台

北

科

大

資

工

系

2

0

2

0

年

物

件

導

向

程

式

設

計

實

習

報

告

第

16

組

題

目

Candy Crush Saga

**組別 : 第16組**

**題目 : Candy Crush Saga**

**組員 : 107590050 王偉斌**

**107590058 陳小蘭**

**指導老師 : 陳偉凱老師**









**目錄**

1. 簡介

1. 動機：說明選擇這個遊戲的動機 3

2. 分工：說明同組兩人的分工方式 3

1. 遊戲介紹

1. 遊戲說明 4

2. 遊戲圖形 6

3. 遊戲音效 10

1. 程式設計

1. 程式架構 12

2. 程式類別 13

3. 程式技術 14

1. 結語

1. 問題及解決方法 16

2. 時間表 17

3. 貢獻比例 19

4. 自我檢核表 20

5. 收獲 21

6. 心得、感想 22附錄

**簡介**

1）**動機**：

全球類似於寶石方塊的游戲可説是相當的多，但糖果傳奇(Candy Crush Saga) 這款游戲有種神奇的魔力，他用鮮艷的顔色、精緻的特效，剛推出一個月就吸引了超過一千萬名用戶下載。

考慮到我們兩位組員的生活情況，平時比較少時間接觸故事性、RPG或者是玩法複雜需要很多時間來玩的游戲。這款游戲比起很多流行的小游戲，這款游戲基本玩法的演算相較簡單，實作出來所需的時間較爲彈性，可以根據我們擁有的時間來決定擴充多少種游戲玩法和特效。

2）**分工**：

王偉斌 - 主要編寫游戲區域的陣列運算，糖果的掉落，爆炸/爆漿特效，音效處理。規劃有戲製作進度。

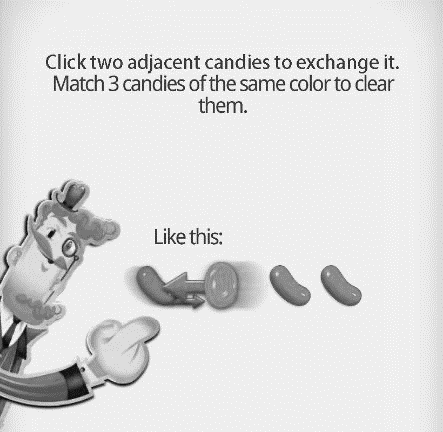
負責物件：GameArea, Candy, Blast(NormalBlast, LineBlast, SuperBlast, MagicBlast), ScoreBoard。

陳小蘭 – 主要負責排版、設計，編寫讀檔存檔，規劃游戲設定，鏈接游戲狀態之間的切換，游戲關卡設計。

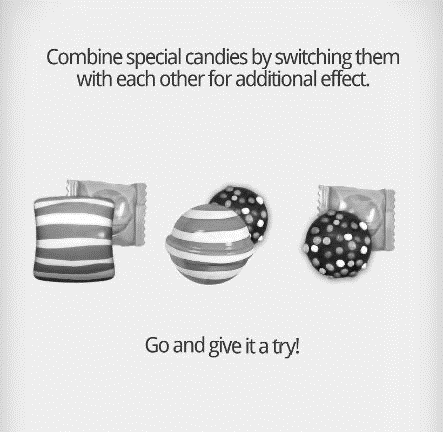
負責物件：CGameState (CGameStateInit, CGameStateRun, CGameStateMenu, CGameStateOver), Stage。

**遊戲介紹**

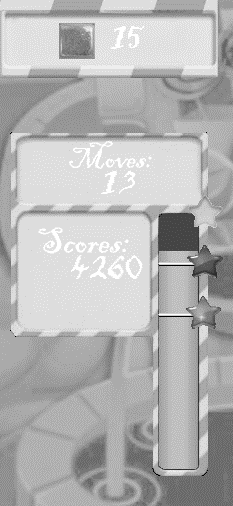
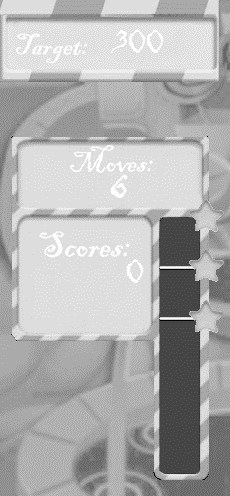
1. **遊戲說明：**
   1. 點擊兩個相鄰的糖果可以進行交換，當3顆相同顔色的糖果連成一綫時會被消除。



* 1. 以不同的形式配對4~5顆糖果可以產生出特別（强化）的糖果
  2. 兩顆特別（强化）的糖果互相交換會有特別的爆炸效果



* 1. 進入游戲後，左邊的計分板(Score Board)上會顯示當前的游戲模式、剩餘步數、當前纍計分數
     1. Target 分數挑戰 ： 當纍計分數達到目標便會自動通關，第一次游戲的目標為第1顆星，≥第二次游戲的目標為2星/3星/歷史最高分。
     2. 果凍挑戰：在指定位置消除糖果可以破壞該位置的果凍，消除場上所有果凍即可通關。



* 1. 秘技/作弊：
     1. 選關界面：長按鍵盤上的F1不放，點擊任何未解鎖關卡可以直接進入游戲
     2. 游戲界面：長安鍵盤上的F1不放，點擊場上任何糖果可以切換强化效果

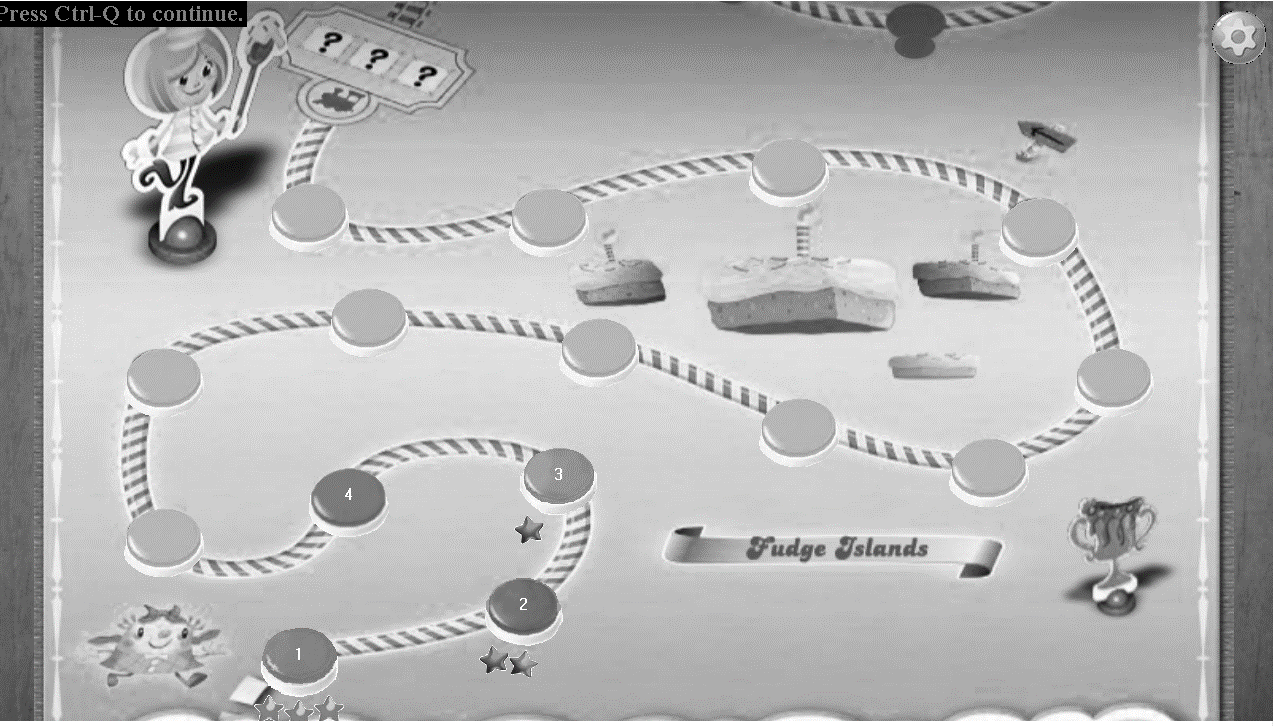
1. **遊戲圖形**

**遊戲畫面**

開啓游戲時的載入畫面



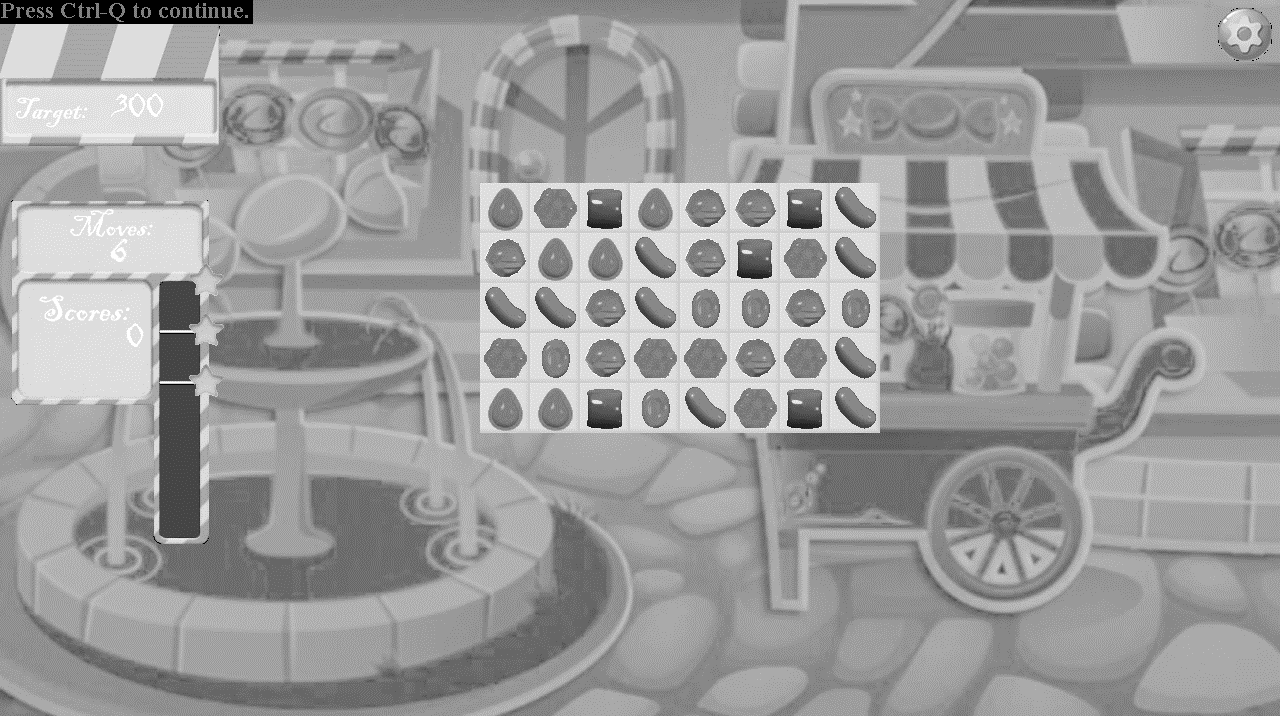
游戲載入完后的畫面

點擊Play按鈕可以切換到選關界面

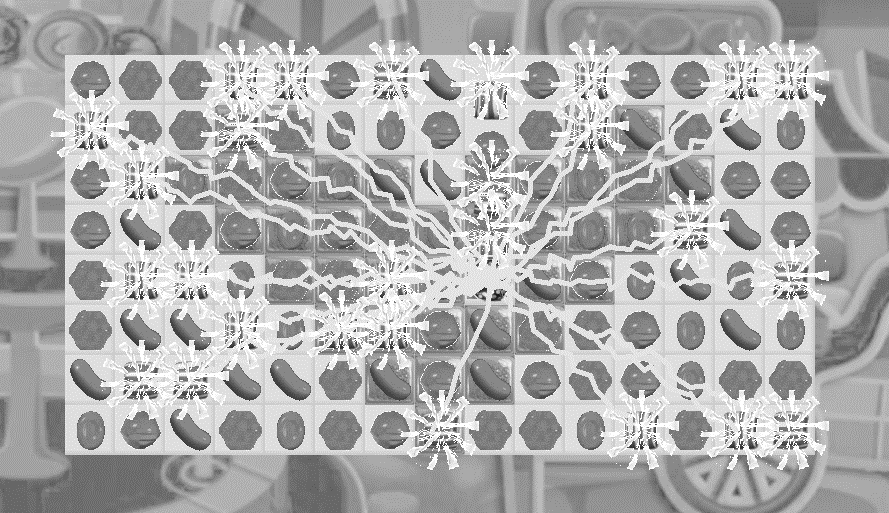
選關後的載入畫面



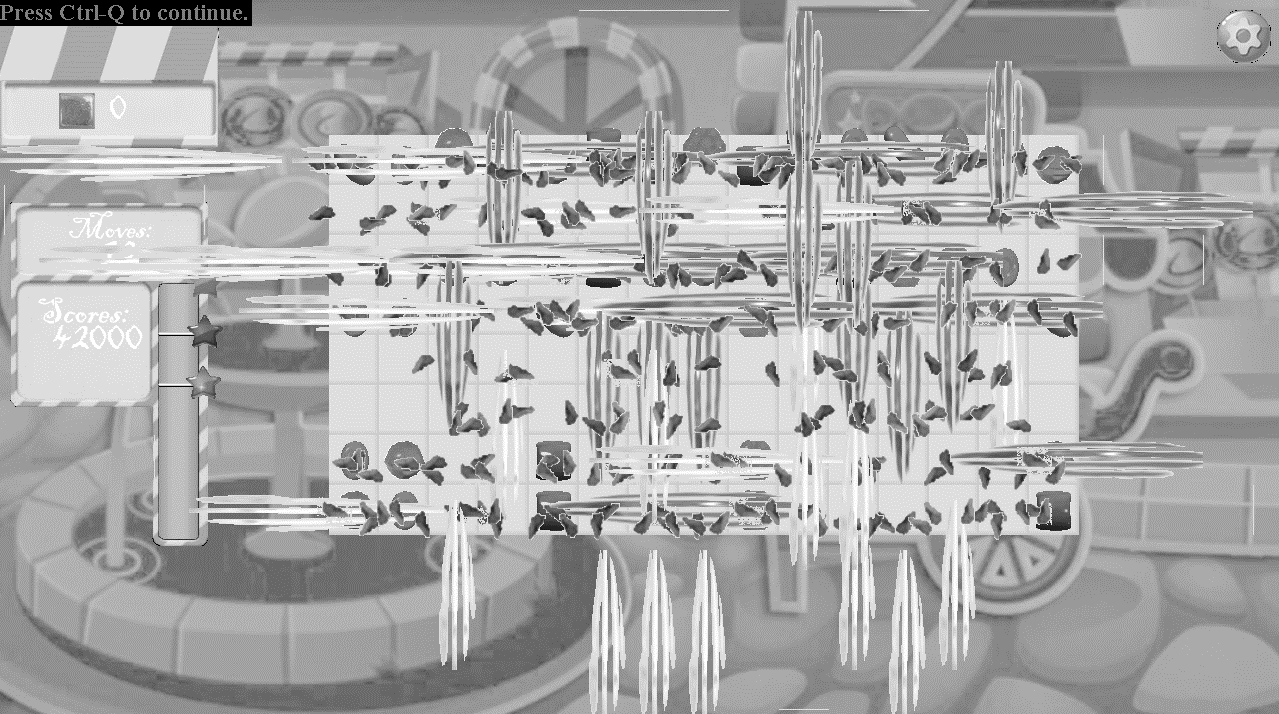
游戲畫面



SuperBlast效果 （巧克力糖與綫條糖果交換）

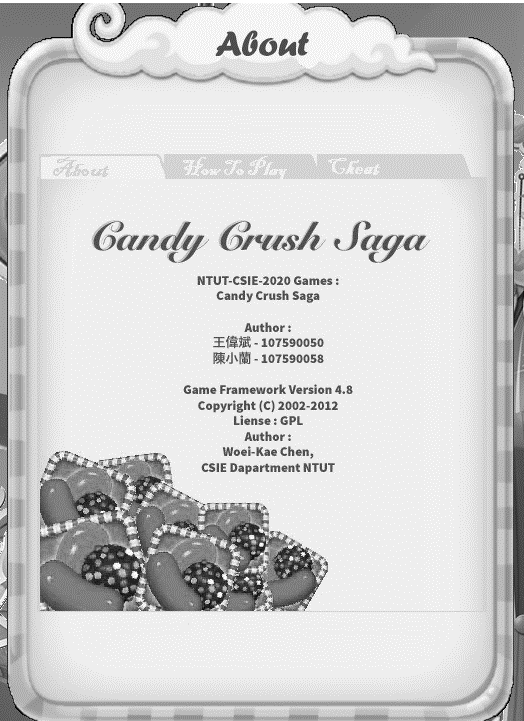


LineBlast效果



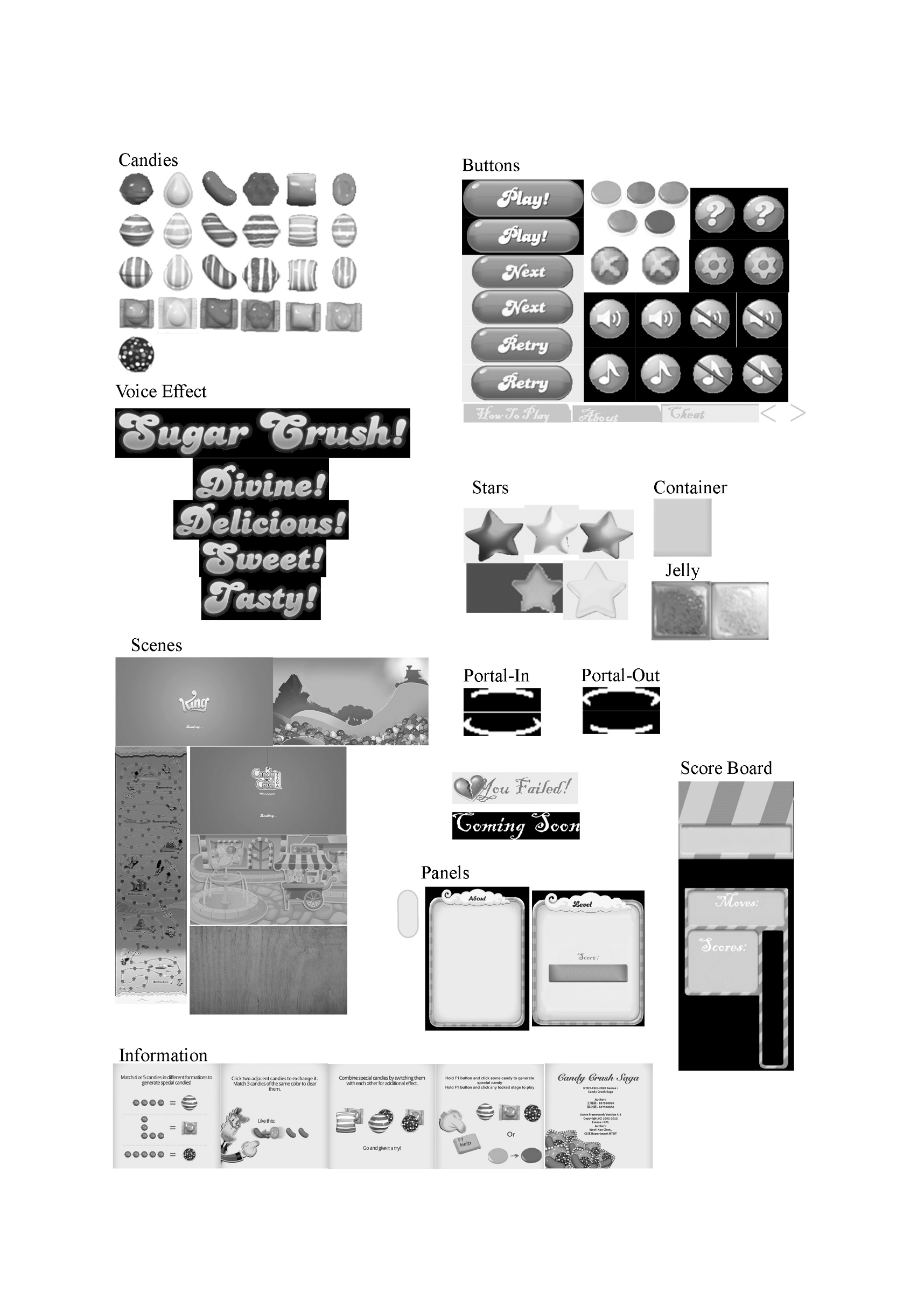
成功通關畫面

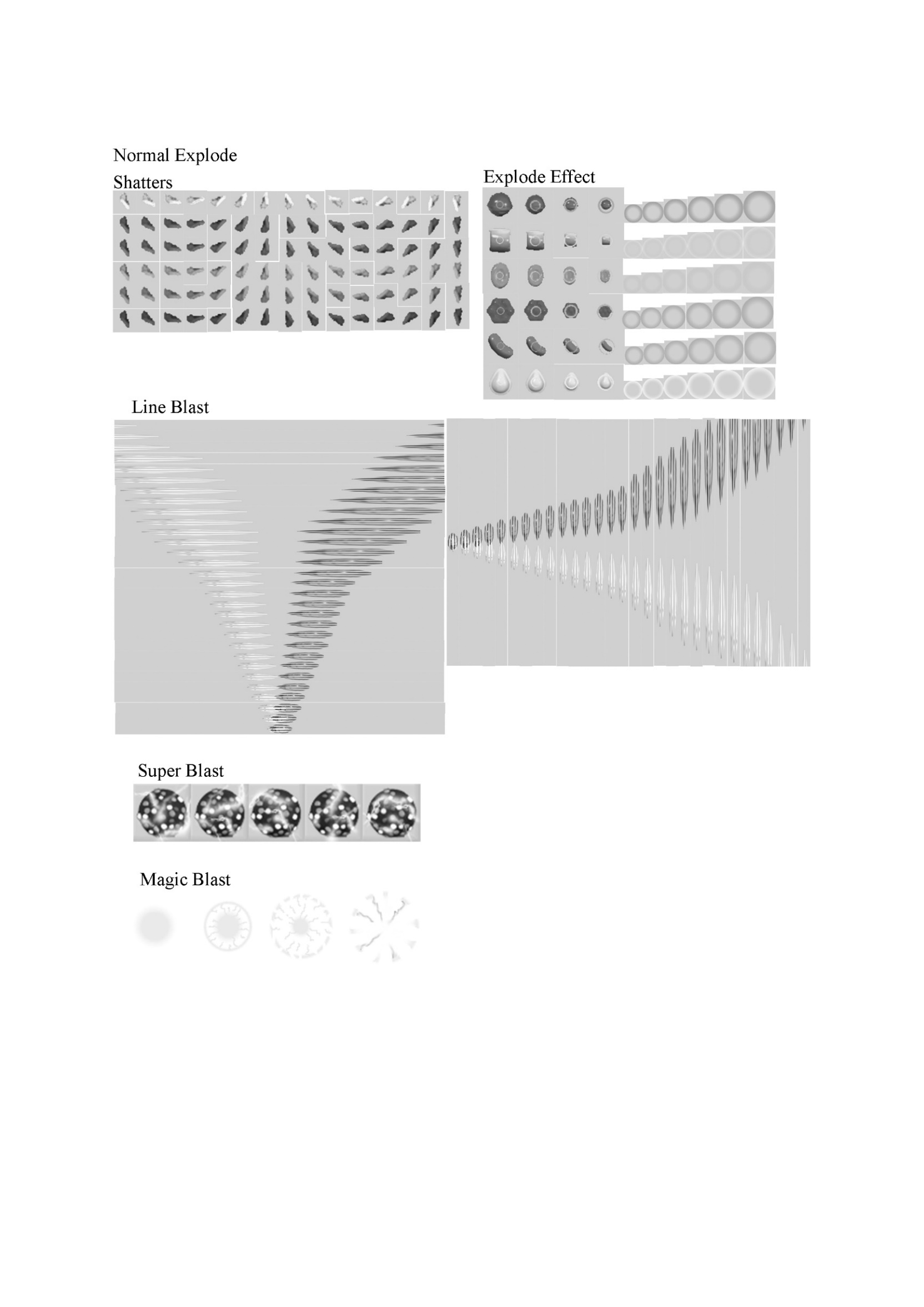


在初始界面點擊問號按鈕會出現此界面

通關失敗畫面



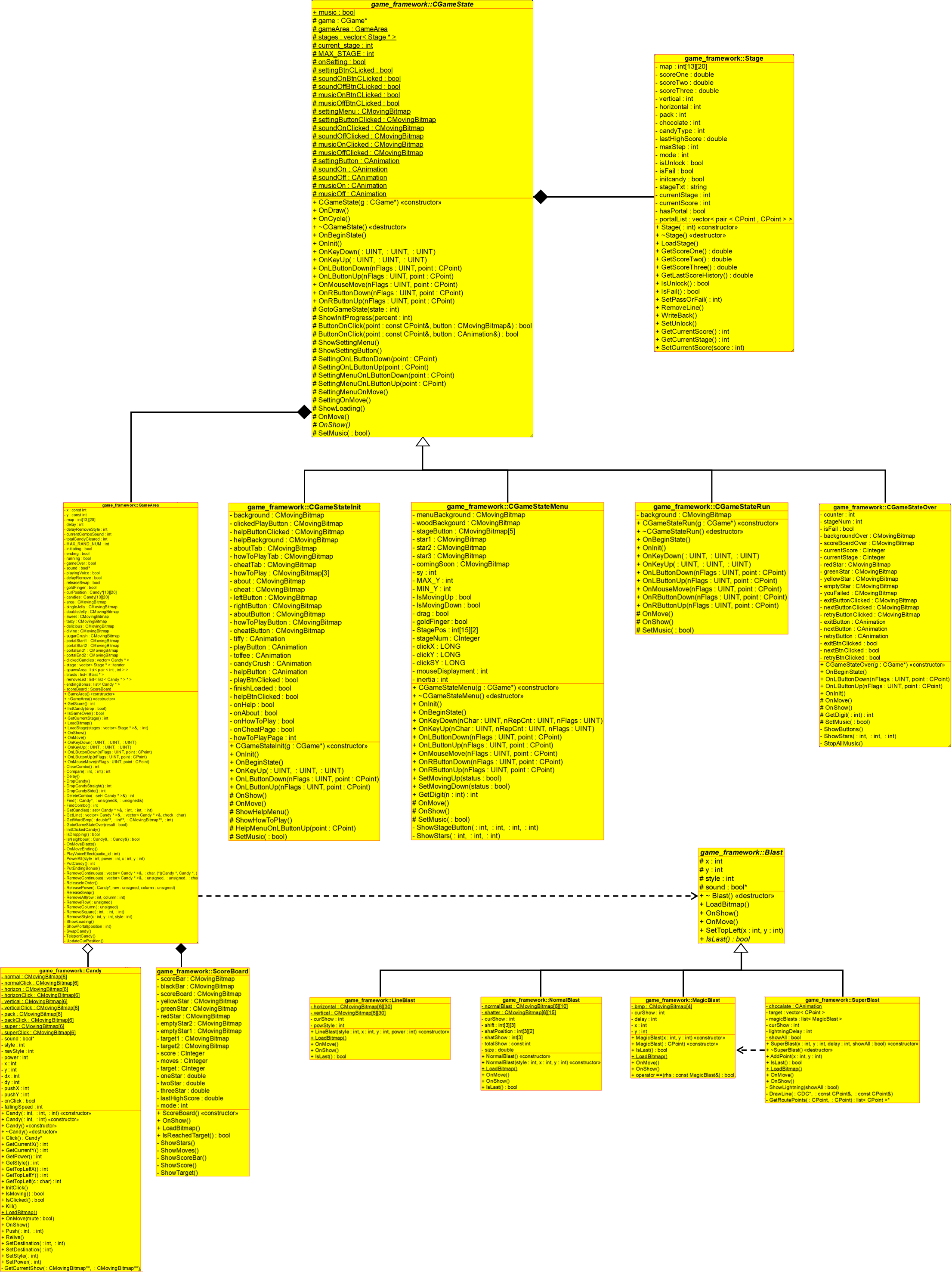
**游戲所用圖形**



**游戲音效**

|  |  |
| --- | --- |
| AUDIO\_ID | 説明 |
| AUDIO\_JELLY | CGameStateRun 背景音樂 |
| AUDIO\_STAGE | CGameStateMenu背景音樂 |
| AUDIO\_NEG\_SWAP | 糖果交換後沒有消除的音效 |
| AUDIO\_SWAP | 糖果交換音效 |
| AUDIO\_SUPER\_CREATE | 產生巧克力糖的音效 |
| AUDIO\_LINE\_CREATE | 產生綫條糖果的音效 |
| AUDIO\_PACK\_CREATE | 產生包裝糖果的音效 |
| AUDIO\_POWER\_ALL | 巧克力糖與非巧克力糖的强化糖果交換後的音效 |
| AUDIO\_SQUARE\_REMOVE1 | 觸發包裝糖果的音效 |
| AUDIO\_SQUARE\_REMOVE2 | 兩個包裝糖果交換後觸發的音效 |
| AUDIO\_LINE\_BLAST | 觸發綫條糖果的音效 |
| AUDIO\_SUPER\_REMOVE | 巧克力糖與普通糖果交換後的音效 |
| AUDIO\_CANDY\_LAND1, AUDIO\_CANDY\_LAND2, AUDIO\_CANDY\_LAND3, AUDIO\_CANDY\_LAND4, AUDIO\_CANDY\_LAND5, AUDIO\_CANDY\_LAND6, AUDIO\_CANDY\_LAND7, AUDIO\_CANDY\_LAND8, AUDIO\_CANDY\_LAND9, AUDIO\_CANDY\_LAND10, AUDIO\_CANDY\_LAND11, AUDIO\_CANDY\_LAND12 | 連續消除糖果(Combo)的音效 |
| AUDIO\_SWEET, AUDIO\_TASTY, AUDIO\_DELICIOUS, AUDIO\_DIVINE, AUDIO\_SUGAR\_CRUSH | Voice Effect 的音效 |
| AUDIO\_LEVEL\_FAIL, AUDIO\_LEVEL\_COMPLETE | CGameStateOver的背景音樂 |
| AUDIO\_STATE\_FAIL, AUDIO\_STATE\_COMPLETE | 通關成功/失敗的音效 |
| AUDIO\_BTN\_CLICK, AUDIO\_BTN\_RELEASE | 點擊按鈕的音效 |

1. **程式設計**
   1. 程式架構：



* 1. 程式類別

|  |  |  |  |
| --- | --- | --- | --- |
| 類別名稱 | .h檔行數 | .cpp檔行數 | 説明 |
| Blast | 27 | 137 | 爆炸動畫 |
| Candy | 51 | 259 | 糖果的樣式，移動與數據（移動速度，位置等） |
| CGameStateMenu | 38 | 285 | 選關界面的顯示與控制 |
| GameArea | 94 | 1206 | 游戲的控制，陣列運算，控制特效 |
| LineBlast | 13 | 49 | 綫條糖果的爆炸動畫 |
| MagicBlast | 15 | 43 | 糖果被SuperBlast擊中的動畫 |
| NormalBlast | 18 | 83 | 每一顆糖果的消除動畫 |
| ScoreBoard | 32 | 143 | 呈現分數版，記錄份數，判斷是否達到游戲目標 |
| Stage | 46 | 205 | 讀取、記錄、寫入游戲數據 |
| SuperBlast | 21 | 148 | 巧克力糖的爆炸動畫 |
| 縂行數 | 355 | 2558 |  |

* 1. 程式技術：

資料結構

* + 1. 二維陣列
       1. Container Map (int)：

此陣列儲存果凍位置、生成區域和糖果可以顯示及移動的區塊，防止糖果移動溢位

* + - 1. Candies (Candy)：

此陣列與Container Map相似，供程式碼用於控制所有糖果的移動，記錄糖果的數據

* + - 1. curPosition(Candy\*)：

每一刻糖果當前的位置，因程式運算的速度與畫面的呈現速度不同，為避免其互相干擾，並且更有效的控制糖果，所以我們利用另一個陣列保存糖果的當前位置。

* + 1. 伫列
       1. 每一種糖果的消除/爆炸特效都會保存在佇列中，再以先進先出的方式消除/顯示

游戲運作方式

開啓游戲的時候程式會先讀取全部游戲檔案然後將其保存到Stage陣列中，正常關閉游戲的時候Stage會將所有游戲數據重新保存會檔案中。

進入游戲後游戲狀態細分為4個狀態：

1. 初始化狀態：
   1. 進入游戲的時候糖果的呈現方式有兩種：
2. 全部靜止：載入的時候程式會預先讓糖果掉落、消除且不算分數、不顯示動畫、不播放聲音，當全部糖果靜止後才算載入完成，顯示游戲畫面。
3. 從指定生成區掉落：糖果會與生成區產生，然後優先直綫掉落，當當前的行陣列填滿後，如果左右的區域爲空，且其最頂端沒有生成區，當過會掉落到側邊。
4. 可操作狀態：玩家可自由操作，交換、消除糖果、纍計分數直到達標/達到步數上限/退出游戲
5. 結尾狀態：
   1. 玩家達到通關條件：剩餘步數將轉換爲獎勵點數，將場上的糖果隨機變成綫條糖果，引爆場上所有强化糖果直到沒有强化糖果且靜止後結束游戲。
   2. 未達到通關條件：延遲數秒後結束游戲
6. 游戲結束狀態：切換到CGameStateOver。

我們實作出的游戲模式有兩種：

1. 目標分數挑戰 ： 當纍計分數達到目標便會自動通關，第一次游戲的目標為第1顆星，≥第二次游戲的目標為2星/3星/歷史最高分。
2. 果凍挑戰：游戲初始化時會自動計算場上剩餘果凍的數量

游戲檢測可消除糖果的方式是依序從糖果陣列中將相鄰且形態相同的糖果收集到陣列中，將這些糖果分類成行與列，再進行分析，最後得出是否消除糖果，是否產生强化糖果。

巧克力糖的爆炸特效(SuperBlast)是以CDC繪製，其它則是Bitmap動畫

游戲結束後，GameArea會將ScoreBoard的部分數據暫存到Stage以供CGameStateOver讀取。

1. **結語**

**問題及解決方法**

|  |  |
| --- | --- |
| 問題 | 解決方法 |
| 剛開始的時候對老師的gameframe很陌生，以爲那只是參考架構，不能使用 | 仔細研究gameframe中各種功能及預先寫好的架構 |
| 程式碼的Code Page全部都是以Big-5撰寫，每次整理過後都會出現亂碼 | 配合vs code將全部有中文注記的程式碼轉換爲UTF-8 |
| 游戲的執行非常非常卡，尤其是播放糖果消除/爆炸效果的時候 | 在部分程式碼及逥圈使用多綫程運算。修改LoadBitmap的方法 |
| 專案只支援32位元系統，編譯速度較爲緩慢 | 將專案調整設定為支援64位元系統 |
| CInteger作爲整數卻在運算上難以使用，寬度/位數固定 | 添加operator overloading，修改初始化方式，寬度/位數會隨著各種運算/初始化自動調整 |
| 程式運算的速度與畫面的呈現速度不同，導致畫面亂七八糟 | 抓取當前顯示的狀態，再用此狀態進行運算，讓程式與畫面同步 |
| 在程式碼中直接使用路徑LoadBitmap的次數達到約7000次時程式會直接當掉，任何Bitmap都沒辦法Load | 將讀取路徑寫在game.rc，利用Bitmap ID LoadBitmap |
| 游戲安裝在電腦上無法讀取游戲檔案 | 讓使用者執行游戲前獲取管理員權限 |
| 搭檔寫程式沒有標準，缺乏可讀性，實作出的功能不進行測試直接發佈 | 不停嘮嘮叨叨 |

* 1. **時間表（不含上課時間)：**

|  |  |  |  |
| --- | --- | --- | --- |
| 周次 | 王偉斌（小時） | 陳小蘭（小時） | 説明 |
| 1 | 0 | 0 | 第一周星期一沒上課 |
| 2 | 6 | 2 | 瞭解游戲架構，完成游戲初始畫面 |
| 3 | 10 | 5 | 完成Play按鈕，添加游戲背景，加入糖果，將所有含中文字的程式碼重新編碼為UTF-8，進入游戲後可隨機產生糖果，加入檔案讀取 |
| 4 | 14 | 4 | 調整讀檔，暫停按鈕改爲p，糖果可以掉落、消除，加入Stage類別 |
| 5 | 19 | 8 | 加入計分板，糖果可以進行交換，可以產生全部種類的强化糖果，觸發强化糖果能夠消除相對應範圍的糖果，加入糖果點擊的視覺回饋，改善糖果掉落，更換游戲圖標 |
| 6 | 27 | 10 | 刪除無用程式碼，改進Stage類別，避開直接通過路徑load太多bitmap會當掉的Bug，進入游戲時糖果有兩種呈現方式，糖果向側邊掉落，加入選關界面，加入包裝糖果爆炸後周圍糖果的位移效果，加入音效 |
| 7 | 10 | 5 | 增加選關界面按鈕、背景音樂，新增黃色糖果，更新計分板，開始加入爆炸特效、動畫 |
| 8 | 18 | 5 | 加入一般爆炸特效，更新選關系統，游戲加入步數限制，計分板加入星星 |
| 9 | 12 | 7 | 一般特效：糖果爆炸會產生碎片，更新選關系統，加入Tiffy，改良CInteger |
| 10 | 9 | 2 | 剛進入靜態模式的關卡不再有已產生的强化糖果、被破壞的果凍、糖果為全部靜止狀態，新增類別ScoreBoard，計分板上會根據游戲模式顯示目標，新增載入畫面，更新巧克力糖爆炸效果，加入更多音效，更新選關界面 |
| 11 | 31 | 16 | 更新巧克力糖爆炸效果，選關界面可以直接利用滑鼠拖移，加入按鈕動畫，改良選關系統，加入新數字字型，加入綫條糖果爆炸效果，更新選關界面，加入Toffee，專案支援64位元系統，解決安裝游戲無法讀檔問題，初始畫面加入CandyCrush logo，新增游戲結尾狀態，加入金手指（作弊），加入游戲結束狀態 |
| 12 | 4 | 1 | 更新游戲結束、初始畫面 |
| 13 | 14 | 8.5 | 更新游戲初始畫面，加入新數字字型，更新游戲結束畫面，加入多綫程運算，解決游戲卡頓問題，調整結束游戲條件，更新巧克力糖爆炸效果 |
| 14 | 8 | 1.5 | 解決游戲數據錯誤問題，更新選關界面，改良綫條糖果爆炸特效，加入MagicBlast, 更新巧克力糖爆炸效果 |
| 15 | 12 | 8 | 加入音樂與音效設定功能，加入About, How To Play, Cheat説明，加入糖果傳送效果，加入Voice Effect |

|  |  |  |  |
| --- | --- | --- | --- |
| 16 | 24 | 5 | 修復音樂與音效設定功能，全屏幕啓動，更新暫停畫面，更新About畫面，加入傳送門，加入按鈕音效，重新設計關卡，寫報告 |
| 17 | 0 | 0 | 交報告 |
| 合計 | 218 | 88 |  |

* 1. **貢獻比例**

王偉斌 - 50%

陳小蘭 - 50%

* 1. **自我檢核表**

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

* 1. **收穫**
     1. 王偉斌：
        1. 這學期學會運用MFC、gameframe來寫視窗游戲，不像過往都是從0開始寫一些小程式，運用別人寫好的框架需要先去好好摸索與適應，才能開始構想自己的程式。
        2. 實際運用到作業中不常見的讀寫檔、靜態變數、enum以及全域變數
        3. 讓使用者可以運用鍵盤滑鼠操控游戲
        4. 靈活運用Debugger以及游戲輸出尋找錯誤
     2. 陳小蘭：
  2. **心得、感想**
     1. 王偉斌：

這算是我第一次與別人協作寫較大的程式，一次過將大一及大二上學期學過的技巧、知識都一并運用上。

與他人合作最大的困難就是維持程式的效率、可讀性以及程式運行邏輯的統一，專題實作不像計算機程式設計的作業，不能只求通過老師網站上的全部測資，大型程式的漏洞層出不窮，必須兼顧程式的可讀性和可擴充性，如果我們經常忽略一些細節和小錯誤，未來很可能會因爲這些層層叠加的錯誤形成無法修復的錯誤，所以遇到/想到可能發生的問題應該當下想辦法解決而不是置之不理。我的生活可說是相當的忙碌，遇到問題的時候我們很難馬上進行討論或者解決，只能說我已經盡力的去維護我們的程式。

如今這份專題似乎已經進入了尾聲，這學期我也相當享受于寫程式所帶來的成就感。覺得有點可惜的部分是一些剛開始就構想好的功能比如：包裝糖果的爆炸效果、包裝糖果與綫條糖果交換的爆炸效果、上鎖的糖果、障礙物等最後都沒有實作出來，一共4種游戲模式最後只實作了兩種。

在我們這個小組裏面，我作爲主導者從中學習與運用到了不少技巧，希望我的搭檔也能得到我擁有的，學到她想要的，反思這學期一路走來所遇到的困難與問題，不再重蹈覆轍。

* + 1. 陳小蘭：

By far the most important lesson I took out of this game is that whenever there's behavior that needs to be repeated around to multiple types of entities, it's better to default to abstracting/generalizing than to copy-pasting. Even if that impulse generally creates more problems than it solves, but later when you want to debug or add some function to your program, it will be much more easy to maintain the program. If the logic of checking permissions was repeated all over the code, it becomes difficult to fix issues that arise in the repeated code. When you fix a problem in one, you could easily forget to fix the problem in other occurrences. Also, if you have to modify the logic, you have to copy-paste all over the place. By having non-repeated code, you only have to maintain the code in a single place. New logic and bug fixes can be made in one place instead of many.

Beside, in the making process I've facing quite much error like syntax error and compilation errors. For me I try to solve it by look trough the internet about the problem or ask other people if needed. Even though there's a context mismatch between most programming advice I read on the Internet vs what I actually have learned but read more source might give me a valid answer to the problem I've been facing.

**附錄**

class CGameStateInit : public CGameState {

public:

CGameStateInit(CGame \*g);

void OnInit(); //遊戲的初值及圖形設定

void OnBeginState(); // 設定每次重玩所需的變數

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

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

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

protected:

void OnShow(); //顯示這個狀態的遊戲畫面

void OnMove();

void ShowHelpMenu();

void ShowHowToPlay();

void HelpMenuOnLButtonUp(CPoint point);

void SetMusic(bool);

private:

CMovingBitmap background, clickedPlayButton, helpButtonClicked; // Candycrush starter background, play button

CMovingBitmap helpBackground, aboutTab, howToPlayTab, cheatTab, howToPlay[3], about, cheat,

leftButton, rightButton, aboutButton, howToPlayButton, cheatButton;

CAnimation tiffy, playButton, toffee,candyCrush, helpButton;

bool playBtnClicked, finishLoaded, helpBtnClicked,onHelp,onAbout,onHowToPlay,onCheatPage;

int howToPlayPage;

};

CGameStateInit::CGameStateInit(CGame\* g)

: CGameState(g)

{

playBtnClicked = finishLoaded = onHelp = helpBtnClicked = onAbout = onHowToPlay = onSetting = onCheatPage= false;

howToPlayPage = 0;

}

void CGameStateInit::OnInit()

{

ShowLoading();

//Load background image

background.LoadBitmap("Bitmaps\\InitBackground.bmp");

/\*================== Load play button ======================\*/

int playBtnBmp[] = { IDB\_PLAYBUTTON\_1, IDB\_PLAYBUTTON\_2, IDB\_PLAYBUTTON\_3, IDB\_PLAYBUTTON\_4,

IDB\_PLAYBUTTON\_5, IDB\_PLAYBUTTON\_6, IDB\_PLAYBUTTON\_7, IDB\_PLAYBUTTON\_8,

IDB\_PLAYBUTTON\_9, IDB\_PLAYBUTTON\_10, IDB\_PLAYBUTTON\_11, IDB\_PLAYBUTTON\_12

};

for (int i = 0; i < 12; i++)

{

playButton.AddBitmap(playBtnBmp[i], RGB(0, 0, 0));

}

playButton.SetDelayCount(4);

clickedPlayButton.LoadBitmap("Bitmaps\\PlayButtonClicked.bmp", RGB(0, 0, 0));

/\*=========================================================\*/

/\*==================== Load tiffy =========================\*/

int TiffyBitmap[10] = { IDB\_TIFFY\_0, IDB\_TIFFY\_1, IDB\_TIFFY\_2, IDB\_TIFFY\_3, IDB\_TIFFY\_4,

IDB\_TIFFY\_5, IDB\_TIFFY\_6, IDB\_TIFFY\_7, IDB\_TIFFY\_8, IDB\_TIFFY\_9

};

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

{

tiffy.AddBitmap(TiffyBitmap[i], RGB(255, 255, 255));

}

for (int i = 8; i > 0; i--)

{

tiffy.AddBitmap(TiffyBitmap[i], RGB(255, 255, 255));

}

tiffy.SetDelayCount(4);

/\*==========================================================\*/

/\*===================== Load Toffee =======================\*/

int ToffeeBitmap[6] = { IDB\_TOFFEE\_1, IDB\_TOFFEE\_2, IDB\_TOFFEE\_3, IDB\_TOFFEE\_4, IDB\_TOFFEE\_5, IDB\_TOFFEE\_6 };

for (int i = 0; i < 6; i++)

{

toffee.AddBitmap(ToffeeBitmap[i], RGB(255, 255, 255));

}

for (int i = 5; i > 0; i--)

{

toffee.AddBitmap(ToffeeBitmap[i], RGB(255, 255, 255));

}

toffee.SetDelayCount(4);

/\*==========================================================\*/

/\*================== Load Candy Crush logo =================\*/

int CandyCrush[] = { IDB\_CANDY\_CRUSH\_1, IDB\_CANDY\_CRUSH\_2, IDB\_CANDY\_CRUSH\_3, IDB\_CANDY\_CRUSH\_4, IDB\_CANDY\_CRUSH\_5, IDB\_CANDY\_CRUSH\_6,

IDB\_CANDY\_CRUSH\_8, IDB\_CANDY\_CRUSH\_10, IDB\_CANDY\_CRUSH\_12, IDB\_CANDY\_CRUSH\_14, IDB\_CANDY\_CRUSH\_16, IDB\_CANDY\_CRUSH\_18,

IDB\_CANDY\_CRUSH\_20, IDB\_CANDY\_CRUSH\_21, IDB\_CANDY\_CRUSH\_22, IDB\_CANDY\_CRUSH\_23,

IDB\_CANDY\_CRUSH\_22, IDB\_CANDY\_CRUSH\_21, IDB\_CANDY\_CRUSH\_20, IDB\_CANDY\_CRUSH\_19, IDB\_CANDY\_CRUSH\_18, IDB\_CANDY\_CRUSH\_17,

IDB\_CANDY\_CRUSH\_16, IDB\_CANDY\_CRUSH\_15, IDB\_CANDY\_CRUSH\_14, IDB\_CANDY\_CRUSH\_13, IDB\_CANDY\_CRUSH\_12, IDB\_CANDY\_CRUSH\_11,

IDB\_CANDY\_CRUSH\_10, IDB\_CANDY\_CRUSH\_9, IDB\_CANDY\_CRUSH\_7, IDB\_CANDY\_CRUSH\_7, IDB\_CANDY\_CRUSH\_6,

IDB\_CANDY\_CRUSH\_7, IDB\_CANDY\_CRUSH\_9,IDB\_CANDY\_CRUSH\_10, IDB\_CANDY\_CRUSH\_11, IDB\_CANDY\_CRUSH\_12, IDB\_CANDY\_CRUSH\_13,

IDB\_CANDY\_CRUSH\_14, IDB\_CANDY\_CRUSH\_15, IDB\_CANDY\_CRUSH\_16, IDB\_CANDY\_CRUSH\_17, IDB\_CANDY\_CRUSH\_18, IDB\_CANDY\_CRUSH\_19,

IDB\_CANDY\_CRUSH\_19, IDB\_CANDY\_CRUSH\_18, IDB\_CANDY\_CRUSH\_17, IDB\_CANDY\_CRUSH\_16, IDB\_CANDY\_CRUSH\_15 };

for (int i = 0; i < 50; i++) {

candyCrush.AddBitmap(CandyCrush[i], RGB(255, 255, 255));

}

candyCrush.SetDelayCount(1);

candyCrush.SetCycle(false);

/\*==========================================================\*/

//load audio

CAudio::Instance()->Load(AUDIO\_STAGE, "sounds\\Overworld\_Level\_Select.mp3");

CAudio::Instance()->Load(AUDIO\_BTN\_CLICK, "sounds\\button\_press.wav");

CAudio::Instance()->Load(AUDIO\_BTN\_RELEASE, "sounds\\button\_release.wav");

//load setting button

settingButton.AddBitmap("Bitmaps/SettingButton-0.bmp", RGB(0, 0, 0));

settingButton.AddBitmap("Bitmaps/SettingButton-1.bmp", RGB(0, 0, 0));

settingButton.AddBitmap("Bitmaps/SettingButton-2.bmp", RGB(0, 0, 0));

settingButton.AddBitmap("Bitmaps/SettingButton-1.bmp", RGB(0, 0, 0));

settingButton.SetDelayCount(8);

settingButtonClicked.LoadBitmap("Bitmaps\\settingButtonClicked.bmp", RGB(0, 0, 0));

//setting menu background

settingMenu.LoadBitmap("Bitmaps\\setting\_0.bmp", RGB(255, 255, 255));

/\*===================== Load Setting Item =======================\*/

//load music on button

musicOn.AddBitmap("Bitmaps/MusicOnButton-0.bmp", RGB(0, 0, 0));

musicOn.AddBitmap("Bitmaps/MusicOnButton-1.bmp", RGB(0, 0, 0));

musicOn.AddBitmap("Bitmaps/MusicOnButton-2.bmp", RGB(0, 0, 0));

musicOn.AddBitmap("Bitmaps/MusicOnButton-1.bmp", RGB(0, 0, 0));

musicOn.SetDelayCount(8);

musicOnClicked.LoadBitmap("Bitmaps\\MusicOnButtonClicked.bmp", RGB(0, 0, 0));

//load music off button

musicOff.AddBitmap("Bitmaps/MusicOffButton-0.bmp", RGB(0, 0, 0));

musicOff.AddBitmap("Bitmaps/MusicOffButton-1.bmp", RGB(0, 0, 0));

musicOff.AddBitmap("Bitmaps/MusicOffButton-2.bmp", RGB(0, 0, 0));

musicOff.AddBitmap("Bitmaps/MusicOffButton-1.bmp", RGB(0, 0, 0));

musicOff.SetDelayCount(8);

musicOffClicked.LoadBitmap("Bitmaps\\MusicOffButtonClicked.bmp", RGB(0, 0, 0));

//loud sound-on button

soundOn.AddBitmap("Bitmaps/SoundOnButton-0.bmp", RGB(0, 0, 0));

soundOn.AddBitmap("Bitmaps/SoundOnButton-1.bmp", RGB(0, 0, 0));

soundOn.AddBitmap("Bitmaps/SoundOnButton-2.bmp", RGB(0, 0, 0));

soundOn.AddBitmap("Bitmaps/SoundOnButton-1.bmp", RGB(0, 0, 0));

soundOn.SetDelayCount(8);

soundOnClicked.LoadBitmap("Bitmaps\\SoundOnButtonClicked.bmp", RGB(0, 0, 0));

//loud sound-off button

soundOff.AddBitmap("Bitmaps/SoundOffButton-0.bmp", RGB(0, 0, 0));

soundOff.AddBitmap("Bitmaps/SoundOffButton-1.bmp", RGB(0, 0, 0));

soundOff.AddBitmap("Bitmaps/SoundOffButton-2.bmp", RGB(0, 0, 0));

soundOff.AddBitmap("Bitmaps/SoundOffButton-1.bmp", RGB(0, 0, 0));

soundOff.SetDelayCount(8);

soundOffClicked.LoadBitmap("Bitmaps\\SoundOffButtonClicked.bmp", RGB(0, 0, 0));

/\*==========================================================\*/

//load help button

helpButton.AddBitmap("Bitmaps/HelpButton-0.bmp", RGB(0, 0, 0));

helpButton.AddBitmap("Bitmaps/HelpButton-1.bmp", RGB(0, 0, 0));

helpButton.AddBitmap("Bitmaps/HelpButton-1.bmp", RGB(0, 0, 0));

helpButton.SetDelayCount(8);

helpButtonClicked.LoadBitmap("Bitmaps/HelpButtonClicked.bmp", RGB(0, 0, 0));

/\*===================== Load Help Item =======================\*/

helpBackground.LoadBitmap("Bitmaps/setting.bmp", RGB(0, 0, 0));

howToPlayTab.LoadBitmap("Bitmaps/TabHowToPlay.bmp", RGB(255, 255, 255));

aboutTab.LoadBitmap("Bitmaps/TabAbout.bmp", RGB(255, 255, 255));

cheatTab.LoadBitmap("Bitmaps/TabCheat.bmp", RGB(255, 255, 255));

leftButton.LoadBitmap("Bitmaps/Left.bmp", RGB(255, 255, 255));

rightButton.LoadBitmap("Bitmaps/Right.bmp", RGB(255, 255, 255));

howToPlay[0].LoadBitmap("Bitmaps/howtoplay1.bmp", RGB(255, 255, 255));

howToPlay[1].LoadBitmap("Bitmaps/howtoplay2.bmp", RGB(255, 255, 255));

howToPlay[2].LoadBitmap("Bitmaps/howtoplay3.bmp", RGB(255, 255, 255));

about.LoadBitmap("Bitmaps/About.bmp", RGB(255, 255, 255));

cheat.LoadBitmap("Bitmaps/Cheat.bmp", RGB(255, 255, 255));

aboutButton.LoadBitmap("Bitmaps/AboutButton.bmp", RGB(255, 255, 255));

howToPlayButton.LoadBitmap("Bitmaps/HowToPlayButton.bmp", RGB(255, 255, 255));

cheatButton.LoadBitmap("Bitmaps/CheatButton.bmp", RGB(255, 255, 255));

/\*==========================================================\*/

finishLoaded = true;

OnBeginState();

}

void CGameStateInit::OnBeginState()

{

if (finishLoaded)

{

candyCrush.Reset(); //reset animation of candy crush logo

playBtnClicked = false; //reset playbutton stste

if (music) CAudio::Instance()->Play(AUDIO\_STAGE, true); //play background music

}

}

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

{

const char KEY\_ESC = 27;

const char KEY\_SPACE = ' ';

if (nChar == KEY\_SPACE)

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

else if (nChar == KEY\_ESC) // Demo 關閉遊戲的方法

PostMessage(AfxGetMainWnd()->m\_hWnd, WM\_CLOSE, 0, 0); // 關閉遊戲

}

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

{

if (onSetting)

{

//setting area

SettingMenuOnLButtonDown(point);

}

//initiate menu

else

{

if (ButtonOnClick(point, playButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_CLICK, false);

playBtnClicked = true;

}

else playBtnClicked = false;

if (ButtonOnClick(point, helpButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_CLICK, false);

helpBtnClicked = true;

}

else helpBtnClicked = false;

}

SettingOnLButtonDown(point);

}

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

{

if (onSetting)

{

//setting area

SettingMenuOnLButtonUp(point);

}

else if (onHelp)

{

//help area

HelpMenuOnLButtonUp(point);

}

//initiate menu

else

{

if (ButtonOnClick(point, playButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_RELEASE, false);

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

}

else playBtnClicked = false;

if (ButtonOnClick(point,helpButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_RELEASE, false);

onHelp = onHelp ? false : true;

onAbout = true;

}

helpBtnClicked = false;

}

SettingOnLButtonUp(point);

}

void CGameStateInit::OnShow()

{

// 貼上背景

background.SetTopLeft(0, 0);

background.ShowBitmap();

//貼上Play Button

if (playBtnClicked)

{

clickedPlayButton.SetTopLeft(SIZE\_X / 2 - playButton.Width() / 2, SIZE\_Y / 5 \* 4 - playButton.Height());

clickedPlayButton.ShowBitmap();

}

else

{

playButton.SetTopLeft(SIZE\_X / 2 - playButton.Width() / 2, SIZE\_Y / 5 \* 4 - playButton.Height());

playButton.OnShow();

}

//character

tiffy.SetTopLeft(95, 400);

tiffy.OnShow();

toffee.SetTopLeft(700, 60);

toffee.OnShow();

candyCrush.SetTopLeft(250, -50);

candyCrush.OnShow();

//setting

ShowSettingButton();

ShowSettingMenu();

//help

if (helpBtnClicked)

{

helpButtonClicked.SetTopLeft(settingButton.Left() - helpButton.Width(), 0);

helpButtonClicked.ShowBitmap();

}

else

{

helpButton.SetTopLeft(settingButton.Left() - helpButton.Width(), 0);

helpButton.OnShow();

}

ShowHelpMenu();

}

void CGameStateInit::OnMove()

{

SettingOnMove();

SettingMenuOnMove();

candyCrush.OnMove();

if (!playBtnClicked) playButton.OnMove();

if (!helpBtnClicked) helpButton.OnMove();

tiffy.OnMove();

toffee.OnMove();

}

void CGameStateInit::ShowHelpMenu()

{

if (onHelp)

{

//show menu

helpBackground.SetTopLeft(SIZE\_X / 2 - (helpBackground.Width() / 2), SIZE\_Y / 2 - (helpBackground.Height() / 2));

helpBackground.ShowBitmap();

if (onAbout)

{

//show tab initiate (about tab)

aboutTab.SetTopLeft(helpBackground.Left() + (helpBackground.Width() - aboutTab.Width()) / 2, helpBackground.Top() + (helpBackground.Height() - aboutTab.Height()) / 2 + 40);

aboutTab.ShowBitmap();

//show about page

about.SetTopLeft(aboutTab.Left() + 3, aboutTab.Top() + 26);

about.ShowBitmap();

}

if (onHowToPlay)

{

//show tab how to play

ShowHowToPlay();

}

if (onCheatPage)

{

//show tab cheat

cheatTab.SetTopLeft(aboutTab.Left(), aboutTab.Top());

cheatTab.ShowBitmap();

//show cheat page

cheat.SetTopLeft(aboutTab.Left() + 3, aboutTab.Top() + 26);

cheat.ShowBitmap();

}

//set loaction of the tab

aboutButton.SetTopLeft(aboutTab.Left(), aboutTab.Top());

howToPlayButton.SetTopLeft(aboutTab.Left()+aboutButton.Width(), aboutTab.Top());

cheatButton.SetTopLeft(howToPlayButton.Left() + howToPlayButton.Width(), aboutTab.Top());

}

}

void CGameStateInit::ShowHowToPlay()

{

//show tab how to play

howToPlayTab.SetTopLeft(aboutTab.Left(), aboutTab.Top());

howToPlayTab.ShowBitmap();

//show initiate how to play page

howToPlay[howToPlayPage].SetTopLeft(howToPlayTab.Left() + 3, howToPlayTab.Top() + 26);

howToPlay[howToPlayPage].ShowBitmap();

//show left right button

if (howToPlayPage != 0) {

leftButton.SetTopLeft(howToPlay[howToPlayPage].Left() + 20, howToPlay[howToPlayPage].Top() + howToPlay[howToPlayPage].Height() - leftButton.Height() - 20);

leftButton.ShowBitmap();

}

if (howToPlayPage != 2) {

rightButton.SetTopLeft(howToPlay[howToPlayPage].Left() + howToPlay[howToPlayPage].Width() - rightButton.Width() - 20, howToPlay[howToPlayPage].Top() + howToPlay[howToPlayPage].Height() - rightButton.Height() - 20);

rightButton.ShowBitmap();

}

}

void CGameStateInit::HelpMenuOnLButtonUp(CPoint point)

{

//on help area

if (ButtonOnClick(point, helpBackground))

{

//about tab

if (ButtonOnClick(point, aboutButton))

{

onAbout = true;

onHowToPlay = onCheatPage = false;

}

//how to play tab

if (ButtonOnClick(point, howToPlayButton))

{

onHowToPlay = true;

onAbout = onCheatPage = false;

}

//cheat tab

if (ButtonOnClick(point, cheatButton))

{

onCheatPage = true;

onAbout = onHowToPlay = false;

}

//left right button

if (onHowToPlay)

{

if (ButtonOnClick(point, leftButton) && howToPlayPage>0)

{

howToPlayPage--;

}

if (ButtonOnClick(point, rightButton) && howToPlayPage<2)

{

howToPlayPage++;

}

}

}

else

{

onHelp = false;

}

}

void CGameStateInit::SetMusic(bool music)

{

this->music = music;

if (music)

CAudio::Instance()->Play(AUDIO\_STAGE, true);

else

CAudio::Instance()->Stop(AUDIO\_STAGE);

}

class CGameStateRun : public CGameState {

public:

CGameStateRun(CGame \*g); //Run Default game

~CGameStateRun();

void OnBeginState(); // 設定每次重玩所需的變數

void OnInit(); // 遊戲的初值及圖形設定

void OnKeyDown(UINT, UINT, UINT);

void OnKeyUp(UINT, UINT, UINT);

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

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

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

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

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

protected:

void OnMove(); // 移動遊戲元素

void OnShow(); // 顯示這個狀態的遊戲畫面

void SetMusic(bool);

private:

CMovingBitmap background; // 背景圖

};

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

{}

CGameStateRun::~CGameStateRun()

{}

void CGameStateRun::OnBeginState()

{

background.SetTopLeft(0, 0); // 設定背景的起始座標

if (music) CAudio::Instance()->Play(AUDIO\_JELLY, true); // 撥放 MIDI

}

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

{

//

// 如果希望修改cursor的樣式，則將下面程式的commment取消即可

//

// SetCursor(AfxGetApp()->LoadCursor(IDC\_GAMECURSOR));

SettingOnMove();

SettingMenuOnMove();

gameArea.OnMove();

if (gameArea.IsGameOver())

{

GotoGameState(GAME\_STATE\_OVER);

CAudio::Instance()->Stop(AUDIO\_JELLY);

}

}

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

{

background.LoadBitmap("Bitmaps\\inGameBG1.bmp");// 載入背景的圖形

//載入游戲音效

CAudio::Instance()->Load(AUDIO\_JELLY, "sounds\\MovesJellyLevels.mp3");

CAudio::Instance()->Load(AUDIO\_NEG\_SWAP, "sounds\\negative\_switch\_sound1.wav");

CAudio::Instance()->Load(AUDIO\_SWAP, "sounds\\switch\_sound1.wav");

CAudio::Instance()->Load(AUDIO\_SUPER\_CREATE, "sounds\\colour\_bomb\_created.wav");

CAudio::Instance()->Load(AUDIO\_LINE\_CREATE, "sounds\\striped\_candy\_created1.wav");

CAudio::Instance()->Load(AUDIO\_PACK\_CREATE, "sounds\\wrapped\_candy\_created1.wav");

CAudio::Instance()->Load(AUDIO\_POWER\_ALL, "sounds\\colour\_bomb1.wav");

CAudio::Instance()->Load(AUDIO\_SQUARE\_REMOVE1, "sounds\\square\_removed1.wav");

CAudio::Instance()->Load(AUDIO\_SQUARE\_REMOVE2, "sounds\\square\_removed2.wav");

CAudio::Instance()->Load(AUDIO\_LINE\_BLAST, "sounds\\line\_blast1.wav");

CAudio::Instance()->Load(AUDIO\_CANDY\_LAND1, "sounds\\candy\_land1.wav");

CAudio::Instance()->Load(AUDIO\_CANDY\_LAND2, "sounds\\candy\_land2.wav");

CAudio::Instance()->Load(AUDIO\_CANDY\_LAND3, "sounds\\candy\_land3.wav");

CAudio::Instance()->Load(AUDIO\_CANDY\_LAND4, "sounds\\candy\_land4.wav");

CAudio::Instance()->Load(AUDIO\_SUPER\_REMOVE, "sounds\\super\_colour\_bomb1.wav");

CAudio::Instance()->Load(AUDIO\_SWEET, "sounds\\sweet.wav");

CAudio::Instance()->Load(AUDIO\_TASTY, "sounds\\tasty.wav");

CAudio::Instance()->Load(AUDIO\_DELICIOUS, "sounds\\delicious.wav");

CAudio::Instance()->Load(AUDIO\_DIVINE, "sounds\\divine.wav");

CAudio::Instance()->Load(AUDIO\_SUGAR\_CRUSH, "sounds\\sugar\_crush.wav");

CAudio::Instance()->Load(AUDIO\_LEVEL\_FAIL, "sounds\\level\_failed1.wav");

CAudio::Instance()->Load(AUDIO\_LEVEL\_COMPLETE, "sounds\\level\_completed.wav");

for (int i = 0; i < 12; i++)

{

char sound[30] = { 0 };

sprintf(sound, "sounds\\combo\_sound%d.wav", i + 1);

CAudio::Instance()->Load(AUDIO\_COMBO1 + i, sound);

}

gameArea.LoadBitmap();

}

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

{

gameArea.OnKeyDown(nChar, nRepCnt, nFlags);

}

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

{

if (nChar == VK\_ESCAPE)

{

CAudio::Instance()->Stop(AUDIO\_JELLY);

GotoGameState(GAME\_STATE\_MENU);

}

gameArea.OnKeyUp(nChar, nRepCnt, nFlags);

}

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

{

SettingOnLButtonDown(point);

if (onSetting)

{

SettingMenuOnLButtonDown(point);

}

else

{

gameArea.OnLButtonDown(nFlags, point);

}

}

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

{

SettingOnLButtonUp(point);

if (onSetting)

{

SettingMenuOnLButtonUp(point);

}

else

{

gameArea.OnLButtonUp(nFlags, point);

}

}

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

{

// 沒事。如果需要處理滑鼠移動的話，寫code在這裡

}

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

{ }

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

{}

void CGameStateRun::OnShow()

{

background.ShowBitmap(); // 貼上背景圖

//setting button

ShowSettingButton();

ShowSettingMenu();

gameArea.OnShow();

}

void CGameStateRun::SetMusic(bool music)

{

this->music = music;

if (music)

CAudio::Instance()->Play(AUDIO\_JELLY, true);

else

CAudio::Instance()->Stop(AUDIO\_JELLY);

}

class CGameStateOver : public CGameState {

public:

CGameStateOver(CGame \*g);

void OnBeginState();

void OnLButtonDown(UINT nFlags, CPoint point); // handle mouse behavior

void OnLButtonUp(UINT nFlags, CPoint point); // handle mouse behavior

void OnInit();

protected:

void OnMove(); // 移動遊戲元素

void OnShow(); // 顯示這個狀態的遊戲畫面

int GetDigit(int);

void SetMusic(bool);

private:

void ShowButtons();

void ShowStars(int, int, int);

void StopAllMusic();

int counter, stageNum; // 倒數之計數器

bool isFail;

CMovingBitmap backgroundOver; // 背景圖

CMovingBitmap scoreBoardOver;

CInteger currentScore;

CInteger currentStage;

CMovingBitmap redStar, greenStar, yellowStar, emptyStar , youFailed;

CMovingBitmap exitButtonClicked, nextButtonClicked, retryButtonClicked;

CAnimation exitButton, nextButton, retryButton;

bool exitBtnClicked, nextBtnClicked, retryBtnClicked;

};

CGameStateOver::CGameStateOver(CGame\* g)

: CGameState(g)

{

currentScore.SetType(2);

currentStage.SetType(3);

nextBtnClicked = retryBtnClicked = exitBtnClicked = onSetting = false;

}

void CGameStateOver::OnMove()

{

SettingOnMove();

SettingMenuOnMove();

exitButton.OnMove();

nextButton.OnMove();

retryButton.OnMove();

}

void CGameStateOver::OnBeginState()

{

currentStage = current\_stage+1;

currentScore = (int)stages[current\_stage]->GetCurrentScore();

isFail = stages[current\_stage]->IsFail();

nextBtnClicked = retryBtnClicked = false;

if (music)

{

if (isFail) CAudio::Instance()->Play(AUDIO\_STATE\_FAIL, true);

else CAudio::Instance()->Play(AUDIO\_STATE\_COMPLETE, true);

}

}

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

{

if (onSetting)

{

SettingMenuOnLButtonDown(point);

}

else

{

//Exit Button

if (ButtonOnClick(point, exitButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_CLICK);

exitBtnClicked = true;

}

//Retry Button

if (ButtonOnClick(point, retryButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_CLICK);

retryBtnClicked = true;

}

//Next Button

if (ButtonOnClick(point, nextButton) && !isFail)

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_CLICK);

nextBtnClicked = true;

}

}

SettingOnLButtonDown(point);

}

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

{

if (onSetting)

{

SettingMenuOnLButtonUp(point);

}

else

{

//Exit Button

if (ButtonOnClick(point, exitButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_RELEASE);

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

StopAllMusic();

}

//Retry Button

if (ButtonOnClick(point, retryButton))

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_RELEASE);

gameArea.LoadStage(stages, current\_stage);

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

StopAllMusic();

}

//Next Button

if (ButtonOnClick(point, nextButton) && !isFail && currentStage.GetInteger() != MAX\_STAGE )

{

if (sound) CAudio::Instance()->Play(AUDIO\_BTN\_RELEASE);

current\_stage += 1;

gameArea.LoadStage(stages, current\_stage);

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

StopAllMusic();

}

nextBtnClicked = retryBtnClicked = exitBtnClicked = false;

}

SettingOnLButtonUp(point);

}

void CGameStateOver::OnInit()

{

//load background

backgroundOver.LoadBitmap("Bitmaps/inGameBG1.bmp");

//load score board

scoreBoardOver.LoadBitmap("Bitmaps/score\_state\_over.bmp", RGB(0, 0, 0));

//load star

redStar.LoadBitmap("Bitmaps/RedStar.bmp", RGB(251, 230, 239));

greenStar.LoadBitmap("Bitmaps/GreenStar.bmp", RGB(251, 230, 239));

yellowStar.LoadBitmap("Bitmaps/YellowStar.bmp", RGB(251, 230, 239));

emptyStar.LoadBitmap("Bitmaps/ContainerStar.bmp", RGB(251, 230, 239));

youFailed.LoadBitmap("Bitmaps/Failed.bmp", RGB(251, 230, 239));

//load button

exitButton.AddBitmap("Bitmaps/ExitButton-0.bmp", RGB(255, 255, 255));

exitButton.AddBitmap("Bitmaps/ExitButton-1.bmp", RGB(255, 255, 255));

exitButton.AddBitmap("Bitmaps/ExitButton-2.bmp", RGB(255, 255, 255));

exitButton.AddBitmap("Bitmaps/ExitButton-1.bmp", RGB(255, 255, 255));

exitButton.SetDelayCount(8);

exitButtonClicked.LoadBitmap("Bitmaps\\ExitButtonClicked.bmp", RGB(255, 255, 255));

int nextBtn[] = { IDB\_NEXTBTN\_0, IDB\_NEXTBTN\_1, IDB\_NEXTBTN\_2, IDB\_NEXTBTN\_3, IDB\_NEXTBTN\_4,

IDB\_NEXTBTN\_5, IDB\_NEXTBTN\_6, IDB\_NEXTBTN\_7, IDB\_NEXTBTN\_8, IDB\_NEXTBTN\_9 };

int retryBtn[] = { IDB\_RETRYBTN\_0, IDB\_RETRYBTN\_1, IDB\_RETRYBTN\_2, IDB\_RETRYBTN\_3, IDB\_RETRYBTN\_4,

IDB\_RETRYBTN\_5, IDB\_RETRYBTN\_6, IDB\_RETRYBTN\_7, IDB\_RETRYBTN\_8, IDB\_RETRYBTN\_9 };

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

{

nextButton.AddBitmap(nextBtn[i], RGB(251, 230, 239));

retryButton.AddBitmap(retryBtn[i], RGB(251, 230, 239));

}

nextButton.SetDelayCount(4);

retryButton.SetDelayCount(4);

nextButtonClicked.LoadBitmap("Bitmaps\\NextButtonClicked.bmp", RGB(251, 230, 239));

retryButtonClicked.LoadBitmap("Bitmaps\\RetryButtonClicked.bmp", RGB(251, 230, 239));

//Load background music

CAudio::Instance()->Load(AUDIO\_STATE\_FAIL, "sounds\\Level\_Failed.mp3");

CAudio::Instance()->Load(AUDIO\_STATE\_COMPLETE, "sounds\\Level\_Complete.mp3");

}

int CGameStateOver::GetDigit(int n)

{

n = abs(n);

int digit = 0;

while (n > 0)

{

digit++;

n /= 10;

}

return digit == 0 ? 1 : digit;

}

void CGameStateOver::SetMusic(bool music)

{

this->music = music;

if (music)

{

if (isFail) CAudio::Instance()->Play(AUDIO\_STATE\_FAIL, true);

else CAudio::Instance()->Play(AUDIO\_STATE\_COMPLETE, true);

}

else

{

CAudio::Instance()->Stop(AUDIO\_STATE\_FAIL);

CAudio::Instance()->Stop(AUDIO\_STATE\_COMPLETE);

}

}

void CGameStateOver::ShowButtons()

{

//set button location

int exitBtnTopLX = scoreBoardOver.Left() + scoreBoardOver.Width() - exitButton.Width();

int exitBtnTopLY = scoreBoardOver.Top() + 30;

int retryBtnTopLX = (backgroundOver.Width() / 2) + (isFail || (current\_stage+1 == 15) ? -nextButton.Width() / 2 : 20 - nextButton.Width());

int retryBtnTopLY = (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2) + 530;

int nextBtnTopLX = (backgroundOver.Width() / 2) - 20;

int nextBtnTopLY = (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2) + 530;

//Show exit button

if (exitBtnClicked)

{

exitButtonClicked.SetTopLeft(exitBtnTopLX, exitBtnTopLY);

exitButtonClicked.ShowBitmap();

}

else

{

exitButton.SetTopLeft(exitBtnTopLX, exitBtnTopLY);

exitButton.OnShow();

}

//Show retry button

if (retryBtnClicked)

{

retryButtonClicked.SetTopLeft(retryBtnTopLX, retryBtnTopLY);

retryButtonClicked.ShowBitmap();

}

else

{

retryButton.SetTopLeft(retryBtnTopLX, retryBtnTopLY);

retryButton.OnShow();

}

//Show "Failed" if fail, show next button else

if (isFail)

{

youFailed.SetTopLeft((backgroundOver.Width() / 2) - (youFailed.Width() / 2), (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2) + 180);

youFailed.ShowBitmap();

}

else if(current\_stage + 1 != 15)

{

if (nextBtnClicked)

{

nextButtonClicked.SetTopLeft(nextBtnTopLX, nextBtnTopLY);

nextButtonClicked.ShowBitmap();

}

else

{

nextButton.SetTopLeft(nextBtnTopLX, nextBtnTopLY);

nextButton.OnShow();

}

}

}

void CGameStateOver::ShowStars(int amount, int xStar, int yStar)

{

//show 3 star

if (amount == 3)

{

yellowStar.SetTopLeft(xStar, yStar);

yellowStar.ShowBitmap();

yellowStar.SetTopLeft(xStar + 110 + 20, yStar - 20);

yellowStar.ShowBitmap();

yellowStar.SetTopLeft(xStar + 220 + 40, yStar);

yellowStar.ShowBitmap();

}

//show 1/2 star

else if (amount >= 1)

{

redStar.SetTopLeft(xStar, yStar);

redStar.ShowBitmap();

if (amount == 2)

{

greenStar.SetTopLeft(xStar + 110 + 20, yStar - 20);

greenStar.ShowBitmap();

}

else {

emptyStar.SetTopLeft(xStar + 110 + 20, yStar - 20);

emptyStar.ShowBitmap();

}

emptyStar.SetTopLeft(xStar + 220 + 40, yStar);

emptyStar.ShowBitmap();

}

//show 0 star

else {

emptyStar.SetTopLeft(xStar, yStar);

emptyStar.ShowBitmap();

emptyStar.SetTopLeft(xStar + 110 + 20, yStar - 20);

emptyStar.ShowBitmap();

emptyStar.SetTopLeft(xStar + 220 + 40, yStar);

emptyStar.ShowBitmap();

}

}

void CGameStateOver::StopAllMusic()

{

CAudio::Instance()->Stop(AUDIO\_STATE\_FAIL);

CAudio::Instance()->Stop(AUDIO\_STATE\_COMPLETE);

}

void CGameStateOver::OnShow()

{

//show background

backgroundOver.SetTopLeft(0, 0);

backgroundOver.ShowBitmap();

//setting button

ShowSettingButton();

ShowSettingMenu();

//show score board

scoreBoardOver.SetTopLeft((backgroundOver.Width() / 2) - (scoreBoardOver.Width() / 2), (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2));

scoreBoardOver.ShowBitmap();

//show stage

currentStage.SetTopLeft((backgroundOver.Width() / 2) + 60, (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2) + 55);

currentStage.ShowBitmap();

//show star

int xStar = (backgroundOver.Width() / 2) - (370 / 2);

int yStar = (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2) + 180;

if (stages[current\_stage]->GetCurrentScore() >= stages[current\_stage]->GetScoreThree() && !isFail)

{ //Show 3 yellow star if current score higher than star three

ShowStars(3, xStar, yStar);

}

else if (stages[current\_stage]->GetCurrentScore() >= stages[current\_stage]->GetScoreTwo() && !isFail)

{ //Show 2 star if current score higher than star two

ShowStars(2, xStar, yStar);

}

else if (stages[current\_stage]->GetCurrentScore() >= stages[current\_stage]->GetScoreOne() && !isFail)

{ //Show 1 star if current score higher than star one

ShowStars(1, xStar, yStar);

}

else if (stages[current\_stage]->GetCurrentScore() < stages[current\_stage]->GetScoreOne() && !isFail)

{ //Show 0 star if current score higher than star one

ShowStars(0, xStar, yStar);

}

//show score

currentScore.SetTopLeft((backgroundOver.Width() / 2) - (60 \* GetDigit(currentScore.GetInteger()) / 2), (backgroundOver.Height() / 2) - (scoreBoardOver.Height() / 2) + 390);

currentScore.ShowBitmap();

ShowButtons();

}

class CGameStateMenu : public CGameState

{

public:

CGameStateMenu(CGame \*g);

~CGameStateMenu();

void OnInit();

void OnBeginState();

void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags); // handle keyboard behavior

void OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags); // handle keyboard behavior

void OnLButtonDown(UINT nFlags, CPoint point); // handle mouse behavior

void OnLButtonUp(UINT nFlags, CPoint point); // handle mouse behavior

void OnMouseMove(UINT nFlags, CPoint point); // handle mouse behavior

void OnRButtonDown(UINT nFlags, CPoint point); // handle mouse behavior

void OnRButtonUp(UINT nFlags, CPoint point); // handle mouse behavior

void SetMovingUp(bool status);

void SetMovingDown(bool status);

int GetDigit(int n);

protected:

void OnMove();

void OnShow();

void SetMusic(bool);

private:

void ShowStageButton(int, int, int, int);

void ShowStars(int, int, int);

CMovingBitmap menuBackground, woodBackgourd, stageButton[5];

CMovingBitmap star1, star2, star3, comingSoon;

int sy;

int MAX\_Y, MIN\_Y;

bool IsMovingUp, IsMovingDown, drag;

bool goldFinger;

int StagePos[15][2];

CInteger stageNum;

LONG clickX, clickY, clickSY; //save position of mouse and menu when mouse clicked

int mouseDisplayment, inertia;

};

CGameStateMenu::CGameStateMenu(CGame \*g)

: CGameState(g), drag(false), mouseDisplayment(0), inertia(0), goldFinger(false)

{

IsMovingUp = false; IsMovingDown = false;

MAX\_Y = 0; MIN\_Y = -3600;

sy = -3600;

int Pos[][2] = { {270,4030},{495,3980},{530,3850},{320,3870},{135,3910},

{135,3750},{340,3690},{570,3720},{770,3800},{960,3840},

{1085,3750},{1010,3600},{760,3540},{520,3590},{280,3585} };

for (int i = 0; i < 15; i++) {

for (int j = 0; j < 2; j++) {

StagePos[i][j] = Pos[i][j];

}

}

}

CGameStateMenu::~CGameStateMenu()

{

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

delete stages[i];

}

void CGameStateMenu::OnInit()

{

//background

woodBackgourd.LoadBitmap("Bitmaps\\WoodBackground.bmp");

menuBackground.LoadBitmap("Bitmaps\\stage\_map.bmp");

comingSoon.LoadBitmap("Bitmaps\\ComingSoon.bmp", RGB(0, 0, 0));

stageNum.SetType(1);

int StageButton[5] = { IDB\_STAGE\_BUTTON\_BLUE, IDB\_STAGE\_BUTTON\_RED, IDB\_STAGE\_BUTTON\_GREEN, IDB\_STAGE\_BUTTON\_YELLOW, IDB\_STAGE\_BUTTON\_GREY };

//unlock icon

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

stageButton[i].LoadBitmap(StageButton[i], RGB(255, 255, 255));

}

//star icon

star1.LoadBitmap("Bitmaps\\SmallRedStar.bmp", RGB(255, 255, 255));

star2.LoadBitmap("Bitmaps\\SmallGreenStar.bmp", RGB(255, 255, 255));

star3.LoadBitmap("Bitmaps\\SmallYellowStar.bmp", RGB(255, 255, 255));

//load stage

for (int i = 0; i < MAX\_STAGE + 1; i++) {

stages.push\_back(new Stage(i + 1));

stages[i]->LoadStage();

}

}

void CGameStateMenu::OnBeginState()

{

goldFinger = false;

}

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

{

const char KEY\_UP = 0x26;

const char KEY\_DOWN = 0x28;

//menu control with keyboard

if (nChar == KEY\_UP) IsMovingUp = true;

if (nChar == KEY\_DOWN) IsMovingDown = true;

if (nChar == VK\_F1) goldFinger = true;

}

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

{

const char KEY\_UP = 0x26;

const char KEY\_DOWN = 0x28;

const char KEY\_ESC = 27;

//menu control with keyboard

if (nChar == KEY\_UP) IsMovingUp = false;

if (nChar == KEY\_DOWN) IsMovingDown = false;

if (nChar == VK\_F1) goldFinger = false;

if (nChar == KEY\_ESC)

{

CAudio::Instance()->Stop(AUDIO\_STAGE);

GotoGameState(GAME\_STATE\_INIT);

}

}

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

{

drag = true;

SettingOnLButtonDown(point);

if (onSetting)

{

SettingMenuOnLButtonDown(point);

}

else

{

clickX = point.x;

clickY = point.y;

clickSY = sy;

}

}

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

{

drag = false;

SettingOnLButtonUp(point);

if (onSetting)

{

//setting button

SettingMenuOnLButtonUp(point);

}

else

{

//stage

int x = point.x;

int y = point.y - sy;

for (int i = 0; i < MAX\_STAGE ; i++) {

if (StagePos[i][0] < x && x < (StagePos[i][0] + 60) && StagePos[i][1] < y && (y < StagePos[i][1] + 60))

{

if (stages[i]->IsUnlock() || goldFinger)

{

current\_stage = i;

gameArea.LoadStage(stages, i);

CAudio::Instance()->Stop(AUDIO\_STAGE);

GotoGameState(GAME\_STATE\_RUN);

}

}

}

}

}

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

{

if (drag)

{

int displayment = point.y - clickY;

sy = clickSY + displayment;

inertia = displayment < 0 ? -20 : 20;

}

}

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

{}

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

{}

void CGameStateMenu::SetMovingUp(bool status)

{

if (status && sy <= MAX\_Y)

sy += 5;

}

void CGameStateMenu::SetMovingDown(bool status)

{

if (status && sy >= MIN\_Y)

sy -= 5;

}

void CGameStateMenu::OnMove()

{

SettingOnMove();

SettingMenuOnMove();

SetMovingUp(IsMovingUp);

SetMovingDown(IsMovingDown);

if (!drag && inertia > 0) sy += inertia--;

else if (!drag && inertia < 0) sy += inertia++;

}

int CGameStateMenu::GetDigit(int n)

{

n = abs(n);

int digit = 0;

while (n > 0)

{

digit++;

n /= 10;

}

return digit == 0 ? 1 : digit;

}

void CGameStateMenu::OnShow()

{

//show wood background

woodBackgourd.SetTopLeft(0, 0);

woodBackgourd.ShowBitmap();

//show menu background

menuBackground.SetTopLeft(40, sy);

menuBackground.ShowBitmap();

//show stage map

if (sy < MAX\_Y && sy < MIN\_Y)

sy = -3600;

if (sy > MAX\_Y && sy > MIN\_Y)

sy = 0;

if (sy > -3000)

{

comingSoon.SetTopLeft(SIZE\_X / 2 - comingSoon.Width() / 2, SIZE\_Y / 2 - comingSoon.Height() / 2);

comingSoon.ShowBitmap();

}

//setting button

ShowSettingButton();

ShowSettingMenu();

//show unlock icon

for (int i = 0; i < MAX\_STAGE ; i++)

{

int xStar = StagePos[i][0] - 10, xButton = StagePos[i][0] - 5;

int yStar = StagePos[i][1] + sy + 65, yButton = StagePos[i][1] - 3 + sy;

stageNum.SetInteger(i + 1);

if (stages[i]->IsUnlock())

{

if (stages[i]->GetLastScoreHistory() >= stages[i]->GetScoreThree())

{ //Show yellow button if history score higher than star three

ShowStageButton(3, i, xButton, yButton);

ShowStars(3, xStar, yStar);

}

else if (stages[i]->GetLastScoreHistory() >= stages[i]->GetScoreTwo())

{ //Show green button if history star three > score higher > star two

ShowStageButton(2, i, xButton, yButton);

ShowStars(2, xStar, yStar);

}

else if (stages[i]->GetLastScoreHistory() >= stages[i]->GetScoreOne())

{ //Show red button if star two > history score > star one

ShowStageButton(1, i, xButton, yButton);

ShowStars(1, xStar, yStar);

}

else if (stages[i]->GetLastScoreHistory() < stages[i]->GetScoreOne())

{ //Show blue button if the stage is unlocked but no history score

ShowStageButton(0, i, xButton, yButton);

}

} //Show gray button if the stage is locked

else ShowStageButton(4, i, xButton, yButton);

}

}

void CGameStateMenu::SetMusic(bool music)

{

this->music = music;

if (music)

CAudio::Instance()->Play(AUDIO\_STAGE, true);

else

CAudio::Instance()->Stop(AUDIO\_STAGE);

}

void CGameStateMenu::ShowStageButton(int stageBtn, int stage, int xButton, int yButton)

{

stageButton[stageBtn].SetTopLeft(xButton, yButton);

stageButton[stageBtn].ShowBitmap();

if (stages[stage]->IsUnlock())

{

stageNum.SetTopLeft(xButton + ((stageButton[stageBtn].Width() / 2) - (10 \* GetDigit(stage) / 2)), yButton + (stageButton[stageBtn].Height() / 4));

stageNum.ShowBitmap();

}

}

void CGameStateMenu::ShowStars(int amount, int xStar, int yStar)

{

if (amount == 3)

{

//show 3 star

star3.SetTopLeft(xStar, yStar);

star3.ShowBitmap();

star3.SetTopLeft(xStar + 30, yStar + 5);

star3.ShowBitmap();

star3.SetTopLeft(xStar + 60, yStar);

star3.ShowBitmap();

}

else if (amount >= 1)

{

//show 1 star

star1.SetTopLeft(xStar, yStar);

star1.ShowBitmap();

if (amount == 2)

{

//show 2 star

star2.SetTopLeft(xStar + 30, yStar + 5);

star2.ShowBitmap();

}

}

}

#pragma once

#ifndef AUDIO\_ID\_H

#define AUDIO\_ID\_H

enum AUDIO\_ID { // 定義各種音效的編號

AUDIO\_JELLY, AUDIO\_STAGE,

AUDIO\_NEG\_SWAP, AUDIO\_SWAP,

AUDIO\_SUPER\_CREATE, AUDIO\_LINE\_CREATE, AUDIO\_PACK\_CREATE, AUDIO\_POWER\_ALL,

AUDIO\_SQUARE\_REMOVE1, AUDIO\_SQUARE\_REMOVE2, AUDIO\_LINE\_BLAST, AUDIO\_SUPER\_REMOVE,

AUDIO\_CANDY\_LAND1, AUDIO\_CANDY\_LAND2, AUDIO\_CANDY\_LAND3, AUDIO\_CANDY\_LAND4,

AUDIO\_COMBO1, AUDIO\_COMBO2, AUDIO\_COMBO3, AUDIO\_COMBO4, AUDIO\_COMBO5, AUDIO\_COMBO6,

AUDIO\_COMBO7, AUDIO\_COMBO8, AUDIO\_COMBO9, AUDIO\_COMBO10, AUDIO\_COMBO11, AUDIO\_COMBO12,

AUDIO\_SWEET, AUDIO\_TASTY, AUDIO\_DELICIOUS, AUDIO\_DIVINE, AUDIO\_SUGAR\_CRUSH,

AUDIO\_LEVEL\_FAIL, AUDIO\_LEVEL\_COMPLETE, AUDIO\_STATE\_FAIL, AUDIO\_STATE\_COMPLETE,

AUDIO\_BTN\_CLICK, AUDIO\_BTN\_RELEASE

};

#endif

#pragma once

#ifndef STAGE\_H

#define STAGE\_H

namespace game\_framework

{

class Stage

{

friend class GameArea;

public:

Stage(int);

~Stage();

void LoadStage(); //Load Stage from file

double GetScoreOne(); //return score of one star

double GetScoreTwo(); //return score of two star

double GetScoreThree(); //return score of three star

double GetLastScoreHistory(); //Get last highest score

bool IsUnlock();

bool IsFail();

void SetPassOrFail(int);

void RemoveLine(); //re-create the cnt\_stg.txt without last 2 lines(last score and isunlock)

void WriteBack(); //Write game data to file

void SetUnlock();

int GetCurrentScore();

int GetCurrentStage();

void SetCurrentScore(int score);

private:

int map[13][20];

double scoreOne, scoreTwo, scoreThree; //Target score for three star

int vertical, horizontal, pack, chocolate; //total special candy spawn onInit

int candyType; //Max candy type in this game

double lastHighScore; //History highest score

int maxStep;

int mode;

bool isUnlock;

bool isFail;

bool initcandy;

string stageTxt;

int currentStage;

int currentScore;

bool hasPortal; //is portal exist/s

vector<pair<CPoint, CPoint>> portalList; //list of all teleporter (start point, end point)

};

}

#endif

#include "stdafx.h"

#include "Resource.h"

#include <mmsystem.h>

#include <ddraw.h>

#include <fstream>

#include <string>

#include "audio.h"

#include "gamelib.h"

#include "Stage.h"

using namespace std;

game\_framework::Stage::Stage(int files)

{

scoreOne = 0; scoreTwo = 0; scoreThree = 0; //Target score for three star

vertical = 0; horizontal = 0; pack = 0; chocolate = 0; //total special candy spawn onInit

candyType = 0; //Max candy type in this game

lastHighScore = 0; //History highest score

maxStep = 0;

mode = 0;

currentScore = 0;

currentStage = files;

isUnlock = 0;

stageTxt = "Stages\\cnt\_stage" + to\_string(files) + ".txt";

hasPortal = false;

}

game\_framework::Stage::~Stage()

{

WriteBack();

}

void game\_framework::Stage::LoadStage()

{

fstream InputStage;

InputStage.open(stageTxt);

if (!InputStage.is\_open())

{

char fileName[200] = "\nOpen file failed! file: ";

strcat(fileName, stageTxt.c\_str());

strcat(fileName, "\nReason: ");

GAME\_ASSERT(0, strcat(fileName, strerror(errno)));

}

string data[13];

string tempPortal;

string file;

//MAP

for (int i = 0; i < 13; i++)

{

getline(InputStage, file);

for (int j = 0; j < 20; j++)

{

switch (file[j])

{

case '0': map[i][j] = 0; break; //0 = none, !0 = gameArea

case '2': map[i][j] = 2; break; //2 = normalArea

case '3': map[i][j] = 3; break; //3 = singleJelly

case '4': map[i][j] = 4; break; //4 = doubleJelly

case '1': map[i][j] = 1; break; //1 = candy spawning area

}

}

}

//Other stage information

for (int i = 2; i < 13; i++)

{

getline(InputStage, file, '\n');

data[i] = file.substr(0, file.find('\t'));

}

//PORTAL

int portal;

string portalLoc;

getline(InputStage, file, '\n');

portal = stoi(file.substr(0, file.find('\t')));

if (portal == 1) {

hasPortal = true;

getline(InputStage, file, '\n');

tempPortal = file.substr(0, file.find('\t'));

portal = stoi(tempPortal.substr(0, tempPortal.find(' ')));

tempPortal.erase(0, tempPortal.find(' ')+1);

for (int i = 0; i < portal; i++) {

CPoint begin, end;

portalLoc = tempPortal.substr(0, tempPortal.find(' '));

tempPortal.erase(0, tempPortal.find(' ') + 1);

begin.x = stoi(portalLoc);

portalLoc = tempPortal.substr(0, tempPortal.find(' '));

tempPortal.erase(0, tempPortal.find(' ') + 1);

begin.y = stoi(portalLoc);

portalLoc = tempPortal.substr(0, tempPortal.find(' '));

tempPortal.erase(0, tempPortal.find(' ') + 1);

end.x = stoi(portalLoc);

portalLoc = tempPortal.substr(0, tempPortal.find(' '));

tempPortal.erase(0, tempPortal.find(' ') + 1);

end.y = stoi(portalLoc);

portalList.push\_back(pair<CPoint, CPoint>(begin, end));

}

}

//LAST SCORE

getline(InputStage, file, '\n');

data[0] = file.substr(0, file.find('\t'));

//IS\_UNCLOCK

getline(InputStage, file, '\n');

data[1] = file.substr(0, file.find('\t'));

lastHighScore = stod(data[0]);

isUnlock = stoi(data[1]);

scoreOne = stod(data[2]);

scoreTwo = stod(data[3]);

scoreThree = stod(data[4]);

maxStep = stoi(data[5]);

vertical = stoi(data[6]);

horizontal = stoi(data[7]);

pack = stoi(data[8]);

chocolate = stoi(data[9]);

candyType = stoi(data[10]);

mode = stoi(data[11]);

initcandy = stoi(data[12]);

InputStage.close();

}

double game\_framework::Stage::GetScoreOne()

{

return scoreOne;

}

double game\_framework::Stage::GetScoreTwo()

{

return scoreTwo;

}

double game\_framework::Stage::GetScoreThree()

{

return scoreThree;

}

double game\_framework::Stage::GetLastScoreHistory()

{

return lastHighScore;

}

bool game\_framework::Stage::IsUnlock()

{

return isUnlock == 1 ? true : false;

}

bool game\_framework::Stage::IsFail()

{

return isFail;

}

void game\_framework::Stage::SetPassOrFail(int fail)

{

isFail = fail == 1 ? 1 : 0;

}

int game\_framework::Stage::GetCurrentScore()

{

return currentScore;

}

int game\_framework::Stage::GetCurrentStage()

{

return currentStage;

}

/\*function to re-create the cnt\_stg.txt

without last 2 lines(last score and isunlock\*/

void game\_framework::Stage::RemoveLine()

{

ifstream read(stageTxt);

ofstream myFile;

string file;

const char\* data = stageTxt.data();

myFile.open("temp.txt", ofstream::out);

int line\_no = 0, n = 26;

if (hasPortal)

n =27;

while (!read.eof())

{

getline(read, file, '\n');

line\_no++;

if (line\_no < n )

{

myFile << file;

myFile << '\n';

}

}

myFile.close();

read.close();

remove(data);

rename("temp.txt", data);

}

void game\_framework::Stage::WriteBack()

{

RemoveLine();

ofstream myFile(stageTxt, ofstream::app);

myFile << lastHighScore << "\t#LastScore\n";

myFile << isUnlock <<"\t#IsUnlock";

myFile.close();

}

void game\_framework::Stage::SetCurrentScore(int score)

{

currentScore = score;

}

void game\_framework::Stage::SetUnlock()

{

isUnlock = true;

}

#ifndef GAMEAREA\_H

#define GAMEAREA\_H

#define MaxHeight 13

#define MaxWidth 20

namespace game\_framework

{

class GameArea

{

public:

GameArea();

~GameArea();

int GetScore(); //Get current score

void InitCandy(bool drop = true); //spawn all candies randomly

bool IsGameOver();

int GetCurrentStage();

void LoadBitmap(); //load game area's background

void LoadStage(vector<Stage\*>& stages, int); //read map from file

void OnShow(); //Display game area and candy

void OnMove();

void OnKeyDown(UINT, UINT, UINT); //handle keyboard press

void OnKeyUp(UINT, UINT, UINT); //handle keyboard unpress

void OnLButtonDown(UINT nFlags, CPoint point); //handle mouse click

void OnLButtonUp(UINT nFlags, CPoint point); //handle mouse unclick

void OnMouseMove(UINT nFlags, CPoint point); //handle mouse move

private:

int ClearCombo(); //search and delete all combo

int Compare(int, int); //Compare two int

void Delay(); //pause game state for (delay)secs

void DropCandy();

int DropCandyStraight();

int DropCandySide();

int DeleteCombo(set<Candy\*>&); //analyze and delete combo

void Find(Candy\*, unsigned&, unsigned&); //find candy and return its' row and column

int FindCombo();

void GetCandies(set<Candy\*>&, int, int, int); //get continuous candies

void GetLine(vector<Candy\*>&, vector<Candy\*>&, char check); //collect candies on a same line

void GetWordBmp(double\*\*, int\*\*, CMovingBitmap\*\*, int);

void GotoGameStateOver(bool result);

void InitClickedCandy(); //unclick & clear candies in clickedCandies

void InitGameArea(int);

bool IsDropping(); //check all candies are still

bool IsNeighbour(Candy&, Candy&); //return are candies in clickedCandies is neighbour

void OnMoveBlasts();

void OnMoveEnding();

void PlayVoiceEffect(int audio\_id);

void PowerAll(int style, int power, int x, int y); //Power all specific candy

int PutCandy(); //spawn candies at spawning area

void PutEndingBonus();

int RemoveContinuous(vector<Candy\*>&, char, bool(\*)(Candy\*, Candy\*), set<Candy\*>&); //Find and remove continuous candy

void RemoveContinuous(vector<Candy\*>&, unsigned, unsigned, char, set<Candy\*>&); //Remove continuous candy

void ReleaseInOrder();

void ReleasePower(Candy\*, unsigned row = 0, unsigned column = 0); //remove candy with consider its' power

void ReleaseSwap(); //activate power of candy when 2 powered candy swapped

void RemoveAll(int row, int column); //remove all candies

void RemoveRow(unsigned); //remove whole row of candies

void RemoveColumn(unsigned); //remove whole column of candies

void RemoveSquare(int, int, int); //remove surrounding candies according to level

void RemoveStyle(int x, int y, int style = 0); //remove all specific candy

void ShowLoading(); //Show loading image

void ShowPortal(int position);

void SwapCandy(); //Swap candies in clickedCandies

void TeleportCandy();

void UpdateCurPosition(); //update current position of every candy

const int x, y; //top left x,y of gameArea

int map[MaxHeight][MaxWidth]; //Array of container

int delay, delayRemoveStyle;

int currentComboSound, totalCandyCleared;

int MAX\_RAND\_NUM; //types of candies in this games

bool initiating, ending, running, gameOver;

bool \*sound;

bool playingVoice;

bool delayRemove, releaseSwap;

bool goldFinger;

Candy\* curPosition[MaxHeight][MaxWidth]; //save current position of every candy

Candy candies[MaxHeight][MaxWidth]; //Array of candy

CMovingBitmap area, singleJelly, doubleJelly; //container bmp

CMovingBitmap sweet, tasty, delicious, divine, sugarCrush; //Voice Effect

CMovingBitmap portalStart1, portalStart2, portalEnd1, portalEnd2;

vector<Candy\*> clickedCandies; //save clicked candy/ies maximun size = 2

vector<Stage\*>::iterator stage; //save current stage

list<pair<int, int>> spawnArea; //save position where candy spawn

list<Blast\*> blasts; //save blasts pending to play

list<list<Candy\*>\*> removeList; //These candies will be removed after delay

list<Candy\*> endingBonus;

ScoreBoard scoreBoard;

};

}

#endif

#include "stdafx.h"

#include <omp.h>

#include "Resource.h"

#include <mmsystem.h>

#include <fstream>

#include <string>

#include <ddraw.h>

#include <algorithm>

#include <set>

#include "audio.h"

#include "gamelib.h"

#include "Blast.h"

#include "Candy.h"

#include "Stage.h"

#include "ScoreBoard.h"

#include "GameArea.h"

namespace game\_framework

{

static int audioID[12] = { AUDIO\_COMBO1, AUDIO\_COMBO2, AUDIO\_COMBO3, AUDIO\_COMBO4, AUDIO\_COMBO5, AUDIO\_COMBO6,

AUDIO\_COMBO7, AUDIO\_COMBO8, AUDIO\_COMBO9, AUDIO\_COMBO10, AUDIO\_COMBO11, AUDIO\_COMBO12 };

GameArea::GameArea()

:x(280), y(35), MAX\_RAND\_NUM(4), initiating(1), ending(0), running(1), gameOver(0), delay(0),

delayRemoveStyle(0), delayRemove(0), currentComboSound(0), goldFinger(0), releaseSwap(0), totalCandyCleared(0),

playingVoice(0)

{

scoreBoard.score = 0;

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

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

curPosition[i][j] = NULL;

sound = &CGameState::sound;

}

GameArea::~GameArea()

{

for (auto i = blasts.begin(); i != blasts.end(); i++)

delete \*i;

for (auto i = removeList.begin(); i != removeList.end(); i++)

delete \*i;

}

void GameArea::LoadBitmap()

{

area.LoadBitmap(IDB\_CONTAINER);

singleJelly.LoadBitmap("Bitmaps\\Jelly1.bmp");

doubleJelly.LoadBitmap("Bitmaps\\Jelly2.bmp");

scoreBoard.LoadBitmap();

sweet.LoadBitmap("Bitmaps\\sweet.bmp", RGB(0, 0, 0));

tasty.LoadBitmap("Bitmaps\\tasty.bmp", RGB(0, 0, 0));

delicious.LoadBitmap("Bitmaps\\delicious.bmp", RGB(0, 0, 0));

divine.LoadBitmap("Bitmaps\\divine.bmp", RGB(0, 0, 0));

sugarCrush.LoadBitmap("Bitmaps\\SugarCrush.bmp", RGB(0, 0, 0));

portalStart1.LoadBitmap("Bitmaps\\portalStart1.bmp", RGB(0, 0, 0));

portalStart2.LoadBitmap("Bitmaps\\portalStart2.bmp", RGB(0, 0, 0));

portalEnd1.LoadBitmap("Bitmaps\\portalEnd1.bmp", RGB(0, 0, 0));

portalEnd2.LoadBitmap("Bitmaps\\portalEnd2.bmp", RGB(0, 0, 0));

Candy::LoadBitmap();

NormalBlast::LoadBitmap();

LineBlast::LoadBitmap();

SuperBlast::LoadBitmap();

MagicBlast::LoadBitmap();

}

void GameArea::LoadStage(vector<Stage\*>& stages, int index)

{

int totalJelly = 0;

//Update new spawn area

spawnArea.clear();

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

{

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

{

map[i][j] = stages[index]->map[i][j];

if (map[i][j] == 1)

spawnArea.push\_back(pair<int, int>(i, j));

else if (map[i][j] == 3)

totalJelly += 1;

else if (map[i][j] == 4)

totalJelly += 2;

}

}

//Update new gamedata

MAX\_RAND\_NUM = stages[index]->candyType;

scoreBoard.oneStar = stages[index]->scoreOne;

scoreBoard.twoStar = stages[index]->scoreTwo;

scoreBoard.threeStar = stages[index]->scoreThree;

scoreBoard.lastHighScore = stages[index]->lastHighScore;

scoreBoard.moves = stages[index]->maxStep;

scoreBoard.mode = stages[index]->mode;

this->stage = find(stages.begin(), stages.end(), stages[index]);

//Init GameArea

InitGameArea(totalJelly);

InitCandy(stages[index]->initcandy);

}

//get candy position

void GameArea::Find(Candy \*candy, unsigned &row, unsigned &column)

{

for (unsigned i = 0; i < MaxHeight; i++)

for (unsigned j = 0; j < MaxWidth; j++)

if (&candies[i][j] == candy)

{

row = i;

column = j;

return;

}

}

void GameArea::ReleasePower(Candy \*candy, unsigned row, unsigned column)

{

if(candy != NULL) Find(candy, row, column); //if candy != NULL, get its' position in array

else candy = &candies[row][column]; //else, row & column is candy's position

if (!map[row][column]) return; //return if candy is out of range in map

if (!initiating && (map[row][column] == 3 || map[row][column] == 4))

{ //break if there is a jelly at this position

map[row][column]--;

scoreBoard.target--; //deduct total amount of jelly

}

if (!initiating)scoreBoard.score += 60; //increase total score

if (!initiating && candy->GetStyle())

{ //play animation of a candy being destroyed

blasts.push\_back(new NormalBlast(candy->GetStyle(), candy->GetTopLeftX(), candy->GetTopLeftY()));

totalCandyCleared++;

}

int power = candy->GetPower(), style = candy->GetStyle();

candy->Kill();

switch (power)

{

case 0:

break;

case 1://trigger horizontal Line blast

if (!initiating && style)

{

blasts.push\_back(new LineBlast(style, candy->GetTopLeftX(), candy->GetTopLeftY(), 1));

}

RemoveRow(row);

break;

case 2://trigger vertical Line Blast

if (!initiating && style)

{

blasts.push\_back(new LineBlast(style, candy->GetTopLeftX(), candy->GetTopLeftY(), 2));

}

RemoveColumn(column);

break;

case 3://trigger pack blast

RemoveSquare(row, column, 1);

if (!initiating && \*sound)CAudio::Instance()->Play(AUDIO\_SQUARE\_REMOVE1, false);

break;

case 4: //trigger super Blast

RemoveStyle(candy->GetTopLeftX(), candy->GetTopLeftY());

break;

}

}

void GameArea::ReleaseSwap()

{

int firstPow = clickedCandies[0]->GetPower(), secondPow = clickedCandies[1]->GetPower();

if (firstPow == 4 && secondPow == 4)

{ //Swap 2 super candy

unsigned row, column;

Find(clickedCandies[0], row, column);

clickedCandies[0]->Kill();

clickedCandies[1]->Kill();

RemoveAll(row, column);

}

else if (firstPow == 4 && secondPow)

{ //Swap 1 super candy with power 1~3

PowerAll(clickedCandies[1]->GetStyle(), secondPow, clickedCandies[0]->GetTopLeftX(), clickedCandies[0]->GetTopLeftY());

clickedCandies[0]->Kill();

}

else if (secondPow == 4 && firstPow)

{ //Swap 1 super candy with power 1~3

PowerAll(clickedCandies[0]->GetStyle(), firstPow, clickedCandies[1]->GetTopLeftX(), clickedCandies[1]->GetTopLeftY());

clickedCandies[1]->Kill();

}

else if (firstPow == 4 && !secondPow)

{ //Swap 1 super candy with normal candy

RemoveStyle(clickedCandies[0]->GetTopLeftX(), clickedCandies[0]->GetTopLeftY(), clickedCandies[1]->GetStyle());

clickedCandies[0]->Kill();

}

else if (secondPow == 4 && !firstPow)

{ //Swap 1 super candy with normal candy

RemoveStyle(clickedCandies[1]->GetTopLeftX(), clickedCandies[1]->GetTopLeftY(), clickedCandies[0]->GetStyle());

clickedCandies[1]->Kill();

}

else if (firstPow > 0 && firstPow < 3 && secondPow > 0 && secondPow < 3)

{ //Swap 2 striped candy

unsigned row, column;

Find(clickedCandies[0], row, column);

blasts.push\_back(new LineBlast(clickedCandies[0]->GetStyle(), clickedCandies[0]->GetTopLeftX(), clickedCandies[0]->GetTopLeftY(), 1));

blasts.push\_back(new LineBlast(clickedCandies[0]->GetStyle(), clickedCandies[1]->GetTopLeftX(), clickedCandies[1]->GetTopLeftY(), 2));

clickedCandies[0]->Kill();

clickedCandies[1]->Kill();

RemoveRow(row);

RemoveColumn(column);

}

else if (firstPow == 3 && secondPow == 3)

{ //Swap 2 wrapped candy

unsigned row, column;

Find(clickedCandies[0], row, column);

RemoveSquare(row, column, 2);

if (\*sound) CAudio::Instance()->Play(AUDIO\_SQUARE\_REMOVE2, false);

}

else if (firstPow == 3 && secondPow > 0 && secondPow < 3)

{ //Swap 1 wrapped candy with striped candy

unsigned row, column;

Find(clickedCandies[1], row, column);

for (unsigned i = row - 1; i < row + 2; i++)

RemoveRow(i);

for (unsigned i = column - 1; i < column + 2; i++)

RemoveColumn(i);

}

else if (secondPow == 3 && firstPow > 0 && firstPow < 3)

{ //Swap 1 wrapped candy with striped candy

unsigned row, column;

Find(clickedCandies[0], row, column);

for (unsigned i = row - 1; i < row + 2; i++)

RemoveRow(i);

for (unsigned i = column - 1; i < column + 2; i++)

RemoveColumn(i);

}

InitClickedCandy();

clickedCandies.clear();

scoreBoard.moves--;

}

void GameArea::RemoveRow(unsigned row)

{

for (unsigned i = 0; i < MaxWidth; i++)

if (map[row][i])

ReleasePower(&candies[row][i]);

}

void GameArea::RemoveColumn(unsigned column)

{

for (unsigned i = 0; i < MaxWidth; i++)

if (map[i][column])

ReleasePower(&candies[i][column]);

}

void GameArea::RemoveSquare(int row, int column, int level)

{ //Level 1: wrapped candy be activated by normal way

if (level == 1)

{

for (int i = row - 2; i < row + 3; i++)

for (int j = column - 2; j < column + 3; j++)

{

if (i == row && j == column) continue;

if (i >= 0 && i < MaxHeight && j >= 0 && j < MaxWidth)

if (i >= row - 1 && i < row + 2 && j >= column - 1 && j < column + 2)

ReleasePower(NULL, i, j);

else

{

int pushX = Compare(candies[i][j].GetTopLeftX(), candies[row][column].GetTopLeftX());

int pushY = Compare(candies[i][j].GetTopLeftY(), candies[row][column].GetTopLeftY());

candies[i][j].Push(pushX, pushY);

}

}

}

//Level 2: two wrapped candies being swapped with each other

else if (level == 2)

{

for (int i = row - 2; i < row + 3; i++)

for (int j = column - 2; j < column + 3; j++)

{

if (i == row && j == column) continue;

if (i >= 0 && i < MaxHeight && j >= 0 && j < MaxWidth)

ReleasePower(NULL, i, j);

}

}

}

void GameArea::RemoveAll(int row, int column)

{ //two superCandy being swapped with each other

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

{

SuperBlast\* superBlast = new SuperBlast(column \* 50 + 280, row \* 50 + 35);

removeList.push\_back(new list<Candy\*>);

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

{

if (map[i][j])

{

(\*removeList.back()).push\_back(&candies[i][j]);

superBlast->AddPoint(candies[i][j].GetTopLeftX() + 25, candies[i][j].GetTopLeftY() + 25);

}

}

blasts.push\_back(superBlast);

if (!(\*removeList.back()).size())

{ //to avoid game crash, delete list if there's no candy to be removed in this row

delete removeList.back();

removeList.pop\_back();

delete blasts.back();

blasts.pop\_back();

}

}

}

void GameArea::RemoveStyle(int x, int y, int style)

{

if (!style) style = rand() % MAX\_RAND\_NUM + 1;

removeList.push\_back(new list<Candy\*>);

SuperBlast\* superBlast = (x || y) ? new SuperBlast(x, y, 4) : NULL;

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

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

if (candies[i][j].GetStyle() == style && candies[i][j].GetPower() != 4)

{

(\*removeList.rbegin())->push\_back(&candies[i][j]);

if (superBlast != NULL) superBlast->AddPoint(candies[i][j].GetTopLeftX() + 25, candies[i][j].GetTopLeftY() + 25);

}

if (superBlast != NULL) blasts.push\_back(superBlast);

if (!(\*removeList.back()).size())

{ //to avoid game crash, delete list if there's no candy to be removed in this row

delete removeList.back();

removeList.pop\_back();

}

else if (superBlast != NULL && \*sound) CAudio::Instance()->Play(AUDIO\_SUPER\_REMOVE, false);

}

void GameArea::PowerAll(int style, int power, int x, int y)

{

SuperBlast\* superBlast = new SuperBlast(x, y, 0, true);

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

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

if (candies[i][j].GetStyle() == style && candies[i][j].GetPower() != 4)

{

if (power == 1 || power == 2) power = rand() % 2 + 1;

candies[i][j].SetPower(power);

superBlast->AddPoint(candies[i][j].GetTopLeftX() + 25, candies[i][j].GetTopLeftY() + 25);

}

blasts.push\_back(superBlast);

if (\*sound) CAudio::Instance()->Play(AUDIO\_POWER\_ALL, false);

delay = (int)(1000.0 / GAME\_CYCLE\_TIME);

delayRemoveStyle = style;

delayRemove = true;

}

int GameArea::GetScore()

{

return scoreBoard.score.GetInteger();

}

int GameArea::Compare(int first, int second)

{

return first < second ? - 1 : first > second ? 1 : 0;

}

void GameArea::Delay()

{

if (delayRemove)

{

if (delay > 0) delay--;

else

{

RemoveStyle(0, 0, delayRemoveStyle);

delayRemove = false;

}

}

}

void GameArea::UpdateCurPosition()

{

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

{

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

{

//Get candy current position by its' center coordinate

int curMapX = (candies[i][j].GetCurrentX() - 280 + 25) / 50;

int curMapY = (candies[i][j].GetCurrentY() - 35 + 25) / 50;

if(curMapX >= 0 && curMapX < MaxWidth && curMapY >= 0 && curMapY < MaxHeight)

if(candies[i][j].GetStyle() > 0 && (map[curMapY][curMapX] || map[curMapY - 1][curMapX]))

curPosition[curMapY][curMapX] = &candies[i][j];

else

curPosition[i][j] = NULL;

}

}

}

void GameArea::OnShow()

{

scoreBoard.OnShow();

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

// Show gamearea ///

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

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

{

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

{

switch (map[i][j])

{

case 0:

continue;

case 3:

singleJelly.SetTopLeft(j \* 50 + x, i \* 50 + y);

singleJelly.ShowBitmap();

break;

case 4:

doubleJelly.SetTopLeft(j \* 50 + x, i \* 50 + y);

doubleJelly.ShowBitmap();

break;

default:

area.SetTopLeft(j \* 50 + x, i \* 50 + y);

area.ShowBitmap();

break;

}

}

}

ShowPortal(1);

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

// Show all candy ///

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

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

{

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

{

if (curPosition[i][j] != NULL)

curPosition[i][j]->OnShow();

}

}

ShowPortal(2);

for (auto i = blasts.begin(); i != blasts.end(); i++)

(\*i)->OnShow();

PlayVoiceEffect(-1);

}

void GameArea::OnMove()

{

UpdateCurPosition();

PutCandy(); //put candy at apawning area if it's empty

DropCandy(); //drop if candy hvnt touch the ground/other candy

TeleportCandy();

#pragma omp parallel for

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

#pragma omp parallel for

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

if(candies[i][j].GetStyle() > 0)

candies[i][j].OnMove(initiating);

int comboCleared = IsDropping() ? 0 : FindCombo();

OnMoveBlasts();

if (!comboCleared && !IsDropping())

OnMoveEnding();

ReleaseInOrder();

Delay();

}

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

{

if (nChar == VK\_F1) goldFinger = true;

}

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

{

if (nChar == VK\_F1) goldFinger = false;

}

int GameArea::FindCombo()

{

if (releaseSwap)

{

ReleaseSwap();

InitClickedCandy();

releaseSwap = false;

return 1;

}

if (!initiating && !delayRemove && !removeList.size())

{

int candyCleared = ClearCombo();

if (candyCleared)

{ //play combo sound

currentComboSound = currentComboSound > 11 ? 11 : currentComboSound;

if (\*sound) CAudio::Instance()->Play(audioID[currentComboSound++], false);

}

else

{

if (totalCandyCleared)

{

char cc[50];

sprintf(cc, "comboCleared : %d\n", totalCandyCleared);

TRACE(cc);

}

if (totalCandyCleared > 30) PlayVoiceEffect(AUDIO\_DIVINE);

else if (totalCandyCleared >= 24) PlayVoiceEffect(AUDIO\_DELICIOUS);

else if (totalCandyCleared > 12) PlayVoiceEffect(AUDIO\_TASTY);

else if (totalCandyCleared == 12) PlayVoiceEffect(AUDIO\_SWEET);

currentComboSound = totalCandyCleared = 0;

}

if (candyCleared && clickedCandies.size() == 2)

{//If there is a combo after swapping candies, initiate click, -1 moves

scoreBoard.moves -= 1;

InitClickedCandy();

}

else if (!candyCleared && clickedCandies.size() == 2)

{ //else swap two candies back to original position

if (\*sound) CAudio::Instance()->Play(AUDIO\_NEG\_SWAP, false);

SwapCandy();

InitClickedCandy();

}

}

else if (!delayRemove && !removeList.size()) ClearCombo();

return totalCandyCleared;

}

void GameArea::OnMoveBlasts()

{

for (auto i = blasts.begin(); i != blasts.end();)

{

if ((\*i)->IsLast())

{

delete \*i;

i = blasts.erase(i);

}

else

{

(\*i)->OnMove();

i++;

}

}

}

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

{}

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

{

if (!delay && !ending && !IsDropping())

{

int column = (point.x - 280) / 50;

int row = (point.y - 35) / 50;

if (!map[row][column]) return;

if (goldFinger && candies[row][column].GetStyle() > 0)

{ //Gold Finger: developer use only

int currentPower = candies[row][column].GetPower();

candies[row][column].SetPower(currentPower == 4 ? 0 : currentPower + 1);

}

else

{

Candy\* clickedCandy = candies[row][column].Click();

auto candy = find(clickedCandies.begin(), clickedCandies.end(), clickedCandy); //GetCandyIterator

if (candy == clickedCandies.end())

clickedCandies.push\_back(clickedCandy);

else clickedCandies.erase(candy); //If player click a candy twice, initiate click

if (clickedCandies.size() == 2)

{ //if two candies clicked, swap if they are neighbour

if (IsNeighbour(\*clickedCandies[0], \*clickedCandies[1]))

{

SwapCandy();

if (\*sound) CAudio::Instance()->Play(AUDIO\_SWAP, false);

//Release swapPower when each of clickedCandy is superCandy or both are poweredCandy

if (clickedCandies[0]->GetPower() == 4 || clickedCandies[1]->GetPower() == 4 || (clickedCandies[0]->GetPower() && clickedCandies[1]->GetPower()))

releaseSwap = true;

}

else if (!releaseSwap) InitClickedCandy();

}

}

}

}

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

{

}

void GameArea::ShowLoading()

{

CMovingBitmap loading; // 貼上loading圖示

loading.LoadBitmap(IDB\_INGAME\_LOADING, RGB(0, 0, 0));

loading.SetTopLeft(0, 0);

loading.ShowBitmap();

CDDraw::BltBackToPrimary(); // 將 Back Plain 貼到螢幕

}

void GameArea::ShowPortal(int position)

{

vector<pair<CPoint, CPoint>>\* portalList = &(\*stage)->portalList; // get portal position

for (int i = 0; i < (int)portalList->size(); i++)

{

if (!map[(\*portalList)[i].first.y][(\*portalList)[i].first.x] || !map[(\*portalList)[i].second.y][(\*portalList)[i].second.x])

continue;

int toTopLeftX = (\*portalList)[i].first.x \* 50 + x;

int toTopLeftY = (\*portalList)[i].first.y \* 50 + y + 40;

int fromTopLeftX = (\*portalList)[i].second.x \* 50 + x;

int fromTopLeftY = (\*portalList)[i].second.y \* 50 + y - 5;

if (position == 1)

{

portalStart1.SetTopLeft(fromTopLeftX, fromTopLeftY);

portalStart1.ShowBitmap();

portalEnd1.SetTopLeft(toTopLeftX, toTopLeftY);

portalEnd1.ShowBitmap();

}

else if (position == 2)

{

portalStart2.SetTopLeft(fromTopLeftX, fromTopLeftY);

portalStart2.ShowBitmap();

portalEnd2.SetTopLeft(toTopLeftX, toTopLeftY);

portalEnd2.ShowBitmap();

}

}

}

void GameArea::InitCandy(bool drop)

{

for (auto i = blasts.begin(); i != blasts.end();)

{

delete \*i;

blasts.erase(i++);

}

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

{

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

{

switch (map[i][j])

{

case 0:

candies[i][j] = Candy(j \* 50 + x, i \* 50 + y);

break;

default:

int id = drop == true ? 0 : rand() % MAX\_RAND\_NUM + 1;

candies[i][j] = Candy(id, j \* 50 + x, i \* 50 + y);

break;

}

}

}

UpdateCurPosition();

while (!drop && (ClearCombo() || IsDropping()))

{

OnMove();

ShowLoading();

}

initiating = false;

}

bool GameArea::IsGameOver()

{

return !running;

}

int GameArea::GetCurrentStage()

{

return ((\*stage)->GetCurrentStage());

}

void GameArea::DropCandy()

{

if(DropCandyStraight()) return;

DropCandySide();

}

int GameArea::DropCandyStraight()

{

int total = 0, count = 1;

//Drop in current column

do

{

for (int i = MaxHeight - 1; i >= 0; i--)

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

if (map[i + 1][j] != 0 && candies[i][j].GetStyle() > 0 && !candies[i + 1][j].GetStyle())

{

candies[i][j].SetDestination(candies[i][j].GetTopLeftY() + 50);

candies[i + 1][j] = candies[i][j];

candies[i][j].SetStyle(0);

total++;

count = 1;

}

}

while (count--);

return total;

}

int GameArea::DropCandySide()

{

int total = 0;

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

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

if (map[i + 1][j] != 0 && candies[i][j].GetStyle() > 0)

if (map[i + 1][j - 1] && curPosition[i + 1][j] != NULL && candies[i][j - 1].GetStyle() < 0 && !candies[i + 1][j - 1].GetStyle())

{

candies[i][j].SetDestination(candies[i][j].GetTopLeftX() - 50, candies[i][j].GetTopLeftY() + 50);

candies[i + 1][j - 1] = candies[i][j];

candies[i][j].SetStyle(0);

total++;

}

else if (map[i + 1][j + 1] && curPosition[i + 1][j] != NULL && candies[i][j + 1].GetStyle() < 0 && !candies[i + 1][j + 1].GetStyle())

{

candies[i][j].SetDestination(candies[i][j].GetTopLeftX() + 50, candies[i][j].GetTopLeftY() + 50);

candies[i + 1][j + 1] = candies[i][j];

candies[i][j].SetStyle(0);

total++;

}

return total;

}

int GameArea::ClearCombo()

{

set<Candy\*> accumulateCandy;

int comboDeleted = 0;

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

{

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

{

if (candies[i][j].GetStyle() <= 0) continue;

accumulateCandy.insert(&candies[i][j]); //put the first candy into set

GetCandies(accumulateCandy, i, j, candies[i][j].GetStyle());//collect all similar candies that follow-up with first

comboDeleted += DeleteCombo(accumulateCandy); //delete all combo

}

}

return comboDeleted;

}

void GameArea::GetCandies(set<Candy\*>& accumulateCandy, int i, int j, int checkStyle)

{

if (candies[i][j].GetPower() == 4) return;

//Recursive condition: there is(are) same candy(ies) nearby

int currentStyle = candies[i][j].GetStyle();

candies[i][j].SetStyle(0);

if (j + 1 < MaxWidth && candies[i][j + 1].GetStyle() == checkStyle && candies[i][j + 1].GetPower() != 4)

{//check to the right

accumulateCandy.insert(&candies[i][j + 1]);

GetCandies(accumulateCandy, i, j + 1, checkStyle);

}

if (i + 1 < MaxHeight && candies[i + 1][j].GetStyle() == checkStyle && candies[i + 1][j].GetPower() != 4)

{//Check downward

accumulateCandy.insert(&candies[i + 1][j]);

GetCandies(accumulateCandy, i + 1, j, checkStyle);

}

if (j - 1 >= 0 && candies[i][j - 1].GetStyle() == checkStyle && candies[i][j - 1].GetPower() != 4)

{//Check to the left

accumulateCandy.insert(&candies[i][j - 1]);

GetCandies(accumulateCandy, i, j - 1, checkStyle);

}

if (i - 1 >= 0 && candies[i - 1][j].GetStyle() == checkStyle && candies[i - 1][j].GetPower() != 4)

{//Check upward

accumulateCandy.insert(&candies[i - 1][j]);

GetCandies(accumulateCandy, i - 1, j, checkStyle);

}

candies[i][j].SetStyle(currentStyle);

}

//Functions CompareX() and CompareY() are used by stable\_sort

bool CompareX(Candy\* candy1, Candy\* candy2) //

{ //

return (candy1->GetTopLeftX() < candy2->GetTopLeftX()); //

} //

//

bool CompareY(Candy\* candy1, Candy\* candy2) //

{ //

return candy1->GetTopLeftY() < candy2->GetTopLeftY(); //

} //

//==========================================================//

int GameArea::DeleteCombo(set<Candy\*>&accumulateCandy)

{

if (accumulateCandy.size() < 3)

{ //Pass

accumulateCandy.clear();

return 0;

}

vector<int> x, y;

vector<Candy\*> toDelete;

set<Candy\*> temp;

int comboDeleted = 0;

for (auto i = accumulateCandy.begin(); i != accumulateCandy.end(); i++)

{

x.push\_back((\*i)->GetTopLeftX());

y.push\_back((\*i)->GetTopLeftY());

}

for (auto i = accumulateCandy.begin(); i != accumulateCandy.end(); i++)

{ /\*delete if more than 3 candies on a vertical line\*/

if (count(x.begin(), x.end(), (\*i)->GetTopLeftX()) >= 3)

toDelete.push\_back(\*i);

}

comboDeleted += RemoveContinuous(toDelete, 'y', &CompareY, temp);

for (auto i = accumulateCandy.begin(); i != accumulateCandy.end(); i++)

{ /\*delete if more than 3 candies on a horizontal line\*/

if (count(y.begin(), y.end(), (\*i)->GetTopLeftY()) >= 3)

toDelete.push\_back(\*i);

}

comboDeleted += RemoveContinuous(toDelete, 'x', &CompareX, temp);

accumulateCandy.clear();

return comboDeleted;

}

void GameArea::PutEndingBonus()

{

int totalPut = scoreBoard.moves.GetInteger();

list<Candy\*> candy;

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

{

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

{

if (candies[i][j].GetStyle() > 0 && candies[i][j].GetPower() == 0)

candy.push\_back(&candies[i][j]);

}

}

do

{

for (auto i = candy.begin(); i != candy.end();)

{

if (totalPut > 0 && !(rand() % totalPut))

{

endingBonus.push\_back(\*i);

i = candy.erase(i);

totalPut--;

}

else i++;

}

} while (totalPut > 0);

}

int GameArea::RemoveContinuous(vector<Candy\*>& toDelete, char axis, bool(\*Compare)(Candy\*, Candy\*), set<Candy\*>& temp)

{

if (!toDelete.size()) return 0; //pass

vector<Candy\*> line;

char check = axis == 'x' ? 'y' : 'x';

int comboDeleted = 0;

while(1)

{

GetLine(line, toDelete, check); //collect candies on a same line

stable\_sort(line.begin(), line.end(), Compare); //When their x/y are same, sort according to y/x

int count = 1;

for (unsigned int i = 0; i < line.size() - 1; i++)

{

if (line[i]->GetTopLeft(axis) + 50 == line[i + 1]->GetTopLeft(axis))

count++; //If next candy is follow-up with current, keep counting

else if (count < 3) count = 1; //else, if count >= 3 -> combo, or pass

else

{

RemoveContinuous(line, i - (count - 1), i, axis, temp);

count = 1;

comboDeleted += count;

}

}

if (count >= 3)

{

RemoveContinuous(line, (unsigned)(line.size() - count), (unsigned)(line.size()), axis, temp);

comboDeleted += count;

}

line.clear();

if (toDelete.size() < 3) break; //break if there is not enough candies to form a combo

}

toDelete.clear();

return comboDeleted;

}

void GameArea::RemoveContinuous(vector<Candy\*>& line, unsigned offset, unsigned lineSize, char axis, set<Candy\*>& temp)

{

bool packCandy = !initiating;

bool linePower = !initiating && lineSize - offset == 4 ? true : false;

bool superCandy = !initiating && lineSize - offset > 4 ? true : false;

for (unsigned int j = offset; j < lineSize; j++)

{

ReleasePower(line[j]);

if (axis == 'y') temp.insert(line[j]);

else if(packCandy && find(temp.begin(), temp.end(), line[j]) != temp.end())

{

if (!initiating && \*sound)CAudio::Instance()->Play(AUDIO\_PACK\_CREATE, false);

line[j]->SetPower(3);

line[j]->Relive();

superCandy = linePower = packCandy = false;

continue;

}

if (linePower && find(clickedCandies.begin(), clickedCandies.end(), line[j]) != clickedCandies.end())

{

if (!initiating && \*sound)CAudio::Instance()->Play(AUDIO\_LINE\_CREATE, false);

line[j]->Relive();

line[j]->SetPower(axis == 'x' ? 2 : 1);

linePower = false;

continue;

}

if (superCandy && find(clickedCandies.begin(), clickedCandies.end(), line[j]) != clickedCandies.end())

{

if (!initiating && \*sound)CAudio::Instance()->Play(AUDIO\_SUPER\_CREATE, false);

line[j]->Relive();

line[j]->SetPower(4);

superCandy = false;

continue;

}

}

if (linePower)

{

if (!initiating && \*sound)CAudio::Instance()->Play(AUDIO\_LINE\_CREATE, false);

line[offset]->SetPower(axis == 'x' ? 2 : 1);

line[offset]->Relive();

}

if (superCandy)

{

if (!initiating && \*sound) CAudio::Instance()->Play(AUDIO\_SUPER\_CREATE, false);

line[offset]->SetPower(4);

line[offset]->Relive();

}

}

void GameArea::ReleaseInOrder()

{

for (auto i = removeList.begin(); i != removeList.end();)

{

ReleasePower(\*(\*i)->begin());

(\*i)->pop\_front();

if (!(\*i)->size())

{

delete \*i;

i = removeList.erase(i);

}

else i++;

if (gameOver) gameOver = false;

}

}

void GameArea::GetLine(vector<Candy\*>& line, vector<Candy\*>& toDelete, char check)

{

int currentLine = toDelete[0]->GetTopLeft(check);

for (auto i = toDelete.begin(); i != toDelete.end();)

{

if ((\*i)->GetTopLeft(check) == currentLine)

{

line.push\_back(\*i);

i = toDelete.erase(i);

}

else i++;

}

}

void GameArea::GetWordBmp(double\*\* size, int \*\* frame, CMovingBitmap \*\* word, int audio\_id)

{

\*size = new double(0.2);

\*frame = new int(0);

switch (audio\_id)

{

case AUDIO\_SWEET:

\*word = &sweet;

break;

case AUDIO\_TASTY:

\*word = &tasty;

break;

case AUDIO\_DELICIOUS:

\*word = &delicious;

break;

case AUDIO\_DIVINE:

\*word = &divine;

break;

case AUDIO\_SUGAR\_CRUSH:

\*word = &sugarCrush;

break;

default:

GAME\_ASSERT(0, "Invalid audio id!");

}

playingVoice = true;

}

void GameArea::GotoGameStateOver(bool result)

{

// save result and data to stage after delay, then show result in game state over

if (delay > 0) delay--;

else if (result)

{ // Unlock next stage and save score highest history if win

if (\*sound) CAudio::Instance()->Play(AUDIO\_LEVEL\_COMPLETE, false);

(\*(stage + 1))->SetUnlock();

(\*stage)->currentScore = scoreBoard.score.GetInteger();

(\*stage)->lastHighScore = scoreBoard.lastHighScore < scoreBoard.score ? scoreBoard.score.GetInteger() : scoreBoard.lastHighScore;

(\*stage)->SetPassOrFail(0);

running = false;

}

else

{

if (\*sound) CAudio::Instance()->Play(AUDIO\_LEVEL\_FAIL, false);

(\*stage)->currentScore = scoreBoard.score.GetInteger();

(\*stage)->SetPassOrFail(1);

running = false;

}

}

int GameArea::PutCandy()

{ // Spawn candy if spawning area is empty

int total = 0;

for (auto i = spawnArea.begin(); i != spawnArea.end(); i++)

if (curPosition[i->first][i->second] == NULL)

{

int id = rand() % MAX\_RAND\_NUM + 1; //random type of Candy

candies[i->first][i->second] = Candy(id, i->second \* 50 + x, i->first \* 50 + y - 25);

candies[i->first][i->second].SetDestination(i->first \* 50 + y);

total++;

}

return total;

}

bool GameArea::IsDropping()

{

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

{

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

if (map[i][j] != 0 && candies[i][j].IsMoving())

return true;

}

if (blasts.size() || playingVoice || delayRemove) return true;

return false;

}

void GameArea::SwapCandy()

{

clickedCandies[0]->SetDestination(clickedCandies[1]->GetCurrentX(), clickedCandies[1]->GetCurrentY());

clickedCandies[1]->SetDestination(clickedCandies[0]->GetCurrentX(), clickedCandies[0]->GetCurrentY());

Candy temp = \*clickedCandies[0];

\*clickedCandies[0] = \*clickedCandies[1];

\*clickedCandies[1] = temp;

}

bool GameArea::IsNeighbour(Candy &a, Candy &b)

{

bool vertiNeighbour = fabs(a.GetTopLeftX() - b.GetTopLeftX()) == 50 && a.GetTopLeftY() == b.GetTopLeftY();

bool horztNeighbour = fabs(a.GetTopLeftY() - b.GetTopLeftY()) == 50 && a.GetTopLeftX() == b.GetTopLeftX();

return vertiNeighbour || horztNeighbour;

}

void GameArea::OnMoveEnding()

{

bool result = scoreBoard.IsReachedTarget() || (scoreBoard.score > scoreBoard.oneStar && scoreBoard.mode == 1); //Win or lose

// turn all moving step to bonus and release all powered candy if target reached

if (!ending && (!scoreBoard.moves.GetInteger() || scoreBoard.IsReachedTarget()))

{ // change current state to ending

if (result) PlayVoiceEffect(AUDIO\_SUGAR\_CRUSH);

PutEndingBonus();

ending = true;

}

else if (ending && running)

{

if (endingBonus.size())

{ // Show bonus powered candy

(\*endingBonus.begin())->SetPower(rand() % 2 + 1);

endingBonus.pop\_front();

scoreBoard.moves--;

Sleep(100);

}

else if (result)

{ // remove all powered candy until no powered candy spwan if player win this game

removeList.push\_back(new list<Candy\*>);

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

{ //collect all powered candy

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

{

if (candies[i][j].GetStyle() > 0 && candies[i][j].GetPower() > 0)

(\*removeList.rbegin())->push\_back(&candies[i][j]);

}

}

if (!removeList.back()->size())

{ // to avoid memory leak, delete removelist if no powered candy found

delete (\*removeList.rbegin());

removeList.pop\_back();

}

if (!removeList.size() && !gameOver)

{ // change state to gameOver

delay = (int)(700.0 / GAME\_CYCLE\_TIME);

gameOver = true;

}

}

else if (!result && !gameOver)

{ // change state to gameOver

delay = (int)(700.0 / GAME\_CYCLE\_TIME);

gameOver = true;

}

if (gameOver) GotoGameStateOver(result);

}

}

void GameArea::PlayVoiceEffect(int audio\_id)

{

static double\* size; //current size of word

static int\* frame; //current frame of animation

static CMovingBitmap\* word;

if (audio\_id != -1) GetWordBmp(&size, &frame, &word, audio\_id); // reset animation's size, frame & bitmap

if (!playingVoice) return;

//Show words from small to big at the center

word->SetTopLeft(SIZE\_X / 2 - (int)(word->Width() \* \*size) / 2, SIZE\_Y / 2 - (int)(word->Height() \* \*size) / 2);

word->ShowBitmap(\*size);

if (\*size < 1) \*size += 0.1;

if ((\*frame)++ > 30)

{

delete size;

delete frame;

playingVoice = false;

}

if (!(\*sound) || (\*frame) != 1) return;

CAudio::Instance()->Play(audio\_id, false);

}

void GameArea::TeleportCandy()

{

vector<pair<CPoint, CPoint>>\* portalList = &(\*stage)->portalList; // get portal position

#pragma omp parallel for

for (int i = 0; i < (int)portalList->size(); i++)

{

Candy\* from = curPosition[(\*portalList)[i].first.y][(\*portalList)[i].first.x]; //get candy of starting point

Candy\* to = curPosition[(\*portalList)[i].second.y][(\*portalList)[i].second.x]; //get candy of end

if (from != NULL && !(from->IsMoving()) && to == NULL)

{ //teleport if there is a still candy at starting point and no candy at the end

from->SetDestination(from->GetCurrentY() + 24); //drop into portal

//create a candy come out from portal

candies[(\*portalList)[i].second.y][(\*portalList)[i].second.x] = Candy(from->GetStyle(), (\*portalList)[i].second.x \* 50 + x, (\*portalList)[i].second.y \* 50 + y - 25);

candies[(\*portalList)[i].second.y][(\*portalList)[i].second.x].SetDestination((\*portalList)[i].second.y \* 50 + y);

candies[(\*portalList)[i].second.y][(\*portalList)[i].second.x].SetPower(from->GetPower());

candies[(\*portalList)[i].second.y][(\*portalList)[i].second.x].SetStyle(from->GetStyle());

}

if (from != NULL && from->GetCurrentY() >= (\*portalList)[i].first.y \* 50 + y + 24)

from->Kill(); //candy disappear when drop into portal

}

}

void GameArea::InitClickedCandy()

{

for (auto i = clickedCandies.begin(); i != clickedCandies.end(); i++)

{

(\*i)->InitClick();

}

clickedCandies.clear();

}

void GameArea::InitGameArea(int totalJelly)

{

//Set Target

if (scoreBoard.mode == 1)

{

if (scoreBoard.lastHighScore < scoreBoard.oneStar)

scoreBoard.target = (int)scoreBoard.oneStar;

else if (scoreBoard.lastHighScore < scoreBoard.twoStar)

scoreBoard.target = (int)scoreBoard.twoStar;

else if (scoreBoard.lastHighScore < scoreBoard.threeStar)

scoreBoard.target = (int)scoreBoard.threeStar;

else scoreBoard.target = (int)scoreBoard.lastHighScore;

}

else if (scoreBoard.mode == 2)

{

scoreBoard.target = totalJelly;

}

else GAME\_ASSERT(0, "Game mode unrecognizable!")

scoreBoard.score = 0;

initiating = true;

ending = false;

running = true;

gameOver = false;

delay = 0;

delayRemove = false;

releaseSwap = false;

goldFinger = false;

}

}

#ifndef CANDY\_H

#define CANDY\_H

namespace game\_framework

{

class Candy

{

public:

Candy(int, int, int);

Candy(int, int);

Candy();

~Candy();

Candy\* Click(); //Change state of onClick if being clicked

int GetCurrentX(); //Get current x

int GetCurrentY(); //Get current y

int GetPower(); //return current power

int GetStyle(); //get current style

int GetTopLeftX(); //Get destination x

int GetTopLeftY(); //Get destination y

int GetTopLeft(char c); //Get destination(c) 'x'/'y'

void InitClick(); //Unclick self

bool IsMoving(); //return current coordinate != destination coordinate

bool IsClicked(); //return onClick

void Kill(); //simple remove without release power

static void LoadBitmap(); //Load candy's bitmap according to style

void OnMove(bool mute = false); //move if current coordinate != destination coordinate

void OnShow(); //display candy

void Push(int, int); //Push candies around self

void Relive(); //reset candy's style

void SetDestination(int, int); //set destination x,y

void SetDestination(int); //set destination y

void SetStyle(int); //change current style

void SetPower(int); //Set current power

private:

void GetCurrentShow(CMovingBitmap\*\*, CMovingBitmap\*\*);

static CMovingBitmap normal[6], normalClick[6];

static CMovingBitmap horizon[6], horizonClick[6];

static CMovingBitmap vertical[6], verticalClick[6];

static CMovingBitmap pack[6], packClick[6];

static CMovingBitmap super[6], superClick[6];

bool \*sound;

int style, rawStyle; //0 = non-exist

int power;

int x, y, dx, dy, pushX, pushY; //current || destination coordinate

bool onClick;

int fallingSpeed;

};

}

#endif

#include "stdafx.h"

#include "Resource.h"

#include <mmsystem.h>

#include <ddraw.h>

#include "audio.h"

#include "gamelib.h"

#include "Candy.h"

namespace game\_framework

{

static int redBmpId[8] = {

IDB\_RED\_NOR, IDB\_RED\_NOR\_C, IDB\_RED\_HOR, IDB\_RED\_HOR\_C, IDB\_RED\_VER, IDB\_RED\_VER\_C, IDB\_RED\_PAC, IDB\_RED\_PAC\_C },

orangeBmpId[8] = {

IDB\_ORG\_NOR, IDB\_ORG\_NOR\_C, IDB\_ORG\_HOR, IDB\_ORG\_HOR\_C, IDB\_ORG\_VER, IDB\_ORG\_VER\_C, IDB\_ORG\_PAC, IDB\_ORG\_PAC\_C },

yellowBmpId[8] = {

IDB\_YEL\_NOR, IDB\_YEL\_NOR\_C, IDB\_YEL\_HOR, IDB\_YEL\_HOR\_C, IDB\_YEL\_VER, IDB\_YEL\_VER\_C, IDB\_YEL\_PAC, IDB\_YEL\_PAC\_C },

greenBmpId[8] = {

IDB\_GRE\_NOR, IDB\_GRE\_NOR\_C, IDB\_GRE\_HOR, IDB\_GRE\_HOR\_C, IDB\_GRE\_VER, IDB\_GRE\_VER\_C, IDB\_GRE\_PAC, IDB\_GRE\_PAC\_C },

blueBmpId[8] = {

IDB\_BLU\_NOR, IDB\_BLU\_NOR\_C, IDB\_BLU\_HOR, IDB\_BLU\_HOR\_C, IDB\_BLU\_VER, IDB\_BLU\_VER\_C, IDB\_BLU\_PAC, IDB\_BLU\_PAC\_C },

purpleBmpId[8] = {

IDB\_PUR\_NOR, IDB\_PUR\_NOR\_C, IDB\_PUR\_HOR, IDB\_PUR\_HOR\_C, IDB\_PUR\_VER, IDB\_PUR\_VER\_C, IDB\_PUR\_PAC, IDB\_PUR\_PAC\_C },

audioID[4] = {

AUDIO\_CANDY\_LAND1, AUDIO\_CANDY\_LAND2, AUDIO\_CANDY\_LAND3, AUDIO\_CANDY\_LAND4 };

CMovingBitmap

Candy::normal[6], Candy::normalClick[6],

Candy::horizon[6], Candy::horizonClick[6],

Candy::vertical[6], Candy::verticalClick[6],

Candy::pack[6], Candy::packClick[6],

Candy::super[6], Candy::superClick[6];

Candy::Candy(int id, int x, int y)

: style(id), rawStyle(id), x(x), y(y), dx(x), dy(y), onClick(false), fallingSpeed(0), power(0),

pushX(0), pushY(0), sound(&CGameState::sound)

{ }

Candy::Candy(int x, int y)

: style(-1), x(x), y(y), dx(x), dy(y), sound(&CGameState::sound)

{ }

Candy::Candy()

: style(-1), sound(&CGameState::sound)

{ }

Candy::~Candy()

{}

void Candy::LoadBitmap()

{

int \*bmpID[] = { blueBmpId, purpleBmpId, orangeBmpId, greenBmpId , redBmpId, yellowBmpId };

for (int i = 0; i < 6; i++)

{

normal[i].LoadBitmap(bmpID[i][0], RGB(255, 255, 255));

normalClick[i].LoadBitmap(bmpID[i][1], RGB(255, 255, 255));

horizon[i].LoadBitmap(bmpID[i][2], RGB(255, 255, 255));

horizonClick[i].LoadBitmap(bmpID[i][3], RGB(255, 255, 255));

vertical[i].LoadBitmap(bmpID[i][4], RGB(255, 255, 255));

verticalClick[i].LoadBitmap(bmpID[i][5], RGB(255, 255, 255));

pack[i].LoadBitmap(bmpID[i][6], RGB(255, 255, 255));

packClick[i].LoadBitmap(bmpID[i][7], RGB(255, 255, 255));

super[i].LoadBitmap(IDB\_SUPER, RGB(255, 255, 255));

superClick[i].LoadBitmap(IDB\_SUPER\_C, RGB(255, 255, 255));

}

}

void Candy::OnMove(bool mute)

{

if (pushX != 0 || pushY != 0)

{

int offsetX = pushX > 0 ? 1 : -1, offsetY = pushY > 0 ? 1 : -1;

if (pushX != 0)

{

x += offsetX;

pushX -= offsetX;

}

if (pushY != 0)

{

y += offsetY;

pushY -= offsetY;

}

return;

}

int fixedSpeed = GAME\_CYCLE\_TIME \* 5 / 16;

if (y != dy && x != dx)

{

y < dy ? y += fixedSpeed : y -= fixedSpeed;

x < dx ? x += fixedSpeed : x -= fixedSpeed;

}

else if (x != dx)

x > dx ? x -= fixedSpeed : x += fixedSpeed;

else if (y < dy)

{

y += fallingSpeed;

fallingSpeed += GAME\_CYCLE\_TIME \* 2 / 16;

if (y > dy)

{

y = dy;

fallingSpeed = 0;

int landingSound = rand() % 4;

if (!mute && \*sound) CAudio::Instance()->Play(audioID[landingSound], false);

}

}

y > dy ? y -= fixedSpeed \* 2 : y;

}

void Candy::OnShow()

{

if (!style) return;

CMovingBitmap \*idle, \*click;

GetCurrentShow(&idle, &click);

if (!onClick)

{

idle->SetTopLeft(x, y);

idle->ShowBitmap();

}

else

{

click->SetTopLeft(x, y);

click->ShowBitmap();

}

}

void Candy::SetDestination(int x, int y)

{

dx = x;

dy = y;

}

void Candy::SetDestination(int y)

{

dy = y;

}

int Candy::GetStyle()

{

return style;

}

void Candy::SetStyle(int style)

{

this->style = style;

}

int Candy::GetTopLeftX()

{

return dx;

}

int Candy::GetTopLeftY()

{

return dy;

}

int Candy::GetTopLeft(char c)

{

switch (c)

{

case 'x': return GetTopLeftX();

case 'y': return GetTopLeftY();

default: return 0;

}

}

int Candy::GetCurrentX()

{

return x;

}

int Candy::GetCurrentY()

{

return y;

}

bool Candy::IsMoving()

{

if (style <= 0) return 0;

return x != dx || y != dy;

}

bool Candy::IsClicked()

{

return onClick;

}

void Candy::Kill()

{

power = 0;

style = 0;

}

Candy\* Candy::Click()

{

onClick = onClick == true ? false : true;

return this;

}

void Candy::SetPower(int power)

{

this->power = power;

}

int Candy::GetPower()

{

return power;

}

void Candy::Relive()

{

style = rawStyle;

}

void Candy::InitClick()

{

onClick = false;

}

void Candy::Push(int x, int y)

{

this->x = x > 0 ? this->x + 20 : x < 0 ? this->x - 20 : this->x;

this->y = y > 0 ? this->y + 20 : y < 0 ? this->y - 20 : this->y;

}

void Candy::GetCurrentShow(CMovingBitmap \*\*idle, CMovingBitmap \*\*click)

{

switch (power)

{

case 0:

\*idle = &normal[style - 1];

\*click = &normalClick[style - 1];

break;

case 1:

\*idle = &horizon[style - 1];

\*click = &horizonClick[style - 1];

break;

case 2:

\*idle = &vertical[style - 1];

\*click = &verticalClick[style - 1];

break;

case 3:

\*idle = &pack[style - 1];

\*click = &packClick[style - 1];

break;

case 4:

\*idle = &super[style - 1];

\*click = &superClick[style - 1];

break;

}

}

}

class Blast

{

public:

virtual ~Blast() {};

virtual void LoadBitmap() {};

virtual void OnShow() {};

virtual void OnMove() {};

virtual void SetTopLeft(int x, int y);

virtual bool IsLast() = 0;

protected:

int x, y;

int style;

bool \*sound;

};

static int redBmp[] = {

IDB\_RED\_EXPL1, IDB\_RED\_EXPL2, IDB\_RED\_EXPL3, IDB\_RED\_EXPL4, IDB\_RED\_EXPL5,

IDB\_RED\_EXPL6, IDB\_RED\_EXPL7, IDB\_RED\_EXPL8, IDB\_RED\_EXPL9, IDB\_RED\_EXPL10,

IDB\_RED\_SHAT1, IDB\_RED\_SHAT2, IDB\_RED\_SHAT3, IDB\_RED\_SHAT4, IDB\_RED\_SHAT5,

IDB\_RED\_SHAT6, IDB\_RED\_SHAT7, IDB\_RED\_SHAT8, IDB\_RED\_SHAT9, IDB\_RED\_SHAT10,

IDB\_RED\_SHAT11, IDB\_RED\_SHAT12, IDB\_RED\_SHAT13, IDB\_RED\_SHAT14, IDB\_RED\_SHAT15,

IDB\_RED\_VEXPL1, IDB\_RED\_VEXPL2, IDB\_RED\_VEXPL3, IDB\_RED\_VEXPL4, IDB\_RED\_VEXPL5,

IDB\_RED\_VEXPL6, IDB\_RED\_VEXPL7, IDB\_RED\_VEXPL8, IDB\_RED\_VEXPL9, IDB\_RED\_VEXPL10,

IDB\_RED\_VEXPL11, IDB\_RED\_VEXPL12, IDB\_RED\_VEXPL13, IDB\_RED\_VEXPL14, IDB\_RED\_VEXPL15,

IDB\_RED\_VEXPL16, IDB\_RED\_VEXPL17, IDB\_RED\_VEXPL18, IDB\_RED\_VEXPL19, IDB\_RED\_VEXPL20,

IDB\_RED\_VEXPL21, IDB\_RED\_VEXPL22, IDB\_RED\_VEXPL23, IDB\_RED\_VEXPL24, IDB\_RED\_VEXPL25,

IDB\_RED\_VEXPL26, IDB\_RED\_VEXPL27, IDB\_RED\_VEXPL28, IDB\_RED\_VEXPL29, IDB\_RED\_VEXPL30,

IDB\_RED\_HEXPL1, IDB\_RED\_HEXPL2, IDB\_RED\_HEXPL3, IDB\_RED\_HEXPL4, IDB\_RED\_HEXPL5,

IDB\_RED\_HEXPL6, IDB\_RED\_HEXPL7, IDB\_RED\_HEXPL8, IDB\_RED\_HEXPL9, IDB\_RED\_HEXPL10,

IDB\_RED\_HEXPL11, IDB\_RED\_HEXPL12, IDB\_RED\_HEXPL13, IDB\_RED\_HEXPL14, IDB\_RED\_HEXPL15,

IDB\_RED\_HEXPL16, IDB\_RED\_HEXPL17, IDB\_RED\_HEXPL18, IDB\_RED\_HEXPL19, IDB\_RED\_HEXPL20,

IDB\_RED\_HEXPL21, IDB\_RED\_HEXPL22, IDB\_RED\_HEXPL23, IDB\_RED\_HEXPL24, IDB\_RED\_HEXPL25,

IDB\_RED\_HEXPL26, IDB\_RED\_HEXPL27, IDB\_RED\_HEXPL28, IDB\_RED\_HEXPL29, IDB\_RED\_HEXPL30 },

orangeBmp[] = {

IDB\_ORG\_EXPL1, IDB\_ORG\_EXPL2, IDB\_ORG\_EXPL3, IDB\_ORG\_EXPL4, IDB\_ORG\_EXPL5,

IDB\_ORG\_EXPL6, IDB\_ORG\_EXPL7, IDB\_ORG\_EXPL8, IDB\_ORG\_EXPL9, IDB\_ORG\_EXPL10,

IDB\_ORG\_SHAT1, IDB\_ORG\_SHAT2, IDB\_ORG\_SHAT3, IDB\_ORG\_SHAT4, IDB\_ORG\_SHAT5,

IDB\_ORG\_SHAT6, IDB\_ORG\_SHAT7, IDB\_ORG\_SHAT8, IDB\_ORG\_SHAT9, IDB\_ORG\_SHAT10,

IDB\_ORG\_SHAT11, IDB\_ORG\_SHAT12, IDB\_ORG\_SHAT13, IDB\_ORG\_SHAT14, IDB\_ORG\_SHAT15,

IDB\_ORG\_VEXPL1, IDB\_ORG\_VEXPL2, IDB\_ORG\_VEXPL3, IDB\_ORG\_VEXPL4, IDB\_ORG\_VEXPL5,

IDB\_ORG\_VEXPL6, IDB\_ORG\_VEXPL7, IDB\_ORG\_VEXPL8, IDB\_ORG\_VEXPL9, IDB\_ORG\_VEXPL10,

IDB\_ORG\_VEXPL11, IDB\_ORG\_VEXPL12, IDB\_ORG\_VEXPL13, IDB\_ORG\_VEXPL14, IDB\_ORG\_VEXPL15,

IDB\_ORG\_VEXPL16, IDB\_ORG\_VEXPL17, IDB\_ORG\_VEXPL18, IDB\_ORG\_VEXPL19, IDB\_ORG\_VEXPL20,

IDB\_ORG\_VEXPL21, IDB\_ORG\_VEXPL22, IDB\_ORG\_VEXPL23, IDB\_ORG\_VEXPL24, IDB\_ORG\_VEXPL25,

IDB\_ORG\_VEXPL26, IDB\_ORG\_VEXPL27, IDB\_ORG\_VEXPL28, IDB\_ORG\_VEXPL29, IDB\_ORG\_VEXPL30,

IDB\_ORG\_HEXPL1, IDB\_ORG\_HEXPL2, IDB\_ORG\_HEXPL3, IDB\_ORG\_HEXPL4, IDB\_ORG\_HEXPL5,

IDB\_ORG\_HEXPL6, IDB\_ORG\_HEXPL7, IDB\_ORG\_HEXPL8, IDB\_ORG\_HEXPL9, IDB\_ORG\_HEXPL10,

IDB\_ORG\_HEXPL11, IDB\_ORG\_HEXPL12, IDB\_ORG\_HEXPL13, IDB\_ORG\_HEXPL14, IDB\_ORG\_HEXPL15,

IDB\_ORG\_HEXPL16, IDB\_ORG\_HEXPL17, IDB\_ORG\_HEXPL18, IDB\_ORG\_HEXPL19, IDB\_ORG\_HEXPL20,

IDB\_ORG\_HEXPL21, IDB\_ORG\_HEXPL22, IDB\_ORG\_HEXPL23, IDB\_ORG\_HEXPL24, IDB\_ORG\_HEXPL25,

IDB\_ORG\_HEXPL26, IDB\_ORG\_HEXPL27, IDB\_ORG\_HEXPL28, IDB\_ORG\_HEXPL29, IDB\_ORG\_HEXPL30 },

yellowBmp[] = {

IDB\_YEL\_EXPL1, IDB\_YEL\_EXPL2, IDB\_YEL\_EXPL3, IDB\_YEL\_EXPL4, IDB\_YEL\_EXPL5,

IDB\_YEL\_EXPL6, IDB\_YEL\_EXPL7, IDB\_YEL\_EXPL8, IDB\_YEL\_EXPL9, IDB\_YEL\_EXPL10,

IDB\_YEL\_SHAT1, IDB\_YEL\_SHAT2, IDB\_YEL\_SHAT3, IDB\_YEL\_SHAT4, IDB\_YEL\_SHAT5,

IDB\_YEL\_SHAT6, IDB\_YEL\_SHAT7, IDB\_YEL\_SHAT8, IDB\_YEL\_SHAT9, IDB\_YEL\_SHAT10,

IDB\_YEL\_SHAT11, IDB\_YEL\_SHAT12, IDB\_YEL\_SHAT13, IDB\_YEL\_SHAT14, IDB\_YEL\_SHAT15,

IDB\_YEL\_VEXPL1, IDB\_YEL\_VEXPL2, IDB\_YEL\_VEXPL3, IDB\_YEL\_VEXPL4, IDB\_YEL\_VEXPL5,

IDB\_YEL\_VEXPL6, IDB\_YEL\_VEXPL7, IDB\_YEL\_VEXPL8, IDB\_YEL\_VEXPL9, IDB\_YEL\_VEXPL10,

IDB\_YEL\_VEXPL11, IDB\_YEL\_VEXPL12, IDB\_YEL\_VEXPL13, IDB\_YEL\_VEXPL14, IDB\_YEL\_VEXPL15,

IDB\_YEL\_VEXPL16, IDB\_YEL\_VEXPL17, IDB\_YEL\_VEXPL18, IDB\_YEL\_VEXPL19, IDB\_YEL\_VEXPL20,

IDB\_YEL\_VEXPL21, IDB\_YEL\_VEXPL22, IDB\_YEL\_VEXPL23, IDB\_YEL\_VEXPL24, IDB\_YEL\_VEXPL25,

IDB\_YEL\_VEXPL26, IDB\_YEL\_VEXPL27, IDB\_YEL\_VEXPL28, IDB\_YEL\_VEXPL29, IDB\_YEL\_VEXPL30,

IDB\_YEL\_HEXPL1, IDB\_YEL\_HEXPL2, IDB\_YEL\_HEXPL3, IDB\_YEL\_HEXPL4, IDB\_YEL\_HEXPL5,

IDB\_YEL\_HEXPL6, IDB\_YEL\_HEXPL7, IDB\_YEL\_HEXPL8, IDB\_YEL\_HEXPL9, IDB\_YEL\_HEXPL10,

IDB\_YEL\_HEXPL11, IDB\_YEL\_HEXPL12, IDB\_YEL\_HEXPL13, IDB\_YEL\_HEXPL14, IDB\_YEL\_HEXPL15,

IDB\_YEL\_HEXPL16, IDB\_YEL\_HEXPL17, IDB\_YEL\_HEXPL18, IDB\_YEL\_HEXPL19, IDB\_YEL\_HEXPL20,

IDB\_YEL\_HEXPL21, IDB\_YEL\_HEXPL22, IDB\_YEL\_HEXPL23, IDB\_YEL\_HEXPL24, IDB\_YEL\_HEXPL25,

IDB\_YEL\_HEXPL26, IDB\_YEL\_HEXPL27, IDB\_YEL\_HEXPL28, IDB\_YEL\_HEXPL29, IDB\_YEL\_HEXPL30 },

greenBmp[] = {

IDB\_GRE\_EXPL1, IDB\_GRE\_EXPL2, IDB\_GRE\_EXPL3, IDB\_GRE\_EXPL4, IDB\_GRE\_EXPL5,

IDB\_GRE\_EXPL6, IDB\_GRE\_EXPL7, IDB\_GRE\_EXPL8, IDB\_GRE\_EXPL9, IDB\_GRE\_EXPL10,

IDB\_GRE\_SHAT1, IDB\_GRE\_SHAT2, IDB\_GRE\_SHAT3, IDB\_GRE\_SHAT4, IDB\_GRE\_SHAT5,

IDB\_GRE\_SHAT6, IDB\_GRE\_SHAT7, IDB\_GRE\_SHAT8, IDB\_GRE\_SHAT9, IDB\_GRE\_SHAT10,

IDB\_GRE\_SHAT11, IDB\_GRE\_SHAT12, IDB\_GRE\_SHAT13, IDB\_GRE\_SHAT14, IDB\_GRE\_SHAT15,

IDB\_GRE\_VEXPL1, IDB\_GRE\_VEXPL2, IDB\_GRE\_VEXPL3, IDB\_GRE\_VEXPL4, IDB\_GRE\_VEXPL5,

IDB\_GRE\_VEXPL6, IDB\_GRE\_VEXPL7, IDB\_GRE\_VEXPL8, IDB\_GRE\_VEXPL9, IDB\_GRE\_VEXPL10,

IDB\_GRE\_VEXPL11, IDB\_GRE\_VEXPL12, IDB\_GRE\_VEXPL13, IDB\_GRE\_VEXPL14, IDB\_GRE\_VEXPL15,

IDB\_GRE\_VEXPL16, IDB\_GRE\_VEXPL17, IDB\_GRE\_VEXPL18, IDB\_GRE\_VEXPL19, IDB\_GRE\_VEXPL20,

IDB\_GRE\_VEXPL21, IDB\_GRE\_VEXPL22, IDB\_GRE\_VEXPL23, IDB\_GRE\_VEXPL24, IDB\_GRE\_VEXPL25,

IDB\_GRE\_VEXPL26, IDB\_GRE\_VEXPL27, IDB\_GRE\_VEXPL28, IDB\_GRE\_VEXPL29, IDB\_GRE\_VEXPL30,

IDB\_GRE\_HEXPL1, IDB\_GRE\_HEXPL2, IDB\_GRE\_HEXPL3, IDB\_GRE\_HEXPL4, IDB\_GRE\_HEXPL5,

IDB\_GRE\_HEXPL6, IDB\_GRE\_HEXPL7, IDB\_GRE\_HEXPL8, IDB\_GRE\_HEXPL9, IDB\_GRE\_HEXPL10,

IDB\_GRE\_HEXPL11, IDB\_GRE\_HEXPL12, IDB\_GRE\_HEXPL13, IDB\_GRE\_HEXPL14, IDB\_GRE\_HEXPL15,

IDB\_GRE\_HEXPL16, IDB\_GRE\_HEXPL17, IDB\_GRE\_HEXPL18, IDB\_GRE\_HEXPL19, IDB\_GRE\_HEXPL20,

IDB\_GRE\_HEXPL21, IDB\_GRE\_HEXPL22, IDB\_GRE\_HEXPL23, IDB\_GRE\_HEXPL24, IDB\_GRE\_HEXPL25,

IDB\_GRE\_HEXPL26, IDB\_GRE\_HEXPL27, IDB\_GRE\_HEXPL28, IDB\_GRE\_HEXPL29, IDB\_GRE\_HEXPL30 },

blueBmp[] = {

IDB\_BLU\_EXPL1, IDB\_BLