDst;

this->wDst = wDst;

this->hDst = hDst;

useDestSize = !((wDst == -1) && (wDst == hDst));

}

// 設定圖片透明度

void FImage::SetAlpha(int alpha)

{

if(alpha > 255) alpha = 255;

if(alpha < 0) alpha = 0;

this->alpha = alpha;

}

// 設定圖片縮放比例

void FImage::SetScale(FLOAT scale)

{

this->scale = scale;

useScale = (scale != 1) ? true : false;

}

// 設定圖片旋轉角度

void FImage::SetRotation(FLOAT degree)

{

this->degree = degree;

useRotation = (degree != 0) ? true : false;

}

}

/\* FileName: FImage.h \*/

﻿#pragma once

#ifndef FORAGER\_FIMAGE\_H

#define FORAGER\_FIMAGE\_H

#include "CGameStdLib.h"

using namespace std;

namespace game\_framework

{

class FImage

{

public:

FImage()=default;

void LoadBitmap(int resID, COLORREF RGBMask = CLR\_INVALID);

void ShowBitmap();

void SetTopLeft(int xDst, int yDst, int wDst = -1,

int hDst = -1);

void SetAlpha(int alpha = 255);

void SetScale(FLOAT scale = 1);

void SetRotation(FLOAT degree = 0);

ID2D1Bitmap\* GetBmpRes();

private:

int bmpID = -1, alpha = 255, xDst = 0, yDst = 0, wDst = 0,

hDst = 0;

FLOAT scale = 1, degree = 0;

bool useDestSize = false, useScale = false, useRotation = false;

};

}

#endif // !FORAGER\_FIMAGE\_H

/\* FileName: FMainMenu.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FMainMenu.h"

using namespace std;

namespace game\_framework

{

//destructor 解除註冊 Callback Function

FMainMenu::~FMainMenu()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

D2D1\_SAFERELEASE(helpImage);

}

//constructor 註冊 Callback Function

FMainMenu::FMainMenu()

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FMainMenu::D2D1CallBack,

this));

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FMainMenu::D2D1CallBack()

{

D2D1\_SAFERELEASE(helpImage);

helpImage = D2D1Draw::LoadResourcePNG(IDB\_EXPLANPNG);

}

//讀取圖片及設定座標

void FMainMenu::Initialize()

{

FSaveSystem::LoadAllSaveBitmap();

logo.LoadBitmap(IDB\_GAMELOGO, RGB(51, 153, 253));

background.LoadBitmap(IDB\_MAINMENUBK);

backgroundMask.LoadBitmap(IDB\_MAINBKMASK);

exitBtn.LoadBitmap(IDB\_EXITBTN, RGB(56, 152, 255));

exitBtnPoint = FPosition(SIZE\_X - 530, SIZE\_Y - 175);

exitBtn.SetTopLeft(exitBtnPoint.\_pX, exitBtnPoint.\_pY, 139, 72);

backgroundBlock.LoadBitmap(IDB\_MAINMENUBK, 1, 1, 40, 40);

helpBtn.LoadBitmap(IDB\_HELPBTN, RGB(56, 152, 255));

helpMask.LoadBitmap(IDB\_BAGBACKGROUND);

helpMask.SetTopLeft(0, 0, 1280, 720);

helpMask.SetAlpha(235);

helpBtnPoint = FPosition(SIZE\_X - 895, SIZE\_Y - 175);

helpBtn.SetTopLeft(helpBtnPoint.\_pX, helpBtnPoint.\_pY, 139, 68);

for(int i = 0; i < 3; i++)

{

saveSelectButtonPoint[i] = FPosition(SIZE\_X / 2 - 145 - 371 +

(371 \* i),

SIZE\_Y / 2 - 150);

saveSelectButton[i].Initialize(&FSaveSystem::saveImage[i]);

}

this->mouseTarget.Initialize();

rButtonDownCount = 0;

D2D1CallBack();

}

//顯示主選單背景

void FMainMenu::ShowBackground()

{

//格子移動速度

if(offset >= 80)

offset = 0;

offset += 2;

//獲得格子數量

int mapx = (SIZE\_X / FMap::BLOCK\_SIZE);

int mapy = (SIZE\_Y / FMap::BLOCK\_SIZE);

//主畫面中間的背景

for(int x = 0; x < mapx; x++)

for(int y = 0; y < mapy; y++)

{

FPosition blockLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

background.SetTopLeft(blockLoc.\_pX, blockLoc.\_pY);

background.ShowBitmap();

}

//主畫面上面移動的格子

for(int x = 0; x < mapx + 2; x++)

{

FPosition blockLoc = FPosition(x,

0).ConvertMapToScreenTopLeftPosition();

backgroundMask.SetTopLeft(blockLoc.\_pX - offset,

blockLoc.\_pY - 70);

backgroundMask.ShowBitmap();

backgroundBlock.SetTopLeft(blockLoc.\_pX - offset, blockLoc.\_pY);

backgroundBlock.ShowPartBitmap(0);

backgroundBlock.SetTopLeft(blockLoc.\_pX + 40 - offset,

blockLoc.\_pY + 40);

backgroundBlock.ShowPartBitmap(0);

}

//主畫面下面移動的格子

for(int x = 0; x < mapx + 2; x++)

{

FPosition blockLoc = FPosition(x,

mapy - 1).ConvertMapToScreenTopLeftPosition();

backgroundMask.SetTopLeft(blockLoc.\_pX - offset,

blockLoc.\_pY - 10);

backgroundMask.ShowBitmap();

backgroundBlock.SetTopLeft(blockLoc.\_pX - offset, blockLoc.\_pY);

backgroundBlock.ShowPartBitmap(0);

backgroundBlock.SetTopLeft(blockLoc.\_pX + 40 - offset,

blockLoc.\_pY + 40);

backgroundBlock.ShowPartBitmap(0);

}

//顯示LOGO

logo.SetTopLeft(SIZE\_X / 2 - 142, 30);

logo.ShowBitmap();

}

//顯示存檔按鈕

void FMainMenu::ShowSaveSelectButton()

{

for(int i = 0; i < 3; i++)

{

saveSelectButton[i].SetTopLeft(saveSelectButtonPoint[i].\_pX,

saveSelectButtonPoint[i].\_pY);

if(currentHover == i + 1)

saveSelectButton[i].SetUseEffect(true, !rButtonDownCountLock);

else

saveSelectButton[i].SetUseEffect(false);

saveSelectButton[i].OnShow();

}

}

//回傳point位置按鈕索引

int FMainMenu::GetHoverObjectType(const CPoint& point)

{

for(int i = 0; i < 3; i++)

{

if(point.x >= saveSelectButtonPoint[i].\_pX &&

point.x <= saveSelectButtonPoint[i].\_pX + 291

&& point.y >= saveSelectButtonPoint[i].\_pY &&

point.y <= saveSelectButtonPoint[i].\_pY + 291)

return i + 1;

}

if(point.x >= exitBtnPoint.\_pX &&

point.x <= exitBtnPoint.\_pX + 139

&& point.y >= exitBtnPoint.\_pY &&

point.y <= exitBtnPoint.\_pY + 72)

return EXIT\_BUTTON;

if(point.x >= helpBtnPoint.\_pX &&

point.x <= helpBtnPoint.\_pX + 139

&& point.y >= helpBtnPoint.\_pY &&

point.y <= helpBtnPoint.\_pY + 68)

return HELP\_BUTTON;

return NOTHING;

}

//滑鼠移動事件處理

void FMainMenu::MouseMoveHandle(const CPoint& point)

{

exitBtn.SetScale(1.0f);

helpBtn.SetScale(1.0f);

if(helpMode)

{

mouseTarget.SetPointing(false);

return;

}

HoverObjectType mouseObject = (HoverObjectType)

GetHoverObjectType(point);

this->currentHover = mouseObject;

if(mouseObject != HoverObjectType::NOTHING)

{

if(mouseObject == EXIT\_BUTTON)

{

mouseTarget.SetTopLeftWidth(exitBtnPoint, 209, 142, -35, -35);

exitBtn.SetScale(1.1f);

}

else if(mouseObject == HELP\_BUTTON)

{

mouseTarget.SetTopLeftWidth(helpBtnPoint, 209, 142, -35, -35);

helpBtn.SetScale(1.1f);

}

else

mouseTarget.SetTopLeftWidth(saveSelectButtonPoint[mouseObject -

1], 361, 361,

-35, -35);

mouseTarget.SetPointing(true);

}

else

mouseTarget.SetPointing(false);

}

//顯示主選單

void FMainMenu::OnShow()

{

this->ShowBackground();

this->ShowSaveSelectButton();

this->mouseTarget.OnShow();

if(!rButtonDownCountLock && currentHover > 0 &&

currentHover < 4)

{

rButtonDownCount++;

if(rButtonDownCount > (30 \* 2))

{

if(FSaveSystem::IsExistSaveFile(currentHover - 1))

{

FSaveSystem::DeleteSaveFile(currentHover - 1);

Initialize();

}

}

}

this->helpBtn.ShowBitmap();

this->exitBtn.ShowBitmap();

if(helpMode)

{

this->helpMask.ShowBitmap();

D2D1Draw::DrawBitmap(helpImage, 0, 0);

}

}

//滑鼠左鍵事件處理

bool FMainMenu::OnLButtonDownHandle(const CPoint& point)

{

if(helpMode)

{

helpMode = false;

return false;

}

HoverObjectType mouseObject = (HoverObjectType)

GetHoverObjectType(point);

if(mouseObject == SAVE\_BUTTON\_1)

FSaveSystem::saveID = 0;

else if(mouseObject == SAVE\_BUTTON\_2)

FSaveSystem::saveID = 1;

else if(mouseObject == SAVE\_BUTTON\_3)

FSaveSystem::saveID = 2;

else if(mouseObject == EXIT\_BUTTON)

AfxGetMainWnd()->PostMessage(WM\_CLOSE);

else if(mouseObject == HELP\_BUTTON)

helpMode = true;

return mouseObject == SAVE\_BUTTON\_1 ||

mouseObject == SAVE\_BUTTON\_2 ||

mouseObject == SAVE\_BUTTON\_3;

}

//滑鼠右鍵事件處理

void FMainMenu::OnRButtonDownHandle(const CPoint& point)

{

rButtonDownCount = 0;

rButtonDownCountLock = false;

helpMode = false;

}

//滑鼠右鍵起來事件

void FMainMenu::OnRButtonUpHandle(const CPoint& point)

{

rButtonDownCount = 0;

rButtonDownCountLock = true;

helpMode = false;

}

}

/\* FileName: FMainMenu.h \*/

﻿#pragma once

#ifndef FORAGER\_FMAINMENU\_H

#define FORAGER\_FMAINMENU\_H

#include "CGameStdLib.h"

#include "FSaveButton.h"

#include "FSaveSystem.h"

#include "FImage.h"

#include "FAtlas.h"

#include "FMap.h"

using namespace std;

namespace game\_framework

{

class FMainMenu

{

public:

enum HoverObjectType

{

NOTHING,

SAVE\_BUTTON\_1,

SAVE\_BUTTON\_2,

SAVE\_BUTTON\_3,

HELP\_BUTTON,

EXIT\_BUTTON

};

FMainMenu();

~FMainMenu();

void Initialize();

void OnShow();

void ShowBackground();

void ShowSaveSelectButton();

void D2D1CallBack();

void MouseMoveHandle(const CPoint& point);

bool OnLButtonDownHandle(const CPoint& point);

void OnRButtonDownHandle(const CPoint& point);

void OnRButtonUpHandle(const CPoint& point);

int GetHoverObjectType(const CPoint& point);

private:

ID2D1Bitmap\* helpImage = NULL;

FImage logo;

FImage exitBtn;

FImage helpBtn;

FImage helpMask;

FImage background;

FImage backgroundMask;

FAtlas backgroundBlock;

FSaveButton saveSelectButton[3];

FPosition saveSelectButtonPoint[3];

FPosition exitBtnPoint;

FPosition helpBtnPoint;

FMouseTarget mouseTarget;

HoverObjectType currentHover = HoverObjectType::NOTHING;

int rButtonDownCount = 0;

bool rButtonDownCountLock = true;

bool helpMode = false;

int offset = 0;

int callBackID = 0;

};

}

#endif // !FORAGER\_FMAINMENU\_H

/\* FileName: FMap.cpp \*/

﻿#include "stdafx.h"

#include "FMap.h"

#include "FBuildSystem.h"

#include <random>

using namespace std;

namespace game\_framework

{

//讀取圖片及設定座標及初始化數值

void FMap::Initialize()

{

autoCreateResourceDelay = 0;

for(int i = 0; i < FMapInfo::TYPE\_END; i++)

FMapInfo::imageInfo[i].LoadBitmap();

FMapImage::weedImage.LoadBitmap(IDB\_WEED, 2, 5, 80, 80, RGB(195,

195, 195));

FGrassSeaEffect::Initialize();

// 設定資源大小

resPercemt[FMapInfo::SEA] = 10;

resPercemt[FMapInfo::GRASS] = 0;

resPercemt[FMapInfo::TREE] = 10;

resPercemt[FMapInfo::BERRY] = 5;

resPercemt[FMapInfo::ROCK] = 5;

resPercemt[FMapInfo::COAL] = 5;

resPercemt[FMapInfo::FURNACE\_1] = 0;

resPercemt[FMapInfo::FURNACE\_2] = 0;

resPercemt[FMapInfo::FURNACE\_3] = 0;

resPercemt[FMapInfo::FURNACE\_4] = 0;

resPercemt[FMapInfo::FORGE\_1] = 0;

resPercemt[FMapInfo::FORGE\_2] = 0;

resPercemt[FMapInfo::FORGE\_3] = 0;

resPercemt[FMapInfo::FORGE\_4] = 0;

resPercemt[FMapInfo::TORCH] = 0;

// 初始化亂數產生器

random\_device rd;

gen = mt19937(rd());

// 地圖生成

MapFactory();

}

//隨機產生湖泊

int FMap::SeaFactory(int x, int y)

{

if(m\_mapBlock[y][x].pFrontImageInfo != nullptr ||

m\_mapBlock[y][x].pBackImageInfo->isCollision)

return 0;

uniform\_int\_distribution<int> seaMax(4, 10);

uniform\_int\_distribution<int> ignoreMax(0, 7);

int size = seaMax(gen);

int count = 0;

vector<FPosition> seaQueue;

seaQueue.push\_back(FPosition(x, y));

while(count < size && !seaQueue.empty())

{

FPosition seaPoint = seaQueue[0];

seaQueue.erase(seaQueue.begin());

if(m\_mapBlock[seaPoint.\_pY][seaPoint.\_pX].pFrontImageInfo ==

nullptr &&

!m\_mapBlock[seaPoint.\_pY][seaPoint.\_pX].pBackImageInfo->isCollision

&& seaPoint.\_pY >= 0 && seaPoint.\_pY < MAPY\_SIZE &&

seaPoint.\_pX >= 0 &&

seaPoint.\_pX < MAPX\_SIZE)

{

m\_mapBlock[seaPoint.\_pY][seaPoint.\_pX] = FMapInfo{ FMapInfo::SEA, FMapInfo::NOTHING };

count++;

}

else

break;

vector<int> ignoreList;

while(ignoreList.size() < 4)

{

int ignore = ignoreMax(gen);

if(find(ignoreList.begin(), ignoreList.end(),

ignore) == ignoreList.end())

ignoreList.push\_back(ignore);

}

if(find(ignoreList.begin(), ignoreList.end(),

0) != ignoreList.end())

seaQueue.push\_back(FPosition(x + 1, y));

if(find(ignoreList.begin(), ignoreList.end(),

1) != ignoreList.end())

seaQueue.push\_back(FPosition(x - 1, y));

if(find(ignoreList.begin(), ignoreList.end(),

2) != ignoreList.end())

seaQueue.push\_back(FPosition(x, y + 1));

if(find(ignoreList.begin(), ignoreList.end(),

3) != ignoreList.end())

seaQueue.push\_back(FPosition(x, y - 1));

if(find(ignoreList.begin(), ignoreList.end(),

4) != ignoreList.end())

seaQueue.push\_back(FPosition(x + 1, y - 1));

if(find(ignoreList.begin(), ignoreList.end(),

5) != ignoreList.end())

seaQueue.push\_back(FPosition(x - 1, y + 1));

if(find(ignoreList.begin(), ignoreList.end(),

6) != ignoreList.end())

seaQueue.push\_back(FPosition(x + 1, y + 1));

if(find(ignoreList.begin(), ignoreList.end(),

7) != ignoreList.end())

seaQueue.push\_back(FPosition(x + 1, y - 1));

}

return count;

}

//隨機產生地形及資源

void FMap::MapFactory()

{

// 產生基本地形

for(int y = 0; y < MAPX\_SIZE; y++)

for(int x = 0; x < MAPY\_SIZE; x++)

if(y >= PROTECT\_Y && y <= MAPY\_SIZE - PROTECT\_Y &&

x >= PROTECT\_X &&

x <= MAPX\_SIZE - PROTECT\_X)

m\_mapBlock[y][x] = FMapInfo{ FMapInfo::GRASS, FMapInfo::NOTHING };

else

m\_mapBlock[y][x] = FMapInfo{ FMapInfo::SEA, FMapInfo::NOTHING };

// 湖生成

int count = int(MAPY\_SIZE \* MAPX\_SIZE \*

(resPercemt[FMapInfo::SEA] / 100.0));

while(count > 0)

{

uniform\_int\_distribution<int> disY(0, MAPY\_SIZE - 1);

uniform\_int\_distribution<int> disX(0, MAPX\_SIZE - 1);

int x = disY(gen), y = disX(gen);

count -= SeaFactory(x, y);

}

// 出生點保護

for(int y = MAPY\_SIZE / 2 - 3; y < MAPY\_SIZE / 2 + 3; y++)

for(int x = MAPX\_SIZE / 2 - 3; x < MAPX\_SIZE / 2 + 3; x++)

m\_mapBlock[y][x] = FMapInfo{ FMapInfo::GRASS, FMapInfo::NOTHING };

// 按照資源比例產生資源

for(int i = 0; i < FMapInfo::TYPE\_END; i++)

{

if(i == FMapInfo::SEA)

continue;

int count = int(MAPY\_SIZE \* MAPX\_SIZE \* (resPercemt[i] /

100.0));

while(count != 0)

{

uniform\_int\_distribution<int> disY(0, MAPY\_SIZE - 1);

uniform\_int\_distribution<int> disX(0, MAPX\_SIZE - 1);

int x = disY(gen), y = disX(gen);

if(m\_mapBlock[y][x].pFrontImageInfo == nullptr &&

!m\_mapBlock[y][x].pBackImageInfo->isCollision)

{

m\_mapBlock[y][x].pFrontImageInfo = &FMapInfo::imageInfo[i];

count--;

}

}

}

// 出生點保護

for(int y = MAPY\_SIZE / 2 - 1; y < MAPY\_SIZE / 2 + 1; y++)

for(int x = MAPX\_SIZE / 2 - 1; x < MAPX\_SIZE / 2 + 1; x++)

m\_mapBlock[y][x] = FMapInfo{ FMapInfo::GRASS, FMapInfo::NOTHING };

// 產生雜草

for(int y = 0; y < MAPY\_SIZE; y++)

for(int x = 0; x < MAPX\_SIZE; x++)

if(m\_mapBlock[y][x].pFrontImageInfo == nullptr &&

m\_mapBlock[y][x].pBackImageInfo->type == FMapInfo::GRASS)

{

uniform\_int\_distribution<int> probabilityMax(0, 10);

if(probabilityMax(gen) > 7)

{

uniform\_int\_distribution<int> weedTypeMax(0,

FMapImage::WeedType::WEEDTYPE\_END - 1);

m\_mapBlock[y][x].weedType = (FMapImage::WeedType)weedTypeMax(

gen);

}

}

// 設定海效果

for(int y = 1; y < MAPY\_SIZE; y++)

for(int x = 0; x < MAPX\_SIZE; x++)

if(m\_mapBlock[y][x].pBackImageInfo->type == FMapInfo::SEA &&

m\_mapBlock[y - 1][x].pBackImageInfo->type == FMapInfo::GRASS)

m\_mapBlock[y][x].useSeaEffect = true;

for(int y = 0; y < MAPY\_SIZE; y++)

for(int x = 0; x < MAPX\_SIZE; x++)

if(m\_mapBlock[y][x].pBackImageInfo->type == FMapInfo::SEA)

{

if(CheckMapXY(x, y - 1) &&

m\_mapBlock[y - 1][x].pBackImageInfo->type == FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_U;

if(CheckMapXY(x, y + 1) &&

m\_mapBlock[y + 1][x].pBackImageInfo->type == FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_D;

if(CheckMapXY(x - 1, y) &&

m\_mapBlock[y][x - 1].pBackImageInfo->type == FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_L;

if(CheckMapXY(x + 1, y) &&

m\_mapBlock[y][x + 1].pBackImageInfo->type == FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_R;

if(m\_mapBlock[y][x].edgeFlag & SEAEDGE\_L &&

m\_mapBlock[y][x].edgeFlag & SEAEDGE\_U &&

CheckMapXY(x - 1, y - 1) &&

m\_mapBlock[y - 1][x - 1].pBackImageInfo->type ==

FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_UL;

if(m\_mapBlock[y][x].edgeFlag & SEAEDGE\_R &&

m\_mapBlock[y][x].edgeFlag & SEAEDGE\_U &&

CheckMapXY(x + 1, y - 1) &&

m\_mapBlock[y - 1][x + 1].pBackImageInfo->type ==

FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_UR;

if(m\_mapBlock[y][x].edgeFlag & SEAEDGE\_L &&

m\_mapBlock[y][x].edgeFlag & SEAEDGE\_D &&

CheckMapXY(x - 1, y + 1) &&

m\_mapBlock[y + 1][x - 1].pBackImageInfo->type ==

FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_DL;

if(m\_mapBlock[y][x].edgeFlag & SEAEDGE\_R &&

m\_mapBlock[y][x].edgeFlag & SEAEDGE\_D &&

CheckMapXY(x + 1, y + 1) &&

m\_mapBlock[y + 1][x + 1].pBackImageInfo->type ==

FMapInfo::GRASS)

m\_mapBlock[y][x].edgeFlag |= SEAEDGE\_DR;

}

}

//判斷x, y是否超出地圖範圍

bool FMap::CheckMapXY(int x, int y)

{

return x >= 0 && x < MAPX\_SIZE && y >= 0 && y < MAPY\_SIZE;

}

//回傳地圖格子reference

FMapInfo\* FMap::GetDangerMapInfo(int x, int y)

{

return &m\_mapBlock[y][x];

}

//判斷目標位置是否有碰撞物

bool FMap::IsNoEntry(const FPosition& newHumanLoc)

{

FPosition humanScreenLoc =

newHumanLoc.ConvertHumanToScreenPosition();

FPosition humanMapLoc = newHumanLoc.ConvertHumanToMapPosition();

for(int y = humanMapLoc.\_pY - 1; y <= humanMapLoc.\_pY + 1; y++)

for(int x = humanMapLoc.\_pX - 1; x <= humanMapLoc.\_pX + 1; x++)

{

FPosition blockScreenLoc = FPosition(x,

y).ConvertMapToScreenTopLeftPosition();

FMapImage\* frontBlockData = m\_mapBlock[y][x].pFrontImageInfo;

FMapImage\* backBlockData = m\_mapBlock[y][x].pBackImageInfo;

//目標位置有碰撞物

if(backBlockData && backBlockData->isCollision)

{

humanRect.SetTopLeft(FPosition(humanScreenLoc.\_pX - 14,

humanScreenLoc.\_pY - 17));

backBlockData->SetRectTopLeft(blockScreenLoc.\_pX,

blockScreenLoc.\_pY);

//若移動到位置會發生碰撞

if(humanRect.IsCollision(backBlockData->m\_rect))

return true;

}

//目標位置有碰撞物

if(frontBlockData && frontBlockData->isCollision)

{

humanRect.SetTopLeft(FPosition(humanScreenLoc.\_pX - 14,

humanScreenLoc.\_pY - 17));

frontBlockData->SetRectTopLeft(blockScreenLoc.\_pX,

blockScreenLoc.\_pY);

//若移動到位置會發生碰撞

if(humanRect.IsCollision(frontBlockData->m\_rect))

return true;

}

}

return false;

}

//回傳目標地圖位置的前景

const FMapImage\* FMap::GetBlock(const FPosition& blockLoc)

{

return m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX].pFrontImageInfo;

}

//回傳目標地圖位置的前景的左上角指標(如:佔面積4個的建築, 會回傳左上角的指標, 因資料都放在左上角)

const FMapImage\* FMap::GetRealBlock(const FPosition& blockLoc)

{

if(m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX].pFrontImageInfo ==

nullptr)

return nullptr;

FMapInfo nowBlock = m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX];

int blockX = blockLoc.\_pX +

nowBlock.pFrontImageInfo->realMapLocDX;

int blockY = blockLoc.\_pY +

nowBlock.pFrontImageInfo->realMapLocDY;

return m\_mapBlock[blockY][blockX].pFrontImageInfo;

}

//直接取得地圖前景指標

FMapImage\* FMap::GetDangerBlock(const FPosition& blockLoc)

{

return m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX].pFrontImageInfo;

}

//計算採集特效Delay, 若時間到則重置特效

void FMap::AutoCollectEffectReset()

{

this->autoCollectEffectResetDelay++;

if(this->autoCollectEffectResetDelay < 10)

return;

else

this->autoCollectEffectResetDelay = 0;

this->CollectEffectReset();

}

//重置採集特效

void FMap::CollectEffectReset()

{

//若沒有等待中的特效

if(this->effectBlock.empty())

return;

for(UINT i = 0; i < effectBlock.size(); i++)

{

this->effectBlock[i]->useEffect = false;

if(this->effectBlock[i]->pFrontImageInfo)

this->effectBlock[i]->pFrontImageInfo->m\_effect.ResetScale();

}

}

//回傳目標地圖位置的前景的左上角指標位置

const FPosition FMap::GetRealBlockLoc(const FPosition& blockLoc)

{

if(m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX].pFrontImageInfo ==

nullptr)

return blockLoc;

FMapInfo nowBlock = m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX];

int blockX = blockLoc.\_pX +

nowBlock.pFrontImageInfo->realMapLocDX;

int blockY = blockLoc.\_pY +

nowBlock.pFrontImageInfo->realMapLocDY;

return FPosition(blockX, blockY);

}

//將被採集的物品加入特效等待區並重置Delay

void FMap::SetCollectEffect(FMapInfo\* block)

{

this->autoCollectEffectResetDelay = 0;

block->pFrontImageInfo->m\_effect.ResetScale();

block->useEffect = true;

this->effectBlock.push\_back(block);

}

//自動產生資源(當有資源被採集, 則在地圖上隨機產生同一種資源)

void FMap::AutoCreateResource()

{

autoCreateResourceDelay++;

if(autoCreateResourceDelay < 30 || autoCreateResList.empty())

return;

else

autoCreateResourceDelay = 0;

FMapInfo::Type resType = autoCreateResList[0];

//在地圖位置上產生亂數

uniform\_int\_distribution<int> disY(0, MAPY\_SIZE - 1);

uniform\_int\_distribution<int> disX(0, MAPX\_SIZE - 1);

//獲得人類地圖位置

FPosition humanMapLoc = FHuman::Loc.ConvertHumanToMapPosition();

//獲得螢幕邊界座標

int mapx\_left = humanMapLoc.\_pX - (FMap::PROTECT\_X / 2 + 2),

mapx\_right = humanMapLoc.\_pX + (FMap::PROTECT\_X / 2 + 2);

int mapy\_up = humanMapLoc.\_pY - (FMap::PROTECT\_Y / 2 + 2),

mapy\_down = humanMapLoc.\_pY + (FMap::PROTECT\_Y / 2 + 2);

int times = 0;

while(times < 10)

{

int x = disY(gen), y = disX(gen);

//當選擇的地圖位置上面沒有資源且不在螢幕範圍內

if(m\_mapBlock[y][x].pFrontImageInfo == nullptr &&

(x < mapx\_left ||

x > mapx\_right) && (y < mapy\_up || y > mapy\_down))

{

m\_mapBlock[y][x].pFrontImageInfo =

&FMapInfo::imageInfo[resType];

autoCreateResList.erase(autoCreateResList.begin());

break;

}

times++;

}

}

//當人類採集時, 把地圖上的資源消去

FResource\* FMap::CollectResource(const FPosition& blockLoc)

{

int blockX = blockLoc.\_pX +

m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX].pFrontImageInfo->realMapLocDX;

int blockY = blockLoc.\_pY +

m\_mapBlock[blockLoc.\_pY][blockLoc.\_pX].pFrontImageInfo->realMapLocDY;

const FMapImage\* block =

m\_mapBlock[blockY][blockX].pFrontImageInfo;

//若資源式可採集的

if(block->isCreatable)

//把資源放入重生列表中

autoCreateResList.push\_back((FMapInfo::Type)block->type);

//若物品為多格, 則將全部消去

for(int i = 0; i <= block->buildingExtraY; i++)

for(int j = 0; j <= block->buildingExtraX; j++)

m\_mapBlock[blockY+i][blockX+j].pFrontImageInfo = nullptr;

//隨機產生草

uniform\_int\_distribution<int> probabilityMax(0, 10);

if(probabilityMax(gen) > 7)

{

for(int i = 0; i <= block->buildingExtraY; i++)

for(int j = 0; j <= block->buildingExtraX; j++)

{

uniform\_int\_distribution<int> weedTypeMax(0,

FMapImage::WeedType::WEEDTYPE\_END - 1);

m\_mapBlock[blockY+i][blockX+j].weedType = (FMapImage::WeedType)

weedTypeMax(gen);

}

}

return FResource::NewResource(block->resource);

}

//計算海浪特效Delay, 若時間到則重置特效

void FMap::AutoSetSeaEffect()

{

static int delay = 0;

delay++;

if(delay < 3)

return;

else

delay = 0;

FGrassSeaEffect::NextEffect();

}

}

/\* FileName: FMap.h \*/

﻿#ifndef FORAGER\_FMAP\_H

#define FORAGER\_FMAP\_H

#include "CGameStdLib.h"

#include "FRect.h"

#include "FPosition.h"

#include "FMapImage.h"

#include "FResource.h"

#include "FMapInfo.h"

#include "FHuman.h"

#include <random>

#include <vector>

// Sea Edge type define

#define SEAEDGE\_U 1 << 0

#define SEAEDGE\_D 1 << 1

#define SEAEDGE\_L 1 << 2

#define SEAEDGE\_R 1 << 3

#define SEAEDGE\_UR 1 << 4

#define SEAEDGE\_DR 1 << 5

#define SEAEDGE\_UL 1 << 6

#define SEAEDGE\_DL 1 << 7

using namespace std;

namespace game\_framework

{

class FHuman;

class FBuildSystem;

class FMap

{

public:

FMap() = default;

void Initialize();

bool IsNoEntry(const FPosition& newHumanLoc);

const FMapImage\* GetBlock(const FPosition& blockLoc);

const FMapImage\* GetRealBlock(const FPosition& blockLoc);

const FPosition GetRealBlockLoc(const FPosition& blockLoc);

static const int BLOCK\_SIZE = 80; //每格像素長寬

static const int MAPX\_SIZE = 100;

static const int MAPY\_SIZE = 100;

static const int LAYOUT\_MAX = 4;

static const int PROTECT\_X = SIZE\_X /

BLOCK\_SIZE; // 邊界保護格數

static const int PROTECT\_Y = SIZE\_Y /

BLOCK\_SIZE; // 邊界保護格數

FMapInfo m\_mapBlock[MAPY\_SIZE][MAPX\_SIZE]; //地圖資料

FMapInfo\* GetDangerMapInfo(int x, int y);

bool CheckMapXY(int x, int y);

void AutoCreateResource();

void AutoCollectEffectReset();

void SetCollectEffect(FMapInfo\* block);

void CollectEffectReset();

void AutoSetSeaEffect();

FMapImage\* GetDangerBlock(const FPosition& blockLoc);

FResource\* CollectResource(const FPosition& blockLoc);

private:

void MapFactory();

int SeaFactory(int x, int y);

int resPercemt[FMapInfo::TYPE\_END];

int autoCreateResourceDelay = 0,

autoCollectEffectResetDelay = 0;

FRect humanRect = FRect(28, 21);

mt19937 gen;

vector<FMapInfo::Type> autoCreateResList;

vector<FMapInfo\*> effectBlock;

};

}

#endif // !FORAGER\_FMAP\_H

/\* FileName: FMapImage.cpp \*/

﻿#include "stdafx.h"

#include "FMapImage.h"

#include "FRect.h"

namespace game\_framework

{

FAtlas FMapImage::weedImage;

//constructor 寫入地圖圖片資訊

FMapImage::FMapImage(int resID, int rgbMask, bool isCollision,

const FPosition& screenCorrect,

const FRect& rect, const FPosition& rectCorrect,

bool isCollectible,

int collectCount, int layout, int resource, int type,

bool isCreatable, bool isInteractable, int imgWidth,

int imgHeight,

int realMapLocDX, int realMapLocDY, int buildingExtraX,

int buildingExtraY) :

m\_screenCorrect(screenCorrect), m\_rectCorrect(rectCorrect),

m\_rect(rect),

isCollision(isCollision), rgbMask(rgbMask),

isCollectible(isCollectible), collectCount(collectCount),

layout(layout),

resource(resource), type(type),

isCreatable(isCreatable), resID(resID),

realMapLocDX(realMapLocDX),

realMapLocDY(realMapLocDY),

buildingExtraX(buildingExtraX), buildingExtraY(buildingExtraY),

isInteractable(isInteractable), imgWidth(imgWidth),

imgHeight(imgHeight)

{

}

//載入圖片

void FMapImage::LoadBitmap()

{

if(this->notReady)

{

this->m\_image.LoadBitmap(resID, rgbMask);

this->m\_effect.Initialize(&m\_image);

this->notReady = false;

}

}

//設定圖片螢幕位置

void FMapImage::SetImageTopLeft(int sx, int sy)

{

this->imgX = sx;

this->imgY = sy;

}

//設定碰撞矩形範圍位置

void FMapImage::SetRectTopLeft(int sx, int sy)

{

this->m\_rect.SetTopLeft(FPosition(sx + m\_rectCorrect.\_pX,

sy + m\_rectCorrect.\_pY));

}

//顯示圖片

void FMapImage::ShowBitmap(bool useEffect, WeedType weedType)

{

//若有使用特效

if(useEffect)

{

this->m\_effect.SetTopLeft(this->imgX + m\_screenCorrect.\_pX,

this->imgY + m\_screenCorrect.\_pY);

this->m\_effect.OnShow();

}

else

{

this->m\_image.SetTopLeft(this->imgX + m\_screenCorrect.\_pX,

this->imgY + m\_screenCorrect.\_pY);

this->m\_image.ShowBitmap();

}

if(weedType != WeedType::NOTHING)

{

this->weedImage.SetTopLeft(this->imgX, this->imgY);

this->weedImage.ShowPartBitmap(weedType);

}

}

}

/\* FileName: FMapImage.h \*/

﻿#ifndef FORAGER\_FMAPIMAGE\_H

#define FORAGER\_FMAPIMAGE\_H

#include "FCollectHighLightEffect.h"

#include "CGameStdLib.h"

#include "FPosition.h"

#include "FImage.h"

#include "FAtlas.h"

#include "FRect.h"

namespace game\_framework

{

class FMapImage

{

public:

enum WeedType

{

A, B, C, D, E, F, G, H, I, J, WEEDTYPE\_END, NOTHING

};

FMapImage(int resID, int rgbMask, bool enableCollision,

const FPosition& screenCorrect,

const FRect& rect, const FPosition& rectCorrect,

bool collectible,

int collectCount, int layout, int resource, int type,

bool isCreatable, bool isInteractable = false,

int imgWidth = 80,

int imgHeight = 80, int realMapLocDX = 0,

int realMapLocDY = 0, int buildingExtraX = 0,

int buildingExtraY = 0);

const FPosition m\_screenCorrect, m\_rectCorrect;

FImage m\_image;

FCollectHighLightEffect m\_effect;

FRect m\_rect;

const bool isCollision;

const bool isCollectible;

const bool isCreatable;

const bool isInteractable;

const int collectCount;

const int layout;

const int resource;

const int type;

const int resID;

const int rgbMask;

const int buildingExtraX;

const int buildingExtraY;

const int realMapLocDX;

const int realMapLocDY;

const int imgWidth;

const int imgHeight;

bool notReady = true;

void LoadBitmap();

void SetImageTopLeft(int sx, int sy);

void SetRectTopLeft(int sx, int sy);

void ShowBitmap(bool useEffect = false,

WeedType weedType = WeedType::NOTHING);

static FAtlas weedImage;

int imgX, imgY;

};

}

#endif // !FORAGER\_FMAPIMAGE\_H

/\* FileName: FMapInfo.cpp \*/

﻿#include "stdafx.h"

#include "FMapInfo.h"

#include "FResource.h"

namespace game\_framework

{

FMapImage FMapInfo::imageInfo[TYPE\_END] =

{

{IDB\_SEA, 0, true, FPosition(0, 0), FRect(80, 80), FPosition(0, 0), false, 0, 0, FResource::NOTHING, SEA, false},

{IDB\_GRASS, 0, false, FPosition(0, 0), FRect(0, 0), FPosition(0, 0), false, 0, 0, FResource::NOTHING, GRASS, true},

{IDB\_TREE, RGB(192, 192, 192), true, FPosition(0, -74), FRect(37, 30), FPosition(21, 31), true, 5, 3, FResource::WOOD\_R, TREE, true},

{IDB\_BERRY, RGB(192, 192, 192), true, FPosition(0, 0), FRect(50, 51), FPosition(15, 13), true, 3, 2, FResource::BERRY\_R, BERRY, true},

{IDB\_ROCK, RGB(192, 192, 192), true, FPosition(0, 0), FRect(50, 54), FPosition(14, 15), true, 8, 2, FResource::STONE\_R, ROCK, true},

{IDB\_COAL, RGB(192, 192, 192), true, FPosition(0, 0), FRect(53, 62), FPosition(16, 9), true, 8, 2, FResource::COAL\_R, COAL, true},

{IDB\_FURNACE1, RGB(192, 192, 192), true, FPosition(0, 0), FRect(60, 40), FPosition(20, 40), true, 10, 3, FResource::STONE\_R, FURNACE\_1, false, true, 160, 160, 0, 0, 1, 1},

{IDB\_FURNACE2, RGB(192, 192, 192), true, FPosition(0, 0), FRect(60, 40), FPosition(0, 40), true, 10, 3, FResource::STONE\_R, FURNACE\_2, false, true, 160, 160, -1, 0},

{IDB\_FURNACE3, RGB(192, 192, 192), true, FPosition(0, 0), FRect(60, 80), FPosition(20, 0), true, 10, 3, FResource::STONE\_R, FURNACE\_3, false, true, 160, 160, 0, -1},

{IDB\_FURNACE4, RGB(192, 192, 192), true, FPosition(0, 0), FRect(60, 80), FPosition(0, 0), true, 10, 3, FResource::STONE\_R, FURNACE\_4, false, true, 160, 160, -1, -1},

{IDB\_FORGE1, RGB(192, 192, 192), true, FPosition(0, 0), FRect(38, 38), FPosition(42, 42), true, 10, 3, FResource::STONE\_R, FORGE\_1, false, true, 160, 160, 0, 0, 1, 1},

{IDB\_FORGE2, RGB(192, 192, 192), true, FPosition(0, 0), FRect(38, 38), FPosition(0, 42), true, 10, 3, FResource::STONE\_R, FORGE\_2, false, true, 160, 160, -1, 0},

{IDB\_FORGE3, RGB(192, 192, 192), true, FPosition(0, 0), FRect(38, 38), FPosition(42, 0), true, 10, 3, FResource::STONE\_R, FORGE\_3, false, true, 160, 160, 0, -1},

{IDB\_FORGE4, RGB(192, 192, 192), true, FPosition(0, 0), FRect(38, 38), FPosition(0, 0), true, 10, 3, FResource::STONE\_R, FORGE\_4, false, true, 160, 160, -1, -1},

{IDB\_TORCH, RGB(192, 192, 192), true, FPosition(0, -20), FRect(10, 10), FPosition(35, 40), true, 3, 3, FResource::WOOD\_R, TORCH, false}

};

//constructor 前景後景指標設為空指標

FMapInfo::FMapInfo() : pBackImageInfo(nullptr),

pFrontImageInfo(nullptr)

{

}

//constructor 傳入前景後景Type並設置

FMapInfo::FMapInfo(Type backType, Type frontType)

{

if(backType != Type::NOTHING)

pBackImageInfo = &imageInfo[backType];

else

pBackImageInfo = nullptr;

if(frontType != Type::NOTHING)

pFrontImageInfo = &imageInfo[frontType];

else

pFrontImageInfo = nullptr;

}

//判斷此位置是否能建築

bool FMapInfo::IsBuildable()

{

if(pBackImageInfo == &imageInfo[Type::GRASS])

{

if(pFrontImageInfo == nullptr)

return true;

}

return false;

}

}

/\* FileName: FMapInfo.h \*/

﻿#pragma once

#ifndef FORAGER\_FMAPINFO\_H

#define FORAGER\_FMAPINFO\_H

#include "CGameStdLib.h"

#include "FGrassSeaEffect.h"

#include "FMapImage.h"

using namespace std;

namespace game\_framework

{

class FMapInfo

{

public:

enum Type

{

SEA,

GRASS,

TREE,

BERRY,

ROCK,

COAL,

FURNACE\_1,

FURNACE\_2,

FURNACE\_3,

FURNACE\_4,

FORGE\_1,

FORGE\_2,

FORGE\_3,

FORGE\_4,

TORCH,

TYPE\_END,

NOTHING

};

FMapInfo();

FMapInfo(Type backType, Type frontType);

FMapImage\* pFrontImageInfo = nullptr;

FMapImage\* pBackImageInfo = nullptr;

bool IsBuildable();

bool useEffect = false;

bool useSeaEffect = false;

int edgeFlag = 0;

FGrassSeaEffect seaEffect;

FMapImage::WeedType weedType = FMapImage::WeedType::NOTHING;

static FMapImage imageInfo[TYPE\_END];

};

}

#endif // !FORAGER\_FMAPINFO\_H

/\* FileName: FMenu.cpp \*/

﻿#include "stdafx.h"

#include "FMenu.h"

namespace game\_framework

{

FImage FMenu::backGroundImage;

int FMenu::state;

//讀取圖片及設定座標

void FMenu::Initialize(FHuman\* human, FBuildSystem\* buildSystem,

FProductBuildSystem\* pbms)

{

int blockBeginX = SIZE\_X / 2 - (STATEEND \* (MENUBLOCK\_WIDTH /

2));

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

menuBlockLoc[i] = FPosition(blockBeginX + (MENUBLOCK\_WIDTH \* i),

MENUBLOCK\_PY);

this->state = NOTHING;

this->human = human;

this->buildSystem = buildSystem;

this->pbms = pbms;

escMenu.Initialize(human);

menuBlock.LoadBitmap(IDB\_MENUBLOCK);

menuBlock\_img[SETTING].LoadBitmap(IDB\_MENU\_SETTING, RGB(192,

192, 192));

menuBlock\_img[BAG].LoadBitmap(IDB\_MENU\_BAG, RGB(192, 192, 192));

menuBlock\_img[BUILD].LoadBitmap(IDB\_MENU\_BUILD, RGB(192, 192,

192));

backGroundImage.LoadBitmap(IDB\_BAGBACKGROUND);

backGroundImage.SetTopLeft(0, 0);

backGroundImage.SetAlpha(BACKGROUND\_ALPHA);

}

//判斷是否按下遊戲結束按鈕

bool FMenu::GetExitGameSignal()

{

return escMenu.GetExitGameSignal();

}

//顯示包包, 建築, ESC, 互動建築選單

void FMenu::OnShow()

{

if(this->state != NOTHING && this->state != INTERACTIVEMENU)

{

ShowMenu();

if(this->state == BAG)

human->bag.ShowBag();

else if(this->state == BUILD)

buildSystem->ShowBuildSystem();

else if(this->state == SETTING)

escMenu.OnShow();

}

if(this->state == INTERACTIVEMENU)

pbms->OnShow();

}

//設定目前狀態

void FMenu::SetState(int state)

{

FMenu::state = state;

}

//判斷是否使用選單

bool FMenu::IsUsing()

{

return this->state != NOTHING;

}

//重置選單狀態

void FMenu::Reset()

{

buildSystem->Reset();

human->bag.Reset();

escMenu.Reset();

}

//顯示選單按鈕(最上面)

void FMenu::ShowMenu()

{

int blockBeginX = SIZE\_X / 2 - (STATEEND \* (MENUBLOCK\_WIDTH /

2));

if(this->state != BUILD)

backGroundImage.ShowBitmap();

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

{

menuBlock.SetTopLeft(menuBlockLoc[i].\_pX, menuBlockLoc[i].\_pY);

menuBlock\_img[i].SetTopLeft(menuBlockLoc[i].\_pX,

menuBlockLoc[i].\_pY, 96, 96);

if(this->state == i)

{

menuBlock\_img[i].SetAlpha(255);

menuBlock\_img[i].SetScale(1.25f);

}

else

{

menuBlock\_img[i].SetAlpha(MENU\_BLOCK\_IMG\_ALPHA);

menuBlock\_img[i].SetScale(1.0f);

}

menuBlock.ShowBitmap();

menuBlock\_img[i].ShowBitmap();

}

}

//回傳point位置按鈕索引

int FMenu::GetTargetMenuBlock(const CPoint& point)

{

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

{

if(point.x >= menuBlockLoc[i].\_pX &&

point.x < menuBlockLoc[i].\_pX + MENUBLOCK\_WIDTH &&

point.y >= menuBlockLoc[i].\_pY &&

point.y < menuBlockLoc[i].\_pY + MENUBLOCK\_WIDTH)

return i;

}

return NOTHING;

}

//滑鼠左鍵事件處理

void FMenu::OnLButtonDownHandle(const CPoint& point)

{

if(this->state != NOTHING && this->state != INTERACTIVEMENU)

{

int targetMenuBlock = GetTargetMenuBlock(point);

if(targetMenuBlock != NOTHING)

SetState(targetMenuBlock);

}

if(this->state == BAG)

human->bag.OnLButtonDownHandle(point);

else if(this->state == BUILD)

buildSystem->OnLButtonDownHandle(point);

else if(this->state == SETTING)

escMenu.OnLButtonDownHandle(point);

else if(this->state == INTERACTIVEMENU)

pbms->OnLButtonDownHandle(point);

}

//鍵盤事件處理

void FMenu::OnKeyDownHandle(UINT nChar)

{

const char KEY\_ESC = 0x1b;

const char KEY\_B = 0x42; // 包包

const char KEY\_E = 0x45; // 互動

if(nChar == KEY\_E && this->state == NOTHING &&

human->isInteractable &&

pbms->IsUsable())

this->state = INTERACTIVEMENU;

else if(nChar == KEY\_E && this->state == INTERACTIVEMENU)

{

this->state = NOTHING;

pbms->Reset();

}

else if(nChar == KEY\_ESC && this->state == NOTHING)

this->state = SETTING;

else if(nChar == KEY\_ESC && this->state != NOTHING)

this->state = NOTHING;

else if(nChar == KEY\_B && this->state != BAG)

this->state = BAG;

else if(nChar == KEY\_B && this->state == BAG)

this->state = NOTHING;

Reset();

}

//滑鼠移動事件處理

void FMenu::OnMouseMoveHandle(const CPoint& point)

{

if(this->state == BAG)

human->bag.OnMouseMoveHandle(point);

else if(this->state == BUILD)

buildSystem->OnMouseMoveHandle(point);

else if(this->state == SETTING)

escMenu.OnMouseMoveHandle(point);

if(this->state == INTERACTIVEMENU)

pbms->OnMouseMoveHandle(point);

}

}

/\* FileName: FMenu.h \*/

﻿#pragma once

#ifndef FORAGER\_FMENU\_H

#define FORAGER\_FMENU\_H

#include "FImage.h"

#include "FPosition.h"

#include "FHuman.h"

#include "FBuildSystem.h"

#include "FEscMenu.h"

#include "FProductBuildMenu.h"

#include "FProductBuildSystem.h"

#include "CGameStdLib.h"

using namespace std;

namespace game\_framework

{

class FMenu

{

public:

enum State

{

BAG,

BUILD,

SETTING,

STATEEND,

INTERACTIVEMENU,

NOTHING

};

static const int MENUBLOCK\_WIDTH = 96; //選單方塊寬度

static const int MENUBLOCK\_PY = 5; //選單位置高度

static const int BACKGROUND\_ALPHA = 150; //背景透明度

static const int MENU\_BLOCK\_IMG\_ALPHA =

100; //選單方塊透明度

FPosition menuBlockLoc[STATEEND];

void Initialize(FHuman\* human, FBuildSystem\* buidlSystem,

FProductBuildSystem\* pbms);

void OnShow();

void OnLButtonDownHandle(const CPoint& point);

void OnKeyDownHandle(UINT nChar);

void OnMouseMoveHandle(const CPoint& point);

static void SetState(int state);

bool IsUsing();

void Reset();

bool GetExitGameSignal();

private:

void ShowMenu();

FImage menuBlock;

FImage menuBlock\_img[STATEEND];

FHuman\* human;

FBuildSystem\* buildSystem;

FProductBuildSystem\* pbms;

FEscMenu escMenu;

int GetTargetMenuBlock(const CPoint& point);

static int state;

static FImage backGroundImage;

};

}

#endif // !FORAGER\_FMENU\_H

/\* FileName: FMouseTarget.cpp \*/

﻿#include "stdafx.h"

#include "FMouseTarget.h"

namespace game\_framework

{

//圖取圖片及設定速度, 透明度

void FMouseTarget::Initialize()

{

animationSpeed = 3;

mouseTarget[TOPLEFT].LoadBitmap(IDB\_TARGET\_TL, 1, 4, 40, 40, 4,

RGB(255, 0, 0));

mouseTarget[TOPLEFT].SetDelayCount(animationSpeed);

mouseTarget[TOPRIGHT].LoadBitmap(IDB\_TARGET\_TR, 1, 4, 40, 40, 4,

RGB(255, 0,

0));

mouseTarget[TOPRIGHT].SetDelayCount(animationSpeed);

mouseTarget[BOTTOMLEFT].LoadBitmap(IDB\_TARGET\_BL, 1, 4, 40, 40,

4, RGB(255, 0,

0));

mouseTarget[BOTTOMLEFT].SetDelayCount(animationSpeed);

mouseTarget[BOTTOMRIGHT].LoadBitmap(IDB\_TARGET\_BR, 1, 4, 40, 40,

4, RGB(255, 0,

0));

mouseTarget[BOTTOMRIGHT].SetDelayCount(animationSpeed);

this->isPointing = false;

}

//設定位置及寬度

void FMouseTarget::SetTopLeftWidth(const FPosition&

targetBlockScreenTopLeftLoc,

int width, int height, int dx, int dy)

{

int LocX = targetBlockScreenTopLeftLoc.\_pX;

int LocY = targetBlockScreenTopLeftLoc.\_pY;

mouseTarget[TOPLEFT].SetTopLeft(LocX + dx, LocY + dy);

mouseTarget[TOPRIGHT].SetTopLeft(LocX + dx + width - 40,

LocY + dy);

mouseTarget[BOTTOMLEFT].SetTopLeft(LocX + dx,

LocY + height - 40 + dy);

mouseTarget[BOTTOMRIGHT].SetTopLeft(LocX + dx + width - 40,

LocY + height - 40 + dy);

}

//設定動畫速度

void FMouseTarget::SetAnimationSpeed(int speed)

{

for(int i = TOPLEFT; i < LOCATIONEND; i++)

mouseTarget[i].SetDelayCount(speed);

}

//顯示滑鼠目標

void FMouseTarget::OnShow()

{

if(this->isPointing)

{

for(int i = TOPLEFT; i < LOCATIONEND; i++)

{

mouseTarget[i].OnShow();

mouseTarget[i].OnMove();

}

}

else

{

for(int i = TOPLEFT; i < LOCATIONEND; i++)

mouseTarget[i].Reset();

}

}

//設定是否指到目標

void FMouseTarget::SetPointing(bool isPointing)

{

this->isPointing = isPointing;

}

//判斷是否指到目標

bool FMouseTarget::GetIsPointing()

{

return isPointing;

}

}

/\* FileName: FMouseTarget.h \*/

﻿#ifndef FORAGER\_FMOUSETARGET\_H

#define FORAGER\_FMOUSETARGET\_H

#include "CGameStdLib.h"

#include "FPosition.h"

#include "FAnimation.h"

namespace game\_framework

{

class FMouseTarget

{

enum Location

{

TOPLEFT,

TOPRIGHT,

BOTTOMLEFT,

BOTTOMRIGHT,

LOCATIONEND,

};

public:

int animationSpeed;

void Initialize();

void SetTopLeftWidth(const FPosition&

targetBlockScreenTopLeftLoc, int width,

int height, int dx = 0, int dy = 0);

void SetAnimationSpeed(int speed);

void OnShow();

void SetPointing(bool isPointing);

bool GetIsPointing();

private:

FAnimation mouseTarget[LOCATIONEND];

FPosition targetBlockScreenTopLeftLoc;

bool isPointing;

};

}

#endif // !FORAGER\_FMOUSETARGET\_H

/\* FileName: FNightEffect.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FNightEffect.h"

using namespace std;

namespace game\_framework

{

FNightEffect\* FNightEffect::instance = NULL;

// 註冊 CallBack

FNightEffect::FNightEffect()

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FNightEffect::D2D1CallBack,

this));

}

// 移除 CallBack

FNightEffect::~FNightEffect()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

// 設定火把位置

void FNightEffect::RegisterTorch(int sx, int sy)

{

ComPtr<ID2D1EllipseGeometry> ellipseGeometry;

torchGeometry.push\_back(ellipseGeometry);

D2D1Draw::CreateEllipseGeometry(&D2D1::Ellipse(D2D1::Point2F((

FLOAT)sx + 40,

(FLOAT)sy + 40), 100.f, 100.f),

torchGeometry[torchGeometry.size() -

1].ReleaseAndGetAddressOf());

}

// 存檔

void FNightEffect::SaveEffect(FLOAT\* saveDataArray)

{

saveDataArray[0] = (FLOAT)mode;

saveDataArray[1] = (FLOAT)count;

saveDataArray[2] = brushAlpha;

saveDataArray[3] = lightBrushAlpha;

saveDataArray[4] = lightSolidBrushAlpha;

}

// 讀檔

void FNightEffect::LoadEffect(FLOAT\* saveDataArray)

{

mode = (INT)saveDataArray[0];

count = (INT)saveDataArray[1];

brushAlpha = saveDataArray[2];

lightBrushAlpha = saveDataArray[3];

lightSolidBrushAlpha = saveDataArray[4];

}

// 初始化

void FNightEffect::Init()

{

instance = this;

D2D1CallBack();

count = 0;

mode = DAY\_STATE;

torchGeometry.clear();

}

// 設定圖形筆刷透明度 (控制白天晚上)

void FNightEffect::NightModeCtrl()

{

if(mode == NIGHT\_STATE)

{

brushAlpha = 0.9f;

lightSolidBrushAlpha = 0.01f;

lightBrushAlpha = 0.3f;

if(count > countMax)

{

mode++;

count = 0;

}

}

else if(mode == DAY\_STATE)

{

brushAlpha = 0;

lightSolidBrushAlpha = 0;

lightBrushAlpha = 0;

if(count > countMax)

{

mode++;

count = 0;

}

}

else if(mode == DAY2NIGHT\_STATE)

{

brushAlpha += brushStep;

if(brushAlpha < 0.01f)

lightSolidBrushAlpha += brushStep;

if(count >= changeCountMax / 2)

lightBrushAlpha += lightBrushStep;

else

lightBrushAlpha = 0;

if(count > changeCountMax)

{

mode++;

count = 0;

}

}

else if(mode == NIGHT2DAY\_STATE)

{

brushAlpha -= brushStep;

if(brush->GetOpacity() < 0.01f)

lightSolidBrushAlpha -= brushStep;

if(count == 0)

lightBrushAlpha = 0.3f;

else if(count <= changeCountMax / 2)

lightBrushAlpha -= brushStep;

if(count > changeCountMax)

{

mode = DAY\_STATE;

count = 0;

}

}

// 設定筆刷透明度

brush->SetOpacity(brushAlpha);

lightSolidBrush->SetOpacity(lightSolidBrushAlpha);

lightBrush->SetOpacity(lightBrushAlpha);

}

// 顯示夜晚效果

void FNightEffect::OnShow()

{

count++;

NightModeCtrl();

if(mode != DAY\_STATE)

{

// 將所有光源區塊進行合成

ComPtr<ID2D1PathGeometry> mainPath1;

ComPtr<ID2D1PathGeometry> mainPath2;

ComPtr<ID2D1GeometrySink> mainSink;

D2D1Draw::CreatePathGeometry(

mainPath1.ReleaseAndGetAddressOf());

mainPath1->Open(mainSink.ReleaseAndGetAddressOf());

ellipseGeometry->CombineWithGeometry(ellipseGeometry.Get(),

D2D1\_COMBINE\_MODE\_UNION, NULL, NULL, mainSink.Get());

mainSink->Close();

mainPath2.Reset();

for(UINT i = 0; i < torchGeometry.size(); i++)

{

ComPtr<ID2D1PathGeometry>& mainPath = mainPath1.Get() ?

mainPath1 : mainPath2;

ComPtr<ID2D1PathGeometry>& tempPath = mainPath1.Get() ?

mainPath2 : mainPath1;

D2D1Draw::CreatePathGeometry(&tempPath);

tempPath->Open(mainSink.ReleaseAndGetAddressOf());

mainPath->CombineWithGeometry(torchGeometry[i].Get(),

D2D1\_COMBINE\_MODE\_UNION,

NULL, NULL, mainSink.Get());

mainSink->Close();

mainPath.Reset();

}

// 對光源區塊取反, 達到夜晚的效果

ID2D1PathGeometry\* mainPath = mainPath1.Get() ? mainPath1.Get()

:

mainPath2.Get();

ComPtr<ID2D1GeometrySink> geometrySink;

ComPtr<ID2D1PathGeometry> maskPath;

D2D1Draw::CreatePathGeometry(maskPath.ReleaseAndGetAddressOf());

D2D1Draw::CheckD2D1Fail(maskPath->Open(

geometrySink.ReleaseAndGetAddressOf()),

"Open Path Geometry fail.");

rectangleGeometry->CombineWithGeometry(mainPath,

D2D1\_COMBINE\_MODE\_EXCLUDE,

NULL, NULL, geometrySink.Get());

D2D1Draw::CheckD2D1Fail(geometrySink->Close(),

"Geometry Sink close fail.");

// 印出無光源區

D2D1Draw::FillGeometry(maskPath.Get(), brush.Get());

// 光源區夜晚效果

D2D1Draw::FillGeometry(mainPath, lightSolidBrush.Get());

D2D1Draw::FillGeometry(mainPath, lightBrush.Get());

}

torchGeometry.clear();

}

// 與 Direct2D 有關資源, 必要時被呼叫並重新創建

void FNightEffect::D2D1CallBack()

{

D2D1\_ELLIPSE ellipse = D2D1::Ellipse(D2D1::Point2F(SIZE\_X / 2,

SIZE\_Y / 2),

150.0f, 150.0f);

D2D1Draw::CreateSolidColorBrush(D2D1::ColorF(0, 0, 0),

brush.ReleaseAndGetAddressOf());

D2D1Draw::CreateSolidColorBrush(D2D1::ColorF(0, 0, 0),

lightSolidBrush.ReleaseAndGetAddressOf());

D2D1Draw::CreateEllipseGeometry(&D2D1::Ellipse(D2D1::Point2F(

ellipse.point.x,

ellipse.point.y), ellipse.radiusX,

ellipse.radiusY),

ellipseGeometry.ReleaseAndGetAddressOf());

D2D1Draw::CreateRectangleGeometry(&D2D1::RectF(0, 0, SIZE\_X,

SIZE\_Y),

rectangleGeometry.ReleaseAndGetAddressOf());

D2D1\_GRADIENT\_STOP gradientStops[2];

gradientStops[0].color = D2D1::ColorF(D2D1::ColorF(1, 1, 1));

gradientStops[0].position = 0.f;

gradientStops[1].color = D2D1::ColorF(D2D1::ColorF(0, 0, 0));

gradientStops[1].position = 1.f;

D2D1Draw::CreateGradientStopCollection(gradientStops, 2,

D2D1\_GAMMA\_2\_2,

D2D1\_EXTEND\_MODE\_CLAMP,

gradientStopsCollection.ReleaseAndGetAddressOf());

D2D1Draw::CreateRadialGradientBrush(

D2D1::RadialGradientBrushProperties(

ellipse.point, D2D1::Point2F(0, 0), ellipse.radiusX,

ellipse.radiusY), gradientStopsCollection.Get(),

lightBrush.ReleaseAndGetAddressOf());

}

}

/\* FileName: FNightEffect.h \*/

﻿#pragma once

#ifndef FORAGER\_FNIGHTEFFECT\_H

#define FORAGER\_FNIGHTEFFECT\_H

#include "CGameStdLib.h"

using namespace std;

namespace game\_framework

{

class FNightEffect

{

public:

FNightEffect();

~FNightEffect();

void Init();

void OnShow();

void D2D1CallBack();

void RegisterTorch(int sx, int sy);

void SaveEffect(FLOAT\* saveDataArray);

void LoadEffect(FLOAT\* saveDataArray);

static FNightEffect\* instance;

private:

enum NIGHT\_MODE

{

DAY\_STATE = 0,

DAY2NIGHT\_STATE = 1,

NIGHT\_STATE = 2,

NIGHT2DAY\_STATE = 3

};

void NightModeCtrl();

int callBackID = 0, count = 0;

int mode = DAY\_STATE;

const int countMax = 30 \* 5;

const int changeCountMax = 30 \* 5;

const FLOAT brushStep = 0.9f / countMax;

const FLOAT lightBrushStep = 0.3f / (changeCountMax / 2);

FLOAT brushAlpha = 0, lightSolidBrushAlpha = 0,

lightBrushAlpha = 0;

ComPtr<ID2D1EllipseGeometry> ellipseGeometry;

ComPtr<ID2D1RectangleGeometry> rectangleGeometry;

ComPtr<ID2D1SolidColorBrush> brush;

ComPtr<ID2D1GradientStopCollection> gradientStopsCollection;

ComPtr<ID2D1RadialGradientBrush> lightBrush;

ComPtr<ID2D1SolidColorBrush> lightSolidBrush;

vector<ComPtr<ID2D1EllipseGeometry> > torchGeometry;

};

}

#endif // !FORAGER\_FNIGHTEFFECT\_H

/\* FileName: Forager.h \*/

﻿#ifndef FORAGER\_FORAGER\_H

#define FORAGER\_FORAGER\_H

#include "CGameStateInit.h"

#include "CGameStateRun.h"

#include "CGameStateOver.h"

/\* 新增的 class 請加入至此 \*/

#include "FImage.h"

#include "FCamera.h"

#include "FMap.h"

#include "FHuman.h"

#include "FRect.h"

#include "FPosition.h"

#include "FMapImage.h"

#include "FBag.h"

#include "FResource.h"

#include "FText.h"

#include "FHealth.h"

#include "FCollectionBar.h"

#include "FBuildSystem.h"

#include "FMouseTarget.h"

#include "FMenu.h"

#include "FBuildSystemMenu.h"

#include "FBuildInfo.h"

#include "FEscMenu.h"

#include "FCollectParticle.h"

#include "FProductBuildMenu.h"

#include "FProductBuildSystem.h"

#include "FGameOverEffect.h"

#include "FOutlineText.h"

#include "FAnimation.h"

#include "FTool.h"

#endif // !FORAGER\_FORAGER\_H

/\* FileName: FOutlineText.cpp \*/

﻿#include "stdafx.h"

#include "FOutlineText.h"

namespace game\_framework

{

// constructor 設定初值及註冊 Callback Function

FOutlineText::FOutlineText(const char\* fontName, int fontSize,

COLORREF fillColor, COLORREF outlineColor, FLOAT outlineWeight,

int fontWeight)

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FOutlineText::D2D1CallBack,

this));

this->fontName = string(fontName);

this->fontSize = fontSize;

this->fontWeight = fontWeight;

this->fillColor = fillColor;

this->outlineColor = outlineColor;

this->outlineWeight = outlineWeight;

D2D1CallBack();

}

// destructor 解除註冊 Callback Function

FOutlineText::~FOutlineText()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

// 顯示文字

void FOutlineText::ShowText(const char\* text, int x, int y)

{

int strLen = strlen(text);

UINT\* strCode = new UINT[strLen];

UINT16\* indices = new UINT16[strLen];

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

strCode[i] = text[i];

D2D1Draw::CheckD2D1Fail(fontFace->GetGlyphIndices(strCode,

strLen, indices),

"Get Glyph Indices fail.");

static ComPtr<ID2D1PathGeometry> pathGeometry;

static ComPtr<ID2D1GeometrySink> geometrySink;

D2D1Draw::CreatePathGeometry(

pathGeometry.ReleaseAndGetAddressOf());

D2D1Draw::CheckD2D1Fail(pathGeometry->Open(

geometrySink.ReleaseAndGetAddressOf()),

"Open Path Geometry fail.");

D2D1Draw::CheckD2D1Fail(fontFace->GetGlyphRunOutline((

FLOAT)fontSize, indices,

NULL, NULL, strLen, FALSE, FALSE,

geometrySink.Get())

, "GetGlyphRunOutline fail.");

D2D1Draw::CheckD2D1Fail(geometrySink->Close(),

"Geometry Sink close fail.");

D2D1Draw::DrawDiretWriteOutlineText(text, strlen(text),

textFormat.Get(), x, y,

brush.Get(), outlineBrush.Get(),

pathGeometry.Get(), outlineWeight);

delete[] strCode;

delete[] indices;

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FOutlineText::D2D1CallBack()

{

CDC\* pDC = D2D1Draw::GetBackCDC();

fontCDC.CreateCompatibleDC(pDC);

hFont = CreateFont(fontSize, 0, 0, 0, fontWeight, false, false,

false,

DEFAULT\_CHARSET,

OUT\_DEFAULT\_PRECIS, CLIP\_DEFAULT\_PRECIS, CLEARTYPE\_QUALITY,

DEFAULT\_PITCH,

fontName.c\_str());

fontCDC.SelectObject(hFont);

D2D1Draw::CreateFontFaceFromHdc(fontCDC.m\_hDC,

fontFace.ReleaseAndGetAddressOf());

DeleteObject(hFont);

fontCDC.DeleteDC();

D2D1Draw::ReleaseBackCDC();

D2D1Draw::CreateSolidColorBrush(D2D1::ColorF(RGB(GetBValue(

fillColor),

GetGValue(fillColor), GetRValue(fillColor))),

brush.ReleaseAndGetAddressOf());

D2D1Draw::CreateSolidColorBrush(D2D1::ColorF(RGB(GetBValue(

outlineColor),

GetGValue(outlineColor), GetRValue(outlineColor))),

outlineBrush.ReleaseAndGetAddressOf());

wchar\_t nameBuffer[256] = { 0 };

swprintf(nameBuffer, 256, L"%hs", fontName.c\_str());

D2D1Draw::CreateTextFormat(nameBuffer, nullptr,

(DWRITE\_FONT\_WEIGHT)fontWeight,

DWRITE\_FONT\_STYLE\_NORMAL,

DWRITE\_FONT\_STRETCH\_NORMAL, (FLOAT)fontSize, L"en-us",

textFormat.ReleaseAndGetAddressOf());

}

}

/\* FileName: FOutlineText.h \*/

﻿#pragma once

#ifndef FORAGER\_FOUTLINETEXT\_H

#define FORAGER\_FOUTLINETEXT\_H

#include "CGameStdLib.h"

namespace game\_framework

{

class FOutlineText

{

public:

FOutlineText(const char\* fontName, int fontSize,

COLORREF fillColor,

COLORREF outlineColor, FLOAT outlineWeight = 1.0f,

int fontWeight = FW\_NORMAL);

~FOutlineText();

void D2D1CallBack();

void ShowText(const char\* text, int x, int y);

private:

int callBackID = 0, fontSize = 0, fontWeight = 0;

string fontName = "";

FLOAT outlineWeight = 1.0f;

CDC fontCDC;

HFONT hFont;

ComPtr<IDWriteFontFace> fontFace;

ComPtr<IDWriteTextFormat> textFormat;

ComPtr<ID2D1SolidColorBrush> brush;

ComPtr<ID2D1SolidColorBrush> outlineBrush;

COLORREF fillColor;

COLORREF outlineColor;

};

}

#endif // !FORAGER\_FOUTLINETEXT\_H

/\* FileName: FPosition.cpp \*/

﻿#include "stdafx.h"

#include "FPosition.h"

#include "FMap.h"

namespace game\_framework

{

//constructor 設定座標

FPosition::FPosition(int x, int y) : \_pX(x), \_pY(y)

{

}

//地圖座標轉換至螢幕TopLeft座標

FPosition FPosition::ConvertMapToScreenTopLeftPosition() const

{

return FPosition((this->\_pX \* FMap::BLOCK\_SIZE),

(this->\_pY \* FMap::BLOCK\_SIZE));

}

//人類座標轉換至地圖座標

FPosition FPosition::ConvertHumanToMapPosition() const

{

return FPosition((this->\_pX / 100), (this->\_pY / 100));

}

//人類座標轉換至螢幕座標

FPosition FPosition::ConvertHumanToScreenPosition() const

{

return FPosition(int(this->\_pX / 100.0 \* FMap::BLOCK\_SIZE),

int(this->\_pY / 100.0 \* FMap::BLOCK\_SIZE));

}

//人類座標轉換至螢幕TopLeft座標

FPosition FPosition::ConvertHumanToScreenTopLeftPosition() const

{

return FPosition((this->\_pX / 100) \* FMap::BLOCK\_SIZE,

(this->\_pY / 100) \* FMap::BLOCK\_SIZE);

}

//滑鼠座標轉換至地圖座標

FPosition FPosition::ConvertMouseToBlockMapPosition() const

{

return FPosition(int(this->\_pX / 100), int(this->\_pY / 100));

}

//滑鼠座標轉換至螢幕座標

FPosition FPosition::ConvertMouseToScreenTopLeftPosition() const

{

return FPosition(int(this->\_pX / 100), int(this->\_pY / 100));

}

}

/\* FileName: FPosition.h \*/

﻿#ifndef FORAGER\_FPOSITION\_H

#define FORAGER\_FPOSITION\_H

#include "CGameStdLib.h"

namespace game\_framework

{

class FPosition

{

public:

FPosition(int x = 0, int y = 0);

FPosition ConvertMapToScreenTopLeftPosition() const;

FPosition ConvertHumanToScreenPosition() const;

FPosition ConvertHumanToScreenTopLeftPosition() const;

FPosition ConvertHumanToMapPosition() const;

FPosition ConvertMouseToBlockMapPosition() const;

FPosition ConvertMouseToScreenTopLeftPosition() const;

int \_pX, \_pY;

};

}

#endif // !FORAGER\_FPOSITION\_H

/\* FileName: FProductBuildMenu.cpp \*/

﻿#include "stdafx.h"

#include "FProductBuildMenu.h"

#include "FHuman.h"

#include "FMenu.h"

#include "FMap.h"

namespace game\_framework

{

//constructor 設定合成程序數值

FProductionProcess::FProductionProcess(FPosition productMapLoc,

int resId,

int id, int count, int clock,

int time) :productMapLoc(productMapLoc), count(count),

clock(clock), time(time),

id(id), resId(resId)

{

}

//回傳目前合成程序數量

int FProductionProcess::GetCount()

{

return count;

}

//將程序數量減少

void FProductionProcess::SubCount()

{

count--;

}

//父類別constructor

FProductBuildMenu::FProductBuildMenu()

{

}

//讀取圖片及設定座標

void FProductBuildMenu::Initialize(FHuman\* human, FMap\* map,

const string& name)

{

menu\_img.LoadBitmap(IDB\_FUNCTION\_MENU, RGB(192, 192, 192));

menu\_img.SetTopLeft(MENU\_IMG\_X, MENU\_IMG\_Y, MENU\_IMG\_WIDTH,

MENU\_IMG\_HEIGHT);

menu\_min\_img.LoadBitmap(IDB\_MENU\_MIN\_IMG, RGB(192, 192, 192));

menu\_min\_img.SetTopLeft(MENU\_MIN\_IMG\_X, MENU\_MIN\_IMG\_Y,

MENU\_MIN\_IMG\_WIDTH,

MENU\_MIN\_IMG\_HEIGHT);

button.LoadBitmap(IDB\_BUILD\_BUTTON, 3, 1, 195, 60, RGB(192, 192,

192));

button\_min[0].LoadBitmap(IDB\_FUNCTION\_MENU\_SUB\_BUTTON, RGB(192,

192, 192));

button\_min[1].LoadBitmap(IDB\_FUNCTION\_MENU\_CRAFT\_BUTTON,

RGB(192, 192, 192));

button\_min[2].LoadBitmap(IDB\_FUNCTION\_MENU\_ADD\_BUTTON, RGB(192,

192, 192));

isChoice = false;

nowChoice = 0;

for(int i = 0; i < BUTTON\_COUNT; i++)

buttonLoc[i] = FPosition(MENU\_IMG\_X + 50,

MENU\_IMG\_Y + 100 + (MENU\_PADDING\*i + 1) + BUTTON\_HEIGHT \* i);

button\_minLoc[0] = FPosition(MENU\_MIN\_IMG\_X -11,

MENU\_IMG\_Y + MENU\_MIN\_IMG\_HEIGHT+8);

button\_minLoc[1] = FPosition(MENU\_MIN\_IMG\_X - 11 +

SUBADD\_BUTTON\_WIDTH,

MENU\_IMG\_Y + MENU\_MIN\_IMG\_HEIGHT + 10);

button\_minLoc[2] = FPosition(MENU\_MIN\_IMG\_X - 11 +

SUBADD\_BUTTON\_WIDTH +

CRAFT\_BUTTON\_WIDTH,

MENU\_IMG\_Y + MENU\_MIN\_IMG\_HEIGHT+8);

button\_min[0].SetTopLeft(button\_minLoc[0].\_pX,

button\_minLoc[0].\_pY);

button\_min[1].SetTopLeft(button\_minLoc[1].\_pX,

button\_minLoc[1].\_pY, 195, 60);

button\_min[2].SetTopLeft(button\_minLoc[2].\_pX,

button\_minLoc[2].\_pY);

mouseTarget.Initialize();

mouseTarget.SetAnimationSpeed(5);

this->human = human;

this->map = map;

this->name = name;

productCount = 1;

process\_circuit.LoadBitmap(IDB\_PROCESS\_CIRCUIT, 1, 5, 100, 100,

RGB(192, 192,

192));

process\_circuit.SetAlpha(180);

}

//顯示生產建築選單

void FProductBuildMenu::OnShow()

{

menu\_img.ShowBitmap();

main\_img.ShowBitmap();

static FOutlineText text("Microsoft JhengHei", 25, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f, FW\_HEAVY);

text.ShowText(this->name.c\_str(), MENU\_IMG\_X + 90,

MENU\_IMG\_Y + 20);

//大按鈕

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

{

int disY = 0;

button.SetTopLeft(buttonLoc[i].\_pX, buttonLoc[i].\_pY,

BUTTON\_WIDTH,

BUTTON\_HEIGHT);

button.ShowPartBitmap(0);

if(productionInfo[i].GetName().find('\n') != string::npos)

disY = -11;

product\_img.SetTopLeft(buttonLoc[i].\_pX - 5,

buttonLoc[i].\_pY - 5, 70, 70);

product\_img.ShowPartBitmap(i);

static FOutlineText text("Microsoft JhengHei", 20, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f, FW\_SEMIBOLD);

string temp\_str(productionInfo[i].GetName());

size\_t f = temp\_str.find(' ');

if(f != string::npos)

{

temp\_str[f] = '\0';

string temp\_str2(&(productionInfo[i].GetName()[f]));

text.ShowText(temp\_str.c\_str(), buttonLoc[i].\_pX + 65,

buttonLoc[i].\_pY + disY + 12);

text.ShowText(temp\_str2.c\_str(), buttonLoc[i].\_pX + 65,

buttonLoc[i].\_pY + disY + 32);

}

else

text.ShowText(temp\_str.c\_str(), buttonLoc[i].\_pX + 65,

buttonLoc[i].\_pY + disY + 20);

}

//小選單, 增減按鈕及合成按鈕

if(isChoice)

{

menu\_min\_img.ShowBitmap();

for(int i = 0; i < 3; i++)

button\_min[i].ShowBitmap();

static FOutlineText text("Microsoft JhengHei", 30, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f);

string temp\_str("CRAFT " + to\_string(productCount));

text.ShowText(temp\_str.c\_str(), button\_minLoc[1].\_pX +33,

button\_minLoc[1].\_pY +15);

product\_img.SetTopLeft(MENU\_MIN\_IMG\_X + MENU\_MIN\_IMG\_WIDTH / 2 -

50,

MENU\_MIN\_IMG\_Y - 40);

product\_img.ShowPartBitmap(nowChoice);

for(int i = 0; i < productionInfo[nowChoice].GetResTypeCount();

i++)

{

FImage resTemp = productionInfo[nowChoice].GetResImg(i);

resTemp.SetTopLeft(RES\_BEGIN\_X, RES\_BEGIN\_Y + (i \* 40), 70, 70);

resTemp.ShowBitmap();

static FOutlineText text("Microsoft JhengHei", 30, RGB(255, 255,

255), RGB(0, 0,

0), 3.5f);

string temp\_str(to\_string(human->bag.GetBagResCount(

productionInfo[nowChoice].GetResId(i))) + "/" + to\_string(

productionInfo[nowChoice].GetResCount(i) \* this->productCount));

text.ShowText(temp\_str.c\_str(), RES\_BEGIN\_X + 70,

RES\_BEGIN\_Y + (i \* 40) + 20);

}

}

mouseTarget.OnShow();

}

//滑鼠左鍵事件處理

int FProductBuildMenu::OnLButtonDownHandle(const CPoint& point)

{

int targetButton = GetTargetButton(point);

if(targetButton != -1)

{

if(targetButton >= 0)

{

isChoice = true;

nowChoice = targetButton;

mouseTarget.SetTopLeftWidth(buttonLoc[targetButton],

BUTTON\_WIDTH+40,

BUTTON\_HEIGHT+40, -20, -20);

mouseTarget.SetPointing(true);

return 0;

}

else

{

if(targetButton == -2)

this->productCount > 1 ? this->productCount-- :

this->productCount = 1;

if(targetButton == -3)

Product();

if(targetButton == -4)

this->productCount < 99 ? this->productCount++ :

this->productCount = 99;

}

}

else

{

isChoice = false;

mouseTarget.SetPointing(false);

}

return 0;

}

//滑鼠移動事件處理

void FProductBuildMenu::OnMouseMoveHandle(const CPoint& point)

{

int targetButton = GetTargetButton(point);

if(targetButton == -3)

button\_min[1].SetScale(1.08f);

else

button\_min[1].SetScale(1.0f);

}

//顯示合成程序進度條

void FProductBuildMenu::ShowProcessUI()

{

for(vector<FProductionProcess>::iterator it =

productionProcess.begin();

it != productionProcess.end(); it++)

{

if((\*it).GetCount() != 0)

{

const FMapImage\* block = map->GetRealBlock((\*it).productMapLoc);

FPosition realBlockLoc = map->GetRealBlockLoc((

\*it).productMapLoc);

FPosition humanScreenLoc =

human->Loc.ConvertHumanToScreenPosition();

FPosition blockScreenTopLeftLoc =

realBlockLoc.ConvertMapToScreenTopLeftPosition();

FPosition targetBlockScreenTopLeftLoc = FPosition(

blockScreenTopLeftLoc.\_pX -

(humanScreenLoc.\_pX - (SIZE\_X / 2)),

blockScreenTopLeftLoc.\_pY - (humanScreenLoc.\_pY -

(SIZE\_Y / 2)));

process\_circuit.SetTopLeft(targetBlockScreenTopLeftLoc.\_pX +

block->imgWidth / 2

- 50,

targetBlockScreenTopLeftLoc.\_pY + block->imgHeight / 2 - 50);

product\_img.SetTopLeft(targetBlockScreenTopLeftLoc.\_pX +

block->imgWidth / 2 -

50,

targetBlockScreenTopLeftLoc.\_pY + block->imgHeight / 2 - 50);

process\_circuit.ShowPartBitmap((\*it).time);

product\_img.ShowPartBitmap((\*it).id);

static FText text(L"Consolas", 30, RGB(255, 255, 255),

DWRITE\_FONT\_WEIGHT\_MEDIUM);

string temp\_str("x" + to\_string((\*it).GetCount()));

text.ShowText(temp\_str.c\_str(),

targetBlockScreenTopLeftLoc.\_pX + block->imgWidth / 2,

targetBlockScreenTopLeftLoc.\_pY + block->imgHeight / 2);

}

}

}

//合成(將資源放入程序列表)

void FProductBuildMenu::Product()

{

if(human->bag.IsEnoughToBuild(

productionInfo[nowChoice].GetReq(), productCount))

{

productionProcess.push\_back(FProductionProcess(

FProductBuildSystem::nowInteractiveMapLoc,

productionInfo[nowChoice].GetThisResId(), nowChoice,

productCount));

human->bag.ReduceRes(productionInfo[nowChoice].GetReq(),

productCount);

FMenu::SetState(FMenu::NOTHING);

Reset();

}

}

//重置生產建築選單狀態

void FProductBuildMenu::Reset()

{

this->productCount = 1;

this->isChoice = false;

this->nowChoice = 0;

mouseTarget.SetPointing(false);

}

//判斷此建築是否有在生產資源

bool FProductBuildMenu::IsUsable(FPosition targetMapLoc)

{

for(vector<FProductionProcess>::iterator it =

productionProcess.begin();

it != productionProcess.end(); it++)

{

if((\*it).productMapLoc.\_pX == targetMapLoc.\_pX &&

(\*it).productMapLoc.\_pY == targetMapLoc.\_pY)

return (\*it).GetCount() == 0;

}

return true;

}

//重置特效

void FProductBuildMenu::EffectReset()

{

button\_min[1].SetScale(1);

}

//回傳point位置按鈕索引

int FProductBuildMenu::GetTargetButton(const CPoint& point)

{

//大按鈕

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

{

if(point.x >= buttonLoc[i].\_pX &&

point.x < buttonLoc[i].\_pX + BUTTON\_WIDTH &&

point.y >= buttonLoc[i].\_pY &&

point.y < buttonLoc[i].\_pY + BUTTON\_HEIGHT)

return i;

}

//增減, 合成按鈕

if(isChoice)

{

if(point.x >= button\_minLoc[0].\_pX &&

point.x < button\_minLoc[0].\_pX + SUBADD\_BUTTON\_WIDTH &&

point.y >= button\_minLoc[0].\_pY &&

point.y < button\_minLoc[0].\_pY + SUBADD\_BUTTON\_HEIGHT)

return -2;

if(point.x >= button\_minLoc[1].\_pX &&

point.x < button\_minLoc[1].\_pX + CRAFT\_BUTTON\_WIDTH &&

point.y >= button\_minLoc[1].\_pY &&

point.y < button\_minLoc[1].\_pY + CRAFT\_BUTTON\_HEIGHT)

return -3;

if(point.x >= button\_minLoc[2].\_pX &&

point.x < button\_minLoc[2].\_pX + SUBADD\_BUTTON\_WIDTH &&

point.y >= button\_minLoc[2].\_pY &&

point.y < button\_minLoc[2].\_pY + SUBADD\_BUTTON\_HEIGHT)

return -4;

}

return -1;

}

//熔爐constructor

FFurnaceMenu::FFurnaceMenu():FProductBuildMenu()

{

}

//讀取圖片及設定數值

void FFurnaceMenu::Initialize(FHuman\* human, FMap\* map,

const string& name)

{

FProductBuildMenu::Initialize(human, map, name);

main\_img.LoadBitmap(IDB\_FURNACE, RGB(192, 192, 192));

main\_img.SetTopLeft(MENU\_IMG\_X + 10, MENU\_IMG\_Y+2, 60, 60);

product\_img.LoadBitmap(IDB\_FURNACE\_RES, 1, FURNACEEND, 100, 100,

RGB(192, 192,

192));

productionInfo[COAL].Initialize(FBuildInfo::COAL);

productionInfo[BRICK].Initialize(FBuildInfo::BRICK);

productionInfo[IRON\_IGNOT].Initialize(FBuildInfo::IRON\_IGNOT);

productionInfo[GOLD\_IGNOT].Initialize(FBuildInfo::GOLD\_IGNOT);

productionInfo[STEEL].Initialize(FBuildInfo::STEEL);

productionInfo[GLASS].Initialize(FBuildInfo::GLASS);

buttonCount = FURNACEEND;

}

//合成程序更新

void FFurnaceMenu::OnProcess()

{

for(vector<FProductionProcess>::iterator it =

productionProcess.begin();

it != productionProcess.end();)

{

if((\*it).GetCount() <= 0)

it = productionProcess.erase(it);

else

{

(\*it).clock--;

(\*it).time = (\*it).clock / 30;

if((\*it).clock == 0)

{

human->bag.GetRes(FResource::NewResource((\*it).resId), 1);

(\*it).SubCount();

(\*it).clock = 150;//預設

}

++it;

}

}

}

//鍛造台constructor

FForgeMenu::FForgeMenu() :FProductBuildMenu()

{

}

//讀取圖片及設定數值

void FForgeMenu::Initialize(FHuman\* human, FMap\* map,

const string& name)

{

FProductBuildMenu::Initialize(human, map, name);

main\_img.LoadBitmap(IDB\_FORGE, RGB(192, 192, 192));

main\_img.SetTopLeft(MENU\_IMG\_X + 10, MENU\_IMG\_Y + 2, 60, 60);

product\_img.LoadBitmap(IDB\_FORGE\_RES, 1, FORGEEND, 100, 100,

RGB(192, 192,

192));

productionInfo[PICKAXE].Initialize(FBuildInfo::PICKAXE);

productionInfo[PICKAXE2].Initialize(FBuildInfo::PICKAXE2);

productionInfo[PICKAXE3].Initialize(FBuildInfo::PICKAXE3);

buttonCount = FORGEEND;

}

//合成程序更新

void FForgeMenu::OnProcess()

{

for(vector<FProductionProcess>::iterator it =

productionProcess.begin();

it != productionProcess.end();)

{

if((\*it).GetCount() <= 0)

it = productionProcess.erase(it);

else

{

(\*it).clock--;

(\*it).time = (\*it).clock / 30;

if((\*it).clock == 0)

{

human->nowTool = (\*it).id;

human->ChangeTool();

(\*it).SubCount();

(\*it).clock = 150;//預設

}

++it;

}

}

}

}

/\* FileName: FProductBuildMenu.h \*/

﻿#ifndef FORAGER\_FPRODUCTBUILDMENU\_H

#define FORAGER\_FPRODUCTBUILDMENU\_H

#include "FImage.h"

#include "FMouseTarget.h"

#include "FAtlas.h"

#include "FPosition.h"

#include "FMapInfo.h"

#include "FBuildInfo.h"

#include "FText.h"

#include "FOutlineText.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FHuman;

class FMenu;

class FMap;

//合成程序

class FProductionProcess

{

public:

FProductionProcess(FPosition productMapLoc, int resId, int id,

int count = 1,

int clock = 150, int time = 0);

int clock;

int time;

int GetCount();

int id;

void SubCount();

FPosition productMapLoc;

int resId;

private:

int count;

};

class FProductBuildMenu

{

public:

static const int MENU\_IMG\_WIDTH =

250; //選單圖片寬度

static const int MENU\_IMG\_HEIGHT =

623; //選單圖片高度

static const int MENU\_IMG\_X = SIZE\_X / 2 - MENU\_IMG\_WIDTH / 2

; //選單圖片座標x

static const int MENU\_IMG\_Y =

30; //選單圖片座標y

static const int MENU\_MIN\_IMG\_WIDTH =

250; //小選單圖片寬度

static const int MENU\_MIN\_IMG\_HEIGHT =

250; //小選單圖片高度

static const int MENU\_MIN\_IMG\_X = MENU\_IMG\_X +

MENU\_MIN\_IMG\_WIDTH +

50; //小選單圖片座標x

static const int MENU\_MIN\_IMG\_Y =

MENU\_IMG\_Y; //小選單圖片座標y

static const int MENU\_PADDING = 20; //選單內距

static const int BUTTON\_WIDTH = 150; //按鈕寬度

static const int BUTTON\_HEIGHT = 60; //按鈕高度

static const int BUTTON\_COUNT =

6; //選單按鈕數量

static const int SUBADD\_BUTTON\_HEIGHT =

60; //增減按鈕高度

static const int CRAFT\_BUTTON\_HEIGHT =

60; //合成按鈕高度

static const int SUBADD\_BUTTON\_WIDTH =

40; //增減按鈕寬度

static const int CRAFT\_BUTTON\_WIDTH =

195; //合成按鈕寬度

static const int RES\_BEGIN\_X = MENU\_MIN\_IMG\_X +

50; //合成需求圖片起始座標x

static const int RES\_BEGIN\_Y = MENU\_MIN\_IMG\_Y +

MENU\_MIN\_IMG\_HEIGHT/2 -

100; //合成需求圖片起始座標y

FProductBuildMenu();

void Initialize(FHuman\* human, FMap\* map, const string& name);

int OnLButtonDownHandle(const CPoint& point);

void OnMouseMoveHandle(const CPoint& point);

void OnShow();

void ShowProcessUI();

void Product();

void OnProcess();

void Reset();

bool IsUsable(FPosition targetMapLoc);

protected:

int GetTargetButton(const CPoint& point);

void EffectReset();

string name;

FImage menu\_img;

FImage menu\_min\_img;

FImage button\_min[3];

FImage main\_img;

FAtlas button;

FAtlas product\_img;

FAtlas process\_circuit;

FBuildInfo productionInfo[10];

FPosition buttonLoc[BUTTON\_COUNT];

FPosition button\_minLoc[3];

int productCount;

int nowChoice;

int buttonCount;

bool isChoice;

vector<FProductionProcess> productionProcess;

FHuman\* human;

FMap\* map;

FMouseTarget mouseTarget;

};

class FFurnaceMenu : public FProductBuildMenu

{

public:

enum Furnace

{

COAL,

BRICK,

IRON\_IGNOT,

GOLD\_IGNOT,

STEEL,

GLASS,

FURNACEEND,

NOTHING

};

FFurnaceMenu();

void Initialize(FHuman\* human, FMap\* map, const string& name);

void OnProcess();

};

class FForgeMenu : public FProductBuildMenu

{

public:

enum Furnace

{

PICKAXE,

PICKAXE2,

PICKAXE3,

FORGEEND,

NOTHING

};

FForgeMenu();

void Initialize(FHuman\* human, FMap\* map, const string& name);

void OnProcess();

};

}

#endif // !FORAGER\_FPRODUCTBUILDMENU\_H

/\* FileName: FProductBuildSystem.cpp \*/

﻿#include "stdafx.h"

#include "FProductBuildSystem.h"

#include "FHuman.h"

namespace game\_framework

{

int FProductBuildSystem::nowInteractive = NOTHING;

FPosition FProductBuildSystem::nowInteractiveMapLoc;

FImage FProductBuildSystem::interactiveE;

//讀取圖片及設定數值

void FProductBuildSystem::Initialize(FHuman\* human, FMap\* map)

{

this->human = human;

this->map = map;

furnaceMenu.Initialize(human, map, "FURNACE");

forgeMenu.Initialize(human, map, "FORGE");

interactiveE.LoadBitmap(IDB\_INTERACTIVE\_E, RGB(192, 192, 192));

}

//滑鼠左鍵事件處理

void FProductBuildSystem::OnLButtonDownHandle(

const CPoint& point)

{

if(nowInteractive == FURNACE)

furnaceMenu.OnLButtonDownHandle(point);

else if(nowInteractive == FORGE)

forgeMenu.OnLButtonDownHandle(point);

}

//滑鼠移動事件處理

void FProductBuildSystem::OnMouseMoveHandle(const CPoint& point)

{

if(nowInteractive == FURNACE)

furnaceMenu.OnMouseMoveHandle(point);

else if(nowInteractive == FORGE)

forgeMenu.OnMouseMoveHandle(point);

}

//顯示生產建築系統

void FProductBuildSystem::OnShow()

{

if(nowInteractive == FURNACE)

furnaceMenu.OnShow();

else if(nowInteractive == FORGE)

forgeMenu.OnShow();

}

//判斷目標建築是否可互動

bool FProductBuildSystem::IsUsable()

{

if(nowInteractive == FURNACE)

return furnaceMenu.IsUsable(nowInteractiveMapLoc);

else if(nowInteractive == FORGE)

return forgeMenu.IsUsable(nowInteractiveMapLoc);

return true;

}

//合成程序更新

void FProductBuildSystem::OnProcess()

{

furnaceMenu.OnProcess();

forgeMenu.OnProcess();

}

//顯示合成程序進度條

void FProductBuildSystem::ShowProcessUI()

{

furnaceMenu.ShowProcessUI();

forgeMenu.ShowProcessUI();

}

//將MapInfo的ID轉換為生產建築系統的ID

int FProductBuildSystem::ConverToPB(int id)

{

if(id == FMapInfo::FURNACE\_1)

return FURNACE;

if(id == FMapInfo::FORGE\_1)

return FORGE;

return NOTHING;

}

//重置生產建築系統狀態

void FProductBuildSystem::Reset()

{

FProductBuildSystem::nowInteractive = NOTHING;

furnaceMenu.Reset();

forgeMenu.Reset();

}

//顯示互動按鍵(滑鼠移到可互動建築上會顯示"E")

void FProductBuildSystem::ShowIteractiveE()

{

if(human->isInteractable && IsUsable())

interactiveE.ShowBitmap();

}

}

/\* FileName: FProductBuildSystem.h \*/

﻿#ifndef FORAGER\_FPRODUCTBUILDSYSTEM\_H

#define FORAGER\_FPRODUCTBUILDSYSTEM\_H

#include "FImage.h"

#include "FProductBuildMenu.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FHuman;

class FProductBuildSystem

{

public:

enum ProductBuildMenu

{

FURNACE,

FORGE,

PRODUCTBUILDMENUEND,

NOTHING

};

static int ConverToPB(int id);

static int nowInteractive;

static FPosition nowInteractiveMapLoc;

static FImage interactiveE;

void Initialize(FHuman\* human, FMap\* map);

void OnLButtonDownHandle(const CPoint& point);

void OnMouseMoveHandle(const CPoint& point);

void OnShow();

void OnProcess();

void ShowProcessUI();

void ShowIteractiveE();

void Reset();

bool IsUsable();

private:

FFurnaceMenu furnaceMenu;

FForgeMenu forgeMenu;

FHuman\* human;

FMap\* map;

};

}

#endif // !FORAGER\_FPRODUCTBUILDSYSTEM\_H

/\* FileName: FRect.cpp \*/

﻿#include "stdafx.h"

#include "FRect.h"

#include "FPosition.h"

namespace game\_framework

{

//constructor 設定數值

FRect::FRect(int width, int height) : width(width),

height(height),

\_topLeft(FPosition(0, 0))

{

}

//設定螢幕座標

void FRect::SetTopLeft(const FPosition& topLeft)

{

\_topLeft = topLeft;

}

//判斷是否碰撞

bool FRect::IsCollision(const FRect&

x) // AABB (Axis Aligned Bounding Box)

{

return this->\_topLeft.\_pX < x.\_topLeft.\_pX + x.width &&

this->\_topLeft.\_pX + this->width > x.\_topLeft.\_pX &&

this->\_topLeft.\_pY < x.\_topLeft.\_pY + x.height &&

this->height + this->\_topLeft.\_pY > x.\_topLeft.\_pY;

}

}

/\* FileName: FRect.h \*/

﻿#ifndef FORAGER\_FRECT\_H

#define FORAGER\_FRECT\_H

#include "CGameStdLib.h"

#include "FPosition.h"

namespace game\_framework

{

class FRect

{

public:

FRect(int width = 0, int height = 0);

void SetTopLeft(const FPosition& topLeft);

bool IsCollision(const FRect& x);

int width;

int height;

private:

FPosition \_topLeft;

};

}

#endif // !FORAGER\_FRECT\_H

/\* FileName: FResource.cpp \*/

﻿#include "stdafx.h"

#include "FResource.h"

#include "FHuman.h"

namespace game\_framework

{

FImage FResource::resImages[RESOURCEIDEND] = {};

CString FResource::resNames[RESOURCEIDEND] = { "Wood", "Stone", "Berry", "Coal", "Brick", "Glass", "Steel", "Gold Ignot", "Iron Ignot" };

//父類別constructor 設定數值

FResource::FResource(int id, bool isusable)

{

this->id = id;

this->name = resNames[id];

this->isUsable = isusable;

this->count = 0;

}

//回傳資源ID

int FResource::GetId()

{

return id;

}

//增加資源數量

void FResource::AddCount(int count)

{

this->count += count;

}

//獲得資源數量

int FResource::GetCount()

{

return count;

}

//印出資源圖片

void FResource::OnShow(const FPosition& Loc)

{

resImages[id].SetTopLeft(Loc.\_pX, Loc.\_pY);

resImages[id].ShowBitmap();

}

//讀取所有資源圖片

void FResource::Initialize()

{

int resImagesName[RESOURCEIDEND] = {IDB\_WOOD\_R, IDB\_STONE\_R, IDB\_BERRY\_R, IDB\_COAL\_R, IDB\_BRICK\_R, IDB\_GLASS\_R, IDB\_STEEL\_R, IDB\_GOLD\_IGNOT\_R, IDB\_IRON\_IGNOT\_R };

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

resImages[i].LoadBitmap(resImagesName[i], RGB(192, 192, 192));

}

//回傳ID對應的資源

FResource\* FResource::NewResource(int id)

{

if(id == WOOD\_R || id == STONE\_R || id == COAL\_R)

return new Natural(id);

else if(id == BRICK\_R || id == GLASS\_R || id == STEEL\_R ||

id == GOLD\_IGNOT\_R ||

id == IRON\_IGNOT\_R)

return new Machining(id);

else if(id == BERRY\_R)

return new Food(id);

return nullptr;

}

//回傳資源名字

const CString& FResource::GetName()

{

return name;

}

//判斷資源是否可用

bool FResource::GetIsUsable()

{

return this->isUsable;

}

//使用資源

void FResource::UseResource(FHuman\* human)

{

}

//自然資源constructor

Natural::Natural(int id) :FResource(id)

{

}

//加工品constructor

Machining::Machining(int id) : FResource(id)

{

}

//食物constructor

Food::Food(int id, int hungerValue) : FResource(id, true)

{

this->hungerValue = hungerValue;

}

//食用食物(回復人類血量)

void Food::UseResource(FHuman\* human)

{

human->humanHealth.SubHunger(this->hungerValue);

}

}

/\* FileName: FResource.h \*/

﻿#ifndef FORAGER\_FRESOURCE\_H

#define FORAGER\_FRESOURCE\_H

#include "FPosition.h"

#include "FImage.h"

#include "CGameStdLib.h"

namespace game\_framework

{

class FHuman;

class FResource

{

public:

enum ResourceID

{

WOOD\_R,

STONE\_R,

BERRY\_R,

COAL\_R,

BRICK\_R,

GLASS\_R,

STEEL\_R,

GOLD\_IGNOT\_R,

IRON\_IGNOT\_R,

RESOURCEIDEND,

NOTHING

};

FResource(int id, bool isusable = false);

int GetId();

int GetCount();

void AddCount(int count);

void OnShow(const FPosition& Loc);

static void Initialize();

static FImage resImages[RESOURCEIDEND];

static CString resNames[RESOURCEIDEND];

static FResource\* NewResource(int id);

const CString& GetName();

bool GetIsUsable();

virtual ~FResource()=default;

virtual void UseResource(FHuman\* human);

protected:

int id;

int count;

bool isUsable;

CString name;

};

class Natural :public FResource

{

public:

Natural(int id);

};

class Food :public FResource

{

public:

Food(int id, int hungerValue = 5);

void UseResource(FHuman\* human);

private:

int hungerValue;

};

class Machining :public FResource

{

public:

Machining(int id);

};

}

#endif // !FORAGER\_FRESOURCE\_H

/\* FileName: FSaveButton.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FSaveButton.h"

using namespace std;

namespace game\_framework

{

// 讀取圖片及設定數值

void FSaveButton::Initialize(FSaveImage\* saveImage)

{

baseImage.LoadBitmap(IDB\_SAVEBUTTONBK);

maskImage.LoadBitmap(IDB\_SAVEBUTTONBKMASK);

timeLogo.LoadBitmap(IDB\_TIMELOGO, RGB(0, 0, 0));

notice.LoadBitmap(IDB\_MAINMENUNOTICE, RGB(56, 152, 255));

firstEffect = true;

useEffect = false;

ID2D1Bitmap\* bmpTemp = baseImage.GetBmpRes();

this->w = (int)bmpTemp->GetSize().width;

this->h = (int)bmpTemp->GetSize().height;

this->scale = 1.3f;

this->step = -0.04f;

this->load = true;

this->saveImage = saveImage;

D2DCallBack();

}

// 設定圖片螢幕位置

void FSaveButton::SetTopLeft(int x, int y)

{

baseImage.SetTopLeft(x, y);

notice.SetTopLeft(x + 30, y + 320);

saveImage->SetTopLeft(x + 18, y + 18);

timeLogo.SetTopLeft(x + 25, y + 119, 50, 50);

this->x = x;

this->y = y;

}

// 設定是否使用特效

void FSaveButton::SetUseEffect(bool useEffect,

bool deleteEffect)

{

// 刪除存檔特效(旋轉)

this->deleteEffect = deleteEffect;

if(!this->deleteEffect)

{

this->maxDegree = 10;

this->degree = 0;

this->degreeOffset = 2;

}

// 縮放特效

if(this->useEffect && useEffect)

return;

this->useEffect = useEffect;

if(this->useEffect)

{

this->step = 0.02f;

this->scale = 1.0f;

}

}

// 存檔方塊第一段特效(縮小)

void FSaveButton::ShowFirstEffect()

{

if(step > 0 && scale >= 1)

firstEffect = false;

if(scale <= 0.8)

{

step = 0.02f;

scale = 0.8f;

}

scale += step;

D2D1Draw::DrawEffect(alphaEffect.Get(), this->x, this->y, w, h,

scale, scale);

if(saveImageAlphaEffect.Get())

D2D1Draw::DrawEffect(saveImageAlphaEffect.Get(), this->x + 18,

this->y + 18,

FSAVEIMAGE\_SIZE, FSAVEIMAGE\_SIZE, scale, scale);

}

// 顯示特效

void FSaveButton::ShowEffect()

{

// 儲存方塊第二段特效(放大)

if(scale < 1.08)

scale += step;

// 刪除存檔特效

if(deleteEffect && saveImageAlphaEffect.Get())

{

if(degree > maxDegree || degree < -maxDegree)

{

maxDegree += 5;

degreeOffset = (degreeOffset >= 0) ? degreeOffset + 1 :

degreeOffset - 1;

}

degree += degreeOffset;

if(degree > maxDegree)

degreeOffset = -degreeOffset;

else if(degree < -maxDegree)

degreeOffset = -degreeOffset;

D2D1Draw::DrawEffect(alphaEffect.Get(), this->x, this->y, w, h,

scale, scale,

degree);

D2D1Draw::DrawEffect(saveImageAlphaEffect.Get(), this->x + 18,

this->y + 18,

FSAVEIMAGE\_SIZE, FSAVEIMAGE\_SIZE, scale, scale,

degree);

D2D1Draw::DrawEffect(maskImageAlphaEffect.Get(), this->x + 18,

this->y + 18,

FSAVEIMAGE\_SIZE, FSAVEIMAGE\_SIZE, scale, scale,

degree);

}

else

{

D2D1Draw::DrawEffect(alphaEffect.Get(), this->x, this->y, w, h,

scale, scale);

if(saveImageAlphaEffect.Get())

{

D2D1Draw::DrawEffect(saveImageAlphaEffect.Get(), this->x + 18,

this->y + 18,

FSAVEIMAGE\_SIZE, FSAVEIMAGE\_SIZE, scale, scale);

D2D1Draw::DrawEffect(maskImageAlphaEffect.Get(), this->x + 18,

this->y + 18,

FSAVEIMAGE\_SIZE, FSAVEIMAGE\_SIZE, scale, scale);

// 顯示遊玩時間

if(scale >= 1.08)

{

static FOutlineText text("Consolas", 40, RGB(255, 255, 255),

RGB(0, 0, 0),

3.5f);

int time = saveImage->GetTime();

int hour = (time / 60) / 60;

int minute = (time / 60) - (hour \* 60);

int sec = time - (minute \* 60) - (hour \* 60 \* 60);

string temp\_str(to\_string((hour / 10) % 100) + to\_string(

hour % 10) + ":" +

to\_string((minute / 10) % 100) + to\_string(minute % 10) + ":" +

to\_string((sec / 10) % 100) + to\_string(sec % 10));

text.ShowText(temp\_str.c\_str(), this->x + 90, this->y + 130);

timeLogo.ShowBitmap();

}

}

}

if(saveImageAlphaEffect.Get())

notice.ShowBitmap();

}

// destructor 解除註冊 Callback Function

FSaveButton::~FSaveButton()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

// constructor 註冊 Callback Function

FSaveButton::FSaveButton()

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FSaveButton::D2DCallBack,

this));

}

// 顯示儲存按鈕

void FSaveButton::OnShow()

{

if(firstEffect)

ShowFirstEffect();

else if(useEffect)

ShowEffect();

else

{

baseImage.ShowBitmap();

saveImage->ShowBitmap();

}

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FSaveButton::D2DCallBack()

{

if(this->load)

{

D2D1Draw::CreateEffect(CLSID\_D2D1Opacity,

alphaEffect.ReleaseAndGetAddressOf());

alphaEffect->SetInput(0, baseImage.GetBmpRes());

alphaEffect->SetValue(D2D1\_OPACITY\_PROP\_OPACITY, 1.0f);

D2D1Draw::CreateEffect(CLSID\_D2D1Opacity,

maskImageAlphaEffect.ReleaseAndGetAddressOf());

maskImageAlphaEffect->SetInput(0, maskImage.GetBmpRes());

maskImageAlphaEffect->SetValue(D2D1\_OPACITY\_PROP\_OPACITY, 0.3f);

saveImage->D2DCallBack();

if(saveImage->GetBmpRes() == NULL)

\*(saveImageAlphaEffect.ReleaseAndGetAddressOf()) = NULL;

else

{

D2D1Draw::CreateEffect(CLSID\_D2D1Opacity,

saveImageAlphaEffect.ReleaseAndGetAddressOf());

saveImageAlphaEffect->SetInput(0, saveImage->GetBmpRes());

saveImageAlphaEffect->SetValue(D2D1\_OPACITY\_PROP\_OPACITY, 1.0f);

}

}

}

}

/\* FileName: FSaveButton.h \*/

﻿#pragma once

#ifndef FORAGER\_FSAVEBUTTON\_H

#define FORAGER\_FSAVEBUTTON\_H

#include "CGameStdLib.h"

#include "FSaveImage.h"

#include "FImage.h"

#include "FAtlas.h"

#include "FMap.h"

#include "FOutlineText.h"

using namespace std;

namespace game\_framework

{

class FSaveButton

{

public:

FSaveButton();

~FSaveButton();

void Initialize(FSaveImage\* saveImage);

void SetTopLeft(int x, int y);

void OnShow();

void ShowFirstEffect();

void D2DCallBack();

void SetUseEffect(bool useEffect, bool deleteEffect = false);

void ShowEffect();

private:

FSaveImage\* saveImage;

FImage baseImage;

FImage maskImage;

FImage timeLogo;

FImage notice;

bool load = false;

bool useEffect = false;

bool deleteEffect = false;

bool firstEffect = true;

int callBackID, w, h, x, y;

FLOAT scale, step, degree = 0, maxDegree = 2, degreeOffset = 2;

ComPtr<ID2D1Effect> alphaEffect;

ComPtr<ID2D1Effect> saveImageAlphaEffect;

ComPtr<ID2D1Effect> maskImageAlphaEffect;

};

}

#endif // !FORAGER\_FSAVEBUTTON\_H

/\* FileName: FSaveImage.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FSaveImage.h"

using namespace std;

namespace game\_framework

{

// 讀取存檔的資料(快照圖與時間)

void FSaveImage::LoadBitmapData(unsigned char\* pBuff, int time)

{

load = true;

memcpy(this->pBuff, pBuff,

FSAVEIMAGE\_SIZE \* FSAVEIMAGE\_SIZE \* 4);

this->time = time;

D2DCallBack();

}

// 鎖定圖片不要被顯示

void FSaveImage::Lock()

{

load = false;

D2D1\_SAFERELEASE(d2dBmpPtr);

}

// 取得這個存檔的時間

int FSaveImage::GetTime()

{

return this->time;

}

// 取的 Direct2D 圖片資源實體, 給特銷輸入用

ID2D1Bitmap\* FSaveImage::GetBmpRes()

{

return this->d2dBmpPtr;

}

// 印出圖片到螢幕

void FSaveImage::ShowBitmap()

{

if(!load)

return;

if(useDestSize)

D2D1Draw::DrawBitmap(d2dBmpPtr, xDst, yDst,

FLOAT(alpha / 255.0), wDst, hDst);

else

D2D1Draw::DrawBitmap(d2dBmpPtr, xDst, yDst,

FLOAT(alpha / 255.0));

}

// 設定左上位置與圖片大小(可選)

void FSaveImage::SetTopLeft(int xDst, int yDst, int wDst,

int hDst)

{

this->xDst = xDst;

this->yDst = yDst;

this->wDst = wDst;

this->hDst = hDst;

useDestSize = !((wDst == -1) && (wDst == hDst));

}

// 設定存檔圖片透明度

void FSaveImage::SetAlpha(int alpha)

{

if(alpha > 255) alpha = 255;

if(alpha < 0) alpha = 0;

this->alpha = alpha;

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫在由 FSaveButtom Callback Function 呼叫此處進行重建資源的動做

void FSaveImage::D2DCallBack()

{

if(load)

{

D2D1\_SAFERELEASE(d2dBmpPtr);

this->d2dBmpPtr = D2D1Draw::LoadMemoryBitmap(pBuff,

FSAVEIMAGE\_SIZE,

FSAVEIMAGE\_SIZE);

}

}

// 釋放 Direct2D 資源

FSaveImage::~FSaveImage()

{

D2D1\_SAFERELEASE(this->d2dBmpPtr);

delete[] pBuff;

}

}

/\* FileName: FSaveImage.h \*/

﻿#pragma once

#ifndef FORAGER\_FSAVEIMAGE\_H

#define FORAGER\_FSAVEIMAGE\_H

#include "CGameStdLib.h"

using namespace std;

#define FSAVEIMAGE\_SIZE 252

namespace game\_framework

{

class FSaveImage

{

public:

~FSaveImage();

void LoadBitmapData(unsigned char\* pBuff, int time);

void ShowBitmap();

void SetTopLeft(int xDst, int yDst, int wDst = -1,

int hDst = -1);

void SetAlpha(int alpha = 255);

int GetTime();

void Lock();

void D2DCallBack();

ID2D1Bitmap\* GetBmpRes();

private:

int alpha = 255, xDst = 0, yDst = 0, wDst = 0, hDst = 0;

int time;

bool useDestSize = false;

bool load = false;

int callBackID;

ID2D1Bitmap\* d2dBmpPtr = NULL;

unsigned char\* pBuff = new unsigned char[FSAVEIMAGE\_SIZE\*

FSAVEIMAGE\_SIZE \* 4];

};

}

#endif // !FORAGER\_FSAVEIMAGE\_H

/\* FileName: FSaveSystem.cpp \*/

﻿#include "stdafx.h"

#include "CGameStdLib.h"

#include "FSaveSystem.h"

#include <atlimage.h>

#include <cstdio>

#include <ctime>

using namespace std;

namespace game\_framework

{

int FSaveSystem::saveID = 0;

string FSaveSystem::SAVE\_KEY0 =

"AAAAB3NzaC1yc2EAAAADAQABAAAAgQC1NGzFH2dSEvQYGdxxYS+LQvYk9K7sFC0";

string FSaveSystem::SAVE\_KEY1 =

"VnvVW051PWd2PTSm5LBEzJ4dLIB0/iMhA/V6+diSYcqd/KR3elh4/7mxUPOOgkS";

string FSaveSystem::SAVE\_KEY2 =

"LyQDeNH1AG1f0YLwN1aY2JDzNGjkHOfUsVFq1RMRABZbgcMjhRdxgb6X0gLNEWb";

bool FSaveSystem::lock = false;

FImage FSaveSystem::saveIcon;

int FSaveSystem::showCount = 0;

FSaveImage FSaveSystem::saveImage[3];

DWORD FSaveSystem::gameTime = 0;

DWORD FSaveSystem::gameCount = 0;

// 判斷存檔是否存在

bool FSaveSystem::IsExistSaveFile(int saveID)

{

fstream saveFile(".//SAVE\_" + to\_string(saveID) + ".fsave",

ios::in | ios::binary);

bool result = saveFile ? true : false;

saveFile.close();

return result;

}

// 重設存檔時間

void FSaveSystem::ResetTime()

{

gameTime = 0;

gameCount = 0;

}

// 刪除存檔

void FSaveSystem::DeleteSaveFile(int saveID)

{

if(lock) return;

fstream saveFile(".//SAVE\_" + to\_string(saveID) + ".fsave",

ios::out | ios::binary | ios\_base::trunc);

if(saveFile)

saveFile.write((char\*)"0", sizeof(1));

saveFile.close();

remove(string(".//SAVE\_" + to\_string(saveID) +

".fsave").c\_str());

}

// 檢查存檔狀況

bool FSaveSystem::CheckSave(FSaveGame& saveData)

{

string saveKey = saveData.header.saveKey;

if(saveData.header.saveID == 0)

return (saveKey == SAVE\_KEY0);

else if(saveData.header.saveID == 1)

return (saveKey == SAVE\_KEY1);

else if(saveData.header.saveID == 2)

return (saveKey == SAVE\_KEY2);

return false;

}

// 保存頭資料至存檔結構

void FSaveSystem::SaveHeader(FSaveGame& saveData)

{

saveData.header.time = FSaveSystem::gameTime + (gameCount / 30);

saveData.header.lastSaveTime = timeGetTime();

saveData.header.saveID = saveID;

FNightEffect::instance->SaveEffect(

saveData.header.nightEffectData);

string SAVE\_KEY = saveID == 0 ? SAVE\_KEY0 : (saveID == 1 ?

SAVE\_KEY1 :

(saveID == 2 ? SAVE\_KEY2 : SAVE\_KEY0));

strcpy(saveData.header.saveKey, SAVE\_KEY.c\_str());

}

// 保存地圖各種資料至存檔結構

void FSaveSystem::SaveMapData(FSaveGame& saveData,

FMap\* currentMap)

{

for(int x = 0; x < MAPRECORDSIZE; x++)

for(int y = 0; y < MAPRECORDSIZE; y++)

{

if(currentMap->m\_mapBlock[y][x].pBackImageInfo)

saveData.mapRecord[y][x].backType =

currentMap->m\_mapBlock[y][x].pBackImageInfo->type;

else

saveData.mapRecord[y][x].backType = -1;

if(currentMap->m\_mapBlock[y][x].pFrontImageInfo)

saveData.mapRecord[y][x].frontType =

currentMap->m\_mapBlock[y][x].pFrontImageInfo->type;

else

saveData.mapRecord[y][x].frontType = -1;

saveData.mapRecord[y][x].useEffect =

currentMap->m\_mapBlock[y][x].useEffect;

saveData.mapRecord[y][x].useSeaEffect =

currentMap->m\_mapBlock[y][x].useSeaEffect;

saveData.mapRecord[y][x].edgeFlag =

currentMap->m\_mapBlock[y][x].edgeFlag;

saveData.mapRecord[y][x].weedType =

currentMap->m\_mapBlock[y][x].weedType;

}

}

// 保存人物各種資料至存檔結構

void FSaveSystem::SaveHumanData(FSaveGame& saveData,

FHuman\* currentHuman)

{

saveData.humanRecord = FHumanRecord{ currentHuman->Loc.\_pX, currentHuman->Loc.\_pY };

saveData.humanToolRecord = FHumanToolRecord{ currentHuman->nowTool };

currentHuman->humanHealth.GetSaveData(

saveData.healthRecord.hunger,

saveData.healthRecord.heart, saveData.healthRecord.damage);

vector<FResource\*> bagData = \*(currentHuman->bag.GetBagData());

for(UINT i = 0; i < FBag::BAG\_MAX; i++)

{

if(i >= bagData.size())

saveData.bagRecord[i] = FBagRecord{ 0, 0 };

else

{

saveData.bagRecord[i].resourceType = bagData[i]->GetId();

saveData.bagRecord[i].count = bagData[i]->GetCount();

}

}

}

// 保存目前遊戲快照至存檔結構 (GDI 截圖)

void FSaveSystem::SaveImage(FSaveGame& saveData)

{

CDC\* pCDC = D2D1Draw::GetBackCDC();

CDC memDC;

memDC.CreateCompatibleDC(pCDC);

CBitmap saveBmp;

saveBmp.CreateCompatibleBitmap(pCDC, FSAVEIMAGE\_SIZE,

FSAVEIMAGE\_SIZE);

CBitmap\* pOldBitmap = memDC.SelectObject(&saveBmp);

memDC.SetStretchBltMode(HALFTONE);

memDC.StretchBlt(0, 0, FSAVEIMAGE\_SIZE, FSAVEIMAGE\_SIZE, pCDC,

SIZE\_X / 2 - 250,

SIZE\_Y / 2 - 250, 500, 500, SRCCOPY);

memDC.SelectObject(pOldBitmap);

HBITMAP hBmp = (HBITMAP)saveBmp;

tagBITMAP bm;

::GetObject(hBmp, sizeof(BITMAP), &bm);

tagBITMAPINFO bi;

memset(&bi, 0, sizeof(BITMAPINFO));

bi.bmiHeader = tagBITMAPINFOHEADER{ sizeof(bi.bmiHeader), bm.bmWidth, -bm.bmHeight, bm.bmPlanes, bm.bmBitsPixel, 0, 0, 0, 0, 0, 0 };

UINT pBuffLen = bm.bmWidth \* bm.bmHeight \* 4;

::GetDIBits(pCDC->m\_hDC, hBmp, 0, bm.bmHeight,

(void\*)(saveData.bitmapData.colorBuffer), &bi, DIB\_RGB\_COLORS);

for(UINT i = 0; i < pBuffLen; i += 4)

saveData.bitmapData.colorBuffer[i + 3] = 255;

saveBmp.DeleteObject();

memDC.DeleteDC();

D2D1Draw::ReleaseBackCDC();

}

// 進行存檔工作

void FSaveSystem::SaveGame(FMap\* currentMap,

FHuman\* currentHuman)

{

if(lock) return;

showCount = 30;

fstream saveFile(".//SAVE\_" + to\_string(saveID) + ".fsave",

ios::out | ios::binary | ios\_base::trunc);

if(!saveFile)

{

AfxMessageBox("存檔時發生錯誤! 存檔功能將被關閉.",

MB\_ICONEXCLAMATION);

lock = true;

}

else

{

FSaveGame saveData;

SaveHeader(saveData);

SaveMapData(saveData, currentMap);

SaveHumanData(saveData, currentHuman);

SaveImage(saveData);

saveFile.write((char\*)&saveData.header,

sizeof(saveData.header));

saveFile.write((char\*)&saveData.mapRecord,

sizeof(saveData.mapRecord));

saveFile.write((char\*)&saveData.humanRecord,

sizeof(saveData.humanRecord));

saveFile.write((char\*)&saveData.healthRecord,

sizeof(saveData.healthRecord));

saveFile.write((char\*)&saveData.humanToolRecord,

sizeof(saveData.humanToolRecord));

saveFile.write((char\*)&saveData.bagRecord,

sizeof(saveData.bagRecord));

saveFile.write((char\*)saveData.bitmapData.colorBuffer,

FSAVEIMAGE\_SIZE \* FSAVEIMAGE\_SIZE \* 4);

}

saveFile.close();

}

// 讀存檔工作

bool FSaveSystem::ReadSave(int saveID, FSaveGame& saveData)

{

bool result = false;

fstream saveFile(".//SAVE\_" + to\_string(saveID) + ".fsave",

ios::in | ios::binary);

if(saveFile)

{

saveFile.read((char\*)&saveData.header, sizeof(saveData.header));

saveFile.read((char\*)&saveData.mapRecord,

sizeof(saveData.mapRecord));

saveFile.read((char\*)&saveData.humanRecord,

sizeof(saveData.humanRecord));

saveFile.read((char\*)&saveData.healthRecord,

sizeof(saveData.healthRecord));

saveFile.read((char\*)&saveData.humanToolRecord,

sizeof(saveData.humanToolRecord));

saveFile.read((char\*)&saveData.bagRecord,

sizeof(saveData.bagRecord));

saveFile.read((char\*)saveData.bitmapData.colorBuffer,

FSAVEIMAGE\_SIZE \* FSAVEIMAGE\_SIZE \* 4);

if(CheckSave(saveData))

result = true;

saveFile.close();

}

return result;

}

// 把存檔地圖資料載入到遊戲

void FSaveSystem::LoadMapData(FSaveGame& saveData,

FMap\* currentMap)

{

for(int x = 0; x < MAPRECORDSIZE; x++)

for(int y = 0; y < MAPRECORDSIZE; y++)

{

FMapRecord& mapRecord = saveData.mapRecord[y][x];

FMapInfo::Type backType = (mapRecord.backType == -1) ?

FMapInfo::Type::NOTHING : (FMapInfo::Type)mapRecord.backType;

FMapInfo::Type frontType = (mapRecord.frontType == -1) ?

FMapInfo::Type::NOTHING : (FMapInfo::Type)mapRecord.frontType;

currentMap->m\_mapBlock[y][x] = FMapInfo{ backType, frontType };

currentMap->m\_mapBlock[y][x].useEffect = false;

currentMap->m\_mapBlock[y][x].useSeaEffect =

mapRecord.useSeaEffect;

currentMap->m\_mapBlock[y][x].edgeFlag = mapRecord.edgeFlag;

currentMap->m\_mapBlock[y][x].weedType = (FMapImage::WeedType)

mapRecord.weedType;

}

}

// 把存檔人物資料載入到遊戲

void FSaveSystem::LoadHumanData(FSaveGame& saveData,

FHuman\* currentHuman)

{

currentHuman->Loc.\_pX = saveData.humanRecord.xPoint;

currentHuman->Loc.\_pY = saveData.humanRecord.yPoint;

currentHuman->humanHealth.SetSaveData(

saveData.healthRecord.hunger,

saveData.healthRecord.heart, saveData.healthRecord.damage);

currentHuman->nowTool = saveData.humanToolRecord.toolType;

currentHuman->ChangeTool();

/\* 清空包包 \*/

vector<FResource\*> bagData = \*(currentHuman->bag.GetBagData());

for(int i = 0; i < int(bagData.size()); i++)

delete bagData[i];

currentHuman->bag.GetBagData()->clear();

/\* 還原包包 \*/

for(UINT i = 0; i < FBag::BAG\_MAX; i++)

{

if(saveData.bagRecord[i].count != 0)

{

FResource\* res = nullptr;

res = FResource::NewResource((

saveData.bagRecord[i].resourceType));

res->AddCount(saveData.bagRecord[i].count);

currentHuman->bag.GetBagData()->push\_back(res);

}

}

}

// 把存檔資料載入到遊戲

void FSaveSystem::LoadSaveGame(FMap\* currentMap,

FHuman\* currentHuman)

{

FSaveGame saveData;

if(!ReadSave(saveID, saveData))

{

AfxMessageBox("讀取存檔時發生錯誤!",

MB\_ICONEXCLAMATION);

return;

}

else if(!CheckSave(saveData))

{

AfxMessageBox("存檔已毀損, 將開始新遊戲!",

MB\_ICONEXCLAMATION);

FSaveSystem::SaveGame(currentMap, currentHuman);

return;

}

FNightEffect::instance->LoadEffect(

saveData.header.nightEffectData);

LoadMapData(saveData, currentMap);

LoadHumanData(saveData, currentHuman);

gameTime = saveData.header.time;

gameCount = 0;

}

// 讀取所有存檔圖片的快照 (主選單顯示用)

void FSaveSystem::LoadAllSaveBitmap()

{

for(int i = 0; i < 3; i++)

{

FSaveGame saveData;

if(ReadSave(i, saveData))

saveImage[i].LoadBitmapData(saveData.bitmapData.colorBuffer,

saveData.header.time);

else

saveImage[i].Lock();

}

}

// 顯示存檔圖示

void FSaveSystem::OnShow()

{

gameCount++;

static bool firstRun = true;

if(showCount <= 0)

return;

showCount--;

if(firstRun)

{

saveIcon.LoadBitmap(IDB\_SAVEICON, RGB(255, 242, 0));

firstRun = false;

}

saveIcon.SetTopLeft(SIZE\_X - 80, 30);

saveIcon.ShowBitmap();

}

// 取得遊戲時間

int FSaveSystem::GetGameTime()

{

return FSaveSystem::gameTime + (gameCount / 30);

}

}

/\* FileName: FSaveSystem.h \*/

﻿#pragma once

#ifndef FORAGER\_FSAVESYSTEM\_H

#define FORAGER\_FSAVESYSTEM\_H

#define MAPRECORDSIZE 100

#include "CGameStdLib.h"

#include "FSaveImage.h"

#include "FNightEffect.h"

#include "FMap.h"

#include <fstream>

#include <string>

using namespace std;

namespace game\_framework

{

// 定義存檔頭結構

struct FSaveHeader

{

\_\_int32 saveID;

DWORD time;

DWORD lastSaveTime;

char saveKey[64];

FLOAT nightEffectData[5];

};

// 定義存檔地圖結構

struct FMapRecord

{

\_\_int32 backType;

\_\_int32 frontType;

bool useEffect;

bool useSeaEffect;

\_\_int32 edgeFlag;

\_\_int32 weedType;

};

// 定義存檔人物位置結構

struct FHumanRecord

{

\_\_int32 xPoint;

\_\_int32 yPoint;

};

// 定義存檔人物血量結構

struct FHealthRecord

{

\_\_int32 hunger;

\_\_int32 heart;

\_\_int32 damage;

};

// 定義存檔人物工具結構

struct FHumanToolRecord

{

\_\_int32 toolType;

};

// 定義存檔背包結構

struct FBagRecord

{

\_\_int32 resourceType;

\_\_int32 count;

};

// 定義存檔快照圖片結構

struct FSaveBitmapData

{

FSaveBitmapData()

{

colorBuffer = new unsigned char[FSAVEIMAGE\_SIZE\*

FSAVEIMAGE\_SIZE \*

4]; // struct constructor

}

~FSaveBitmapData()

{

delete[] colorBuffer;

}

unsigned char\* colorBuffer;

};

// 定義存檔整體結構

struct FSaveGame

{

FSaveHeader header;

FMapRecord mapRecord[MAPRECORDSIZE][MAPRECORDSIZE];

FHumanRecord humanRecord;

FHumanToolRecord humanToolRecord;

FHealthRecord healthRecord;

FBagRecord bagRecord[FBag::BAG\_MAX];

FSaveBitmapData bitmapData;

};

// 存檔處理類別

class FSaveSystem

{

public:

static void SaveGame(FMap\* currentMap, FHuman\* currentHuman);

static void LoadSaveGame(FMap\* currentMap,

FHuman\* currentHuman);

static bool IsExistSaveFile(int saveID);

static void OnShow();

static void ResetTime();

static void DeleteSaveFile(int saveID);

static int saveID;

static void LoadAllSaveBitmap();

static int GetGameTime();

static FSaveImage saveImage[3];

private:

static bool CheckSave(FSaveGame& saveData);

static void SaveHeader(FSaveGame& saveData);

static void SaveMapData(FSaveGame& saveData, FMap\* currentMap);

static void SaveHumanData(FSaveGame& saveData,

FHuman\* currentHuman);

static void SaveImage(FSaveGame& saveData);

static bool ReadSave(int saveID, FSaveGame& saveData);

static void LoadMapData(FSaveGame& saveData, FMap\* currentMap);

static void LoadHumanData(FSaveGame& saveData,

FHuman\* currentHuman);

static bool lock;

static DWORD gameTime;

static DWORD gameCount;

static string SAVE\_KEY0;

static string SAVE\_KEY1;

static string SAVE\_KEY2;

static FImage saveIcon;

static int showCount;

};

}

#endif // !FORAGER\_FSAVESYSTEM\_H

/\* FileName: FText.cpp \*/

﻿#include "stdafx.h"

#include "FText.h"

namespace game\_framework

{

// 定義文字字形、大小、顏色、寬度、風格、拉伸與註冊 Callback Function

FText::FText(WCHAR\* fontName, FLOAT fontSize, COLORREF color,

DWRITE\_FONT\_WEIGHT fontWeight,

DWRITE\_FONT\_STYLE fontStyle, DWRITE\_FONT\_STRETCH fontStretch)

{

this->callBackID = D2D1Draw::RegisterCallBack(bind(

&FText::D2DCallBack, this));

wcscpy(this->\_fontName, fontName);

this->\_fontSize = fontSize;

this->\_color = color;

this->\_fontWeight = fontWeight;

this->\_fontStyle = fontStyle;

this->\_fontStretch = fontStretch;

this->D2DCallBack();

}

// destructor 解除註冊 Callback Function

FText::~FText()

{

D2D1Draw::UnRegisterCallBack(this->callBackID);

}

// 部分 Direct2D 資源, 在這個類別管理. 故重建資源時(時機由 D2D1Draw 類別控制), 會被 D2D1Draw 類別呼叫進行重建資源的動做

void FText::D2DCallBack()

{

pTextFormat.Reset();

brush.Reset();

D2D1Draw::CreateTextFormat(\_fontName, nullptr, \_fontWeight,

\_fontStyle,

\_fontStretch, \_fontSize, L"en-us",

pTextFormat.ReleaseAndGetAddressOf());

D2D1Draw::CreateSolidColorBrush(D2D1::ColorF(RGB(GetBValue(

\_color),

GetGValue(\_color), GetRValue(\_color))),

brush.ReleaseAndGetAddressOf());

}

// 呼叫 D2D1Draw 顯示文字

void FText::ShowText(const char\* text, int x, int y)

{

D2D1Draw::DrawDiretWriteText(text, (int)strlen(text),

pTextFormat.Get(), x, y,

brush.Get());

}

}

/\* FileName: FText.h \*/

﻿#pragma once

#ifndef FORAGER\_FTEXT\_H

#define FORAGER\_FTEXT\_H

#include "CGameStdLib.h"

namespace game\_framework

{

class FText

{

public:

FText(WCHAR\* fontName, FLOAT fontSize, COLORREF color=RGB(0, 0,

0),

DWRITE\_FONT\_WEIGHT fontWeight=DWRITE\_FONT\_WEIGHT\_REGULAR,

DWRITE\_FONT\_STYLE fontStyle=DWRITE\_FONT\_STYLE\_NORMAL,

DWRITE\_FONT\_STRETCH fontStretch=DWRITE\_FONT\_STRETCH\_NORMAL);

~FText();

void D2DCallBack();

void ShowText(const char\* text, int x, int y);

private:

ComPtr<IDWriteTextFormat> pTextFormat;

ComPtr<ID2D1SolidColorBrush> brush;

WCHAR \_fontName[256];

FLOAT \_fontSize;

COLORREF \_color = RGB(0, 0, 0);

DWRITE\_FONT\_WEIGHT \_fontWeight = DWRITE\_FONT\_WEIGHT\_REGULAR;

DWRITE\_FONT\_STYLE \_fontStyle = DWRITE\_FONT\_STYLE\_NORMAL;

DWRITE\_FONT\_STRETCH \_fontStretch = DWRITE\_FONT\_STRETCH\_NORMAL;

int callBackID;

};

}

#endif // !FORAGER\_FTEXT\_H

/\* FileName: FTool.cpp \*/

﻿#include "stdafx.h"

#include "FTool.h"

namespace game\_framework

{

//constructor 設定數值

FTool::FTool()

{

isAnimation = false;

collectionCD = 15; //採集冷卻時間

collectionClock = 0;

isCollection = false;

direction = FACERIGHT;

}

//顯示工具

bool FTool::OnShow(int sx, int sy)

{

bool isPressing = false;

static int degree = 0;

//執行動畫

if(isAnimation)

{

degree--;

if(degree == 0)

isAnimation = false;

toolNormal[FACERIGHT].SetRotation(degree\* 20.0f);

toolNormal[FACELEFT].SetRotation(360.0f - degree\* 20.0f);

}

else

{

//若採集且CD到了, 重置動畫數值

if(isCollection && (collectionClock == 0 ||

collectionClock == collectionCD))

{

isAnimation = true;

isPressing = true;

degree = 8;

}

}

if(collectionClock > 0)

collectionClock--;

toolNormal[direction].SetTopLeft(sx +

toolAnimetionDif[direction][0],

sy + toolAnimetionDif[direction][1], 80, 80);

toolNormal[direction].ShowBitmap();

return isPressing;

}

//設定是否在採集

void FTool::SetIsCollection(bool data)

{

this->isCollection = data;

}

//設定面對方向

void FTool::SetDirection(bool data)

{

this->direction = data;

}

//重置採集CD

void FTool::ResetCollectionClock()

{

this->collectionClock = this->collectionCD;

}

//回傳此工具的採集速度

double FTool::GetCollectSpeed()

{

return collectSpeed;

}

//獲得採集CD

int FTool::GetCollectionClock()

{

return this->collectionClock;

}

//十字鎬constructor 讀取圖片

FTool\_Pickaxe::FTool\_Pickaxe():FTool()

{

toolNormal[FACERIGHT].LoadBitmap(IDB\_PICKAXE\_R, RGB(255, 0, 0));

toolNormal[FACELEFT].LoadBitmap(IDB\_PICKAXE\_L, RGB(255, 0, 0));

toolAnimetionDif[FACERIGHT][0] = -40;

toolAnimetionDif[FACERIGHT][1] = -52;

toolAnimetionDif[FACELEFT][0] = -40;

toolAnimetionDif[FACELEFT][1] = -52;

collectSpeed = 1;

}

//十字鎬2constructor 讀取圖片

FTool\_Pickaxe2::FTool\_Pickaxe2()

{

toolNormal[FACERIGHT].LoadBitmap(IDB\_PICKAXE2\_R, RGB(255, 0,

0));

toolNormal[FACELEFT].LoadBitmap(IDB\_PICKAXE2\_L, RGB(255, 0, 0));

toolAnimetionDif[FACERIGHT][0] = -40;

toolAnimetionDif[FACERIGHT][1] = -52;

toolAnimetionDif[FACELEFT][0] = -40;

toolAnimetionDif[FACELEFT][1] = -52;

collectSpeed = 0.7;

}

//十字鎬3constructor 讀取圖片

FTool\_Pickaxe3::FTool\_Pickaxe3()

{

toolNormal[FACERIGHT].LoadBitmap(IDB\_PICKAXE3\_R, RGB(255, 0,

0));

toolNormal[FACELEFT].LoadBitmap(IDB\_PICKAXE3\_L, RGB(255, 0, 0));

toolAnimetionDif[FACERIGHT][0] = -40;

toolAnimetionDif[FACERIGHT][1] = -52;

toolAnimetionDif[FACELEFT][0] = -40;

toolAnimetionDif[FACELEFT][1] = -52;

collectSpeed = 0.5;

}

}

/\* FileName: FTool.h \*/

﻿#pragma once

#ifndef FORAGER\_FTOOL\_H

#define FORAGER\_FTOOL\_H

#include "CGameStdLib.h"

#include "FAnimation.h"

#include "FImage.h"

namespace game\_framework

{

class FTool

{

public:

enum DirectionMode

{

FACERIGHT,

FACELEFT,

FACEEND

};

enum ToolType

{

PICKAXE,

PICKAXE2,

PICKAXE3,

TOOLTYPEEND

};

FTool();

bool OnShow(int sx, int sy);

void SetIsCollection(bool data);

void SetDirection(bool data);

void ResetCollectionClock();

double GetCollectSpeed();

int GetCollectionClock();

protected:

int toolAnimetionDif[FACEEND][2]; //工具顯示修正

double collectSpeed;

FImage toolNormal[FACEEND];

bool isCollection;

int collectionClock;

int collectionCD;

bool isAnimation;

int direction;

};

class FTool\_Pickaxe : public FTool

{

public:

FTool\_Pickaxe();

};

class FTool\_Pickaxe2 : public FTool

{

public:

FTool\_Pickaxe2();

};

class FTool\_Pickaxe3 : public FTool

{

public:

FTool\_Pickaxe3();

};

}

#endif // !FORAGER\_FTOOL\_H

/\* FileName: mygame.h \*/

﻿#pragma once

#ifndef GF\_MYGAME\_H

#define GF\_MYGAME\_H

#include "D2D1Draw.h"

#include "Forager.h"

#endif // !GF\_MYGAME\_H

/\* FileName: XAudio2Engine.cpp \*/

﻿#include "stdafx.h"

#include "XAudio2Engine.h"

namespace game\_framework

{

XAudio2Engine XAudio2Engine::instance;

// 檢查音效錯誤

void XAudio2Engine::CheckXAudio2Fail(HRESULT hr,

const char\* errorMessage)

{

//若沒有偵測到音頻設備

if(hr == HRESULT\_FROM\_WIN32(ERROR\_NOT\_FOUND))

{

//設定為不可撥放

isPlayable = false;

//創建執行緒, 持續檢查是否有音頻設備

thread ReCheckMasterVoice = thread(

&XAudio2Engine::ReCheckMasterVoice, this);

//讓執行緒與主執行緒脫離

ReCheckMasterVoice.detach();

return;

}

if(hr != S\_OK)

{

// 輸出是什麼錯誤訊息, 並終止程式

stringstream hex;

hex << std::hex << hr;

string message = "XAudio2 error:\n\nHRESULT: 0x" + hex.str() +

"\n\nFile: " +

string(\_\_FILE\_\_)

+ "\nLine: " + to\_string(\_\_LINE\_\_) + "\n\nErrorMessage: " +

string(

errorMessage) + "\n\n";

LPTSTR errorText = NULL;

// 取得 Windows 錯誤訊息

FormatMessage(FORMAT\_MESSAGE\_FROM\_SYSTEM |

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER |

FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, hr, MAKELANGID(LANG\_ENGLISH, SUBLANG\_DEFAULT),

(LPTSTR)&errorText, 0,

NULL);

if(errorText != NULL)

{

message += "ErrorDescription: " + string(errorText);

LocalFree(errorText);

errorText = NULL;

}

AfxMessageBox(message.c\_str(), MB\_OK);

AfxGetMainWnd()->PostMessage(WM\_CLOSE);

AfxDebugBreak();

exit(1);

}

}

//找尋特定RIFF Type的指標位置

HRESULT XAudio2Engine::FindChunk(HANDLE hFile, DWORD fourcc,

DWORD& dwChunkSize,

DWORD& dwChunkDataPosition)

{

HRESULT hr = S\_OK;

if(INVALID\_SET\_FILE\_POINTER == SetFilePointer(hFile, 0, NULL,

FILE\_BEGIN))

return HRESULT\_FROM\_WIN32(GetLastError());

DWORD dwChunkType;

DWORD dwChunkDataSize;

DWORD dwRIFFDataSize = 0;

DWORD dwFileType;

DWORD bytesRead = 0;

DWORD dwOffset = 0;

while(hr == S\_OK)

{

DWORD dwRead;

if(0 == ReadFile(hFile, &dwChunkType, sizeof(DWORD), &dwRead,

NULL))

hr = HRESULT\_FROM\_WIN32(GetLastError());

if(0 == ReadFile(hFile, &dwChunkDataSize, sizeof(DWORD),

&dwRead, NULL))

hr = HRESULT\_FROM\_WIN32(GetLastError());

switch(dwChunkType)

{

case fourccRIFF:

dwRIFFDataSize = dwChunkDataSize;

dwChunkDataSize = 4;

if(0 == ReadFile(hFile, &dwFileType, sizeof(DWORD), &dwRead,

NULL))

hr = HRESULT\_FROM\_WIN32(GetLastError());

break;

default:

if(INVALID\_SET\_FILE\_POINTER == SetFilePointer(hFile,

dwChunkDataSize, NULL,

FILE\_CURRENT))

return HRESULT\_FROM\_WIN32(GetLastError());

}

dwOffset += sizeof(DWORD) \* 2;

if(dwChunkType == fourcc)

{

dwChunkSize = dwChunkDataSize;

dwChunkDataPosition = dwOffset;

return S\_OK;

}

dwOffset += dwChunkDataSize;

if(bytesRead >= dwRIFFDataSize) return S\_FALSE;

}

return hr;

}

//讀取Chunk的資料

HRESULT XAudio2Engine::ReadChunkData(HANDLE hFile, void\* buffer,

DWORD buffersize, DWORD bufferoffset)

{

HRESULT hr = S\_OK;

if(INVALID\_SET\_FILE\_POINTER == SetFilePointer(hFile,

bufferoffset, NULL,

FILE\_BEGIN))

return HRESULT\_FROM\_WIN32(GetLastError());

DWORD dwRead;

if(0 == ReadFile(hFile, buffer, buffersize, &dwRead, NULL))

hr = HRESULT\_FROM\_WIN32(GetLastError());

return hr;

}

//destructor 當呼叫這個時, 代表遊戲已經結束, 必須把所有資源釋放

XAudio2Engine::~XAudio2Engine()

{

isGameOver = true;

int exitCount = 0;

while(1)

{

if(nowThreadCount == 0)

break;

Sleep(10);

}

if(masterVoice != nullptr)

{

masterVoice->DestroyVoice();

masterVoice = nullptr;

}

if(xaudio2Device != nullptr)

xaudio2Device->StopEngine();

}

//創建XAudio2資源及設定數值

void XAudio2Engine::Initialize()

{

isGameOver = false;

isPlayable = true;

nowThreadCount = 0;

CheckXAudio2Fail(XAudio2Create(xaudio2Device.GetAddressOf()),

"XAudio2 Create device failed.");

CheckXAudio2Fail(xaudio2Device->CreateMasteringVoice(

&masterVoice),

"XAudio2 Create MasterVoice failed.");

Play("audio\\ForagerBGM.wav", true);

}

//開啟一個執行緒播放音樂

void XAudio2Engine::Play(const char\* path, bool isLoop,

float Volume)

{

//判斷是否可撥放

if(isPlayable)

{

//創建執行緒

thread audioThread = thread(&XAudio2Engine::ReadAndSubmitAudio,

this, path,

Volume, isLoop);

//讓執行緒與主執行緒脫離

audioThread.detach();

}

}

//設定音量

void XAudio2Engine::SetAudioVolume(float Volume)

{

CheckXAudio2Fail(masterVoice->SetVolume(Volume),

"XAudio2 MasterVoice set volume failed.");

}

//判斷目前是否可以播放音樂

bool XAudio2Engine::IsPlayable()

{

return isPlayable;

}

//獲得現在音量

void XAudio2Engine::GetAudioVolume(float\* volumePtr)

{

masterVoice->GetVolume(volumePtr);

}

//讀取音樂資料並送至MasteringVoice撥放(使用流式讀取)

void XAudio2Engine::ReadAndSubmitAudio(const char\* path,

const float Volume,

const bool isLoop)

{

nowThreadCount++;

IXAudio2SourceVoice\* pSourceVoice;

WAVEFORMATEXTENSIBLE wfx;

VoiceCallback voiceCallback;

HANDLE hFile = CreateFile(

path,

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL,

OPEN\_EXISTING,

0,

NULL);

CheckXAudio2Fail((INVALID\_HANDLE\_VALUE == hFile ||

INVALID\_SET\_FILE\_POINTER == SetFilePointer(hFile, 0, NULL,

FILE\_BEGIN)) ? S\_FALSE : S\_OK, "hFile");

DWORD dwChunkSize;

DWORD dwChunkPosition;

//尋找RIFF Type的Chunk指標並讀取資料

CheckXAudio2Fail(FindChunk(hFile, fourccRIFF, dwChunkSize,

dwChunkPosition),

"XAudio2 FindChunk(RIFF) failed.");

DWORD filetype;

CheckXAudio2Fail(ReadChunkData(hFile, &filetype, sizeof(DWORD),

dwChunkPosition), "XAudio2 ReadChunk(RIFF) failed.");

//判斷檔案是否為WAV檔

CheckXAudio2Fail(filetype == fourccWAVE ? S\_OK : S\_FALSE,

"XAudio2 is not a wav file");

//尋找FMT Type的Chunk指標並讀取資料

CheckXAudio2Fail(FindChunk(hFile, fourccFMT, dwChunkSize,

dwChunkPosition),

"XAudio2 FindChunk(FMT) failed.");

CheckXAudio2Fail(ReadChunkData(hFile, &(wfx), dwChunkSize,

dwChunkPosition),

"XAudio2 ReadChunk(FMT) failed.");

//尋找DATA Type的Chunk指標

CheckXAudio2Fail(FindChunk(hFile, fourccDATA, dwChunkSize,

dwChunkPosition),

"XAudio2 FindChunk(DATA) failed.");

CheckXAudio2Fail(xaudio2Device->CreateSourceVoice(&

(pSourceVoice),

(WAVEFORMATEX\*)&(wfx), 0, XAUDIO2\_DEFAULT\_FREQ\_RATIO,

&voiceCallback), "XAudio2 Create SourceVoice failed.");

//SourceVoice開始播放緩衝區音樂

pSourceVoice->Start(0, 0);

//若需要循環撥放, 此迴圈會無限循環

while(true)

{

pSourceVoice->FlushSourceBuffers();

//去雜音, 先把聲音設定0, 因為音樂剛開始會有雜音, 等撥放一個緩衝區的時間, 再設定音量

CheckXAudio2Fail(pSourceVoice->SetVolume(0.0f, 0),

"XAudio2 Set volume failed");

//初始化數值

BYTE buffers[MAX\_BUFFER\_COUNT][STREAMING\_BUFFER\_SIZE];

OVERLAPPED Overlapped = { 0 };

Overlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

DWORD CurrentDiskReadBuffer = 0;

DWORD CurrentPosition = 0;

DWORD dwRead;

XAUDIO2\_VOICE\_STATE state;

bool isFirstBuf = true;

//若目前指標位置還沒到達檔案的最後面

while(CurrentPosition < dwChunkSize && !isGameOver)

{

DWORD readDataSize = min(STREAMING\_BUFFER\_SIZE,

dwChunkSize - CurrentPosition);

//讀取大小為STREAMING\_BUFFER\_SIZE的資料至Buffer(異步讀取)

CheckXAudio2Fail(ReadFile(hFile, buffers[CurrentDiskReadBuffer],

STREAMING\_BUFFER\_SIZE, &dwRead, &Overlapped) ? S\_OK : S\_FALSE,

"XAudio2 Read file to buffer failed.");

//因檔案讀取放式使用異步讀取, 所以必須等待檔案讀取完畢

DWORD NumberBytesTransferred;

::GetOverlappedResult(hFile, &Overlapped,

&NumberBytesTransferred, TRUE);

//改變讀取位置指標

Overlapped.Offset += readDataSize;

CurrentPosition += readDataSize;

//限制緩衝區最大數為MAX\_BUFFER\_COUNT

int waitCount = 0;

while(pSourceVoice->GetState(&state), !isGameOver &&

state.BuffersQueued >= MAX\_BUFFER\_COUNT - 1)

{

WaitForSingleObject(voiceCallback.hBufferEndEvent, 1);

//若等太久, 代表緩衝區沒有撥放,

if(++waitCount == 300)

{

Mmutex.lock();

CloseHandle(hFile);

pSourceVoice->DestroyVoice();

if(isPlayable == true)

{

isPlayable = false;

thread ReCheckMasterVoice = thread(

&XAudio2Engine::ReCheckMasterVoice, this);

ReCheckMasterVoice.detach();

}

nowThreadCount--;

Mmutex.unlock();

return;

}

}

//設定Buffer資料

XAUDIO2\_BUFFER buf = { 0 };

buf.AudioBytes = readDataSize;

buf.pAudioData = buffers[CurrentDiskReadBuffer];

//若指標已到達檔案最後

if(CurrentPosition >= dwChunkSize)

buf.Flags = XAUDIO2\_END\_OF\_STREAM;

//推送Buffer資料到緩衝區

pSourceVoice->SubmitSourceBuffer(&buf);

CurrentDiskReadBuffer++;

CurrentDiskReadBuffer %= MAX\_BUFFER\_COUNT;

//若為第一次推送, 則Delay0.01秒, 再設定聲音(去雜音)

if(isFirstBuf)

{

Sleep(10);

pSourceVoice->SetVolume(Volume, 0);

isFirstBuf = false;

}

}

//若沒有要循環且遊戲還沒結束, 重新RUN一遍迴圈

if(isLoop && !isGameOver)

continue;

else

break;

}

//若檔案太小, 推送完BUFFER就會直接結束, BUFFER的資料會因此消失, 造成音效沒有播放完的情形, 所以讓他Delay一下再關掉

if(!isGameOver)

Sleep(80);

//釋放資源

CloseHandle(hFile);

pSourceVoice->DestroyVoice();

nowThreadCount--;

}

//持續檢查是否有音頻設備連接

void XAudio2Engine::ReCheckMasterVoice()

{

CoInitialize(NULL);

nowThreadCount++;

if(masterVoice != nullptr)

{

masterVoice->DestroyVoice();

masterVoice = nullptr;

}

while(!isGameOver &&

xaudio2Device->CreateMasteringVoice(&masterVoice) != S\_OK)

Sleep(10);

if(!isGameOver)

isPlayable = true;

Play("audio\\ForagerBGM.wav", true);

nowThreadCount--;

CoUninitialize();

}

}

/\* FileName: XAudio2Engine.h \*/

﻿#pragma once

#ifndef FORAGER\_FAUDIOENGINE\_H

#define FORAGER\_FAUDIOENGINE\_H

#include "D2D1Draw.h"

#pragma comment(lib, "xaudio2")

#include <xaudio2.h>

#include <wrl.h>

#include <string>

#include <vector>

#include <iostream>

#include <thread>

#include <mutex>

#define STREAMING\_BUFFER\_SIZE 65536

#define MAX\_BUFFER\_COUNT 3

#define fourccRIFF 'FFIR'

#define fourccDATA 'atad'

#define fourccFMT ' tmf'

#define fourccWAVE 'EVAW'

#define fourccXWMA 'AMWX'

#define fourccDPDS 'sdpd'

using namespace std;

using namespace Microsoft::WRL;

namespace game\_framework

{

class XAudio2Engine

{

public:

~XAudio2Engine();

void Initialize();

void Play(const char\* path, bool isLoop = false,

float Volume = 1.0f);

void SetAudioVolume(float Volume);

bool IsPlayable();

void GetAudioVolume(float\* volumePtr);

static XAudio2Engine instance;

private:

void ReadAndSubmitAudio(const char\* path, const float Volume,

const bool isLoop);

void ReCheckMasterVoice();

mutex Mmutex;

HRESULT FindChunk(HANDLE hFile, DWORD fourcc,

DWORD& dwChunkSize,

DWORD& dwChunkDataPosition);

HRESULT ReadChunkData(HANDLE hFile, void\* buffer,

DWORD buffersize,

DWORD bufferoffset);

ComPtr<IXAudio2> xaudio2Device;

IXAudio2MasteringVoice\* masterVoice;

bool isGameOver;

int nowThreadCount;

bool isPlayable;

void CheckXAudio2Fail(HRESULT hr, const char\* errorMessage);

};

struct VoiceCallback : public IXAudio2VoiceCallback

{

HANDLE hBufferEndEvent;

STDMETHOD\_(void, OnVoiceProcessingPassStart)(UINT32) override {}

STDMETHOD\_(void, OnVoiceProcessingPassEnd)() override {}

STDMETHOD\_(void, OnStreamEnd)() override {}

STDMETHOD\_(void, OnBufferStart)(void\*) override {}

STDMETHOD\_(void, OnVoiceError)(void\*, HRESULT) override {}

STDMETHOD\_(void, OnLoopEnd)(void\*) override {}

STDMETHOD\_(void, OnBufferEnd)(void\*) override

{

SetEvent(hBufferEndEvent);

}

VoiceCallback() : hBufferEndEvent(CreateEventEx(nullptr,

nullptr, 0,

EVENT\_MODIFY\_STATE | SYNCHRONIZE))

{

}

virtual ~VoiceCallback()

{

CloseHandle(hBufferEndEvent);

}

};

}

#endif // !FORAGER\_FAUDIOENGINE\_H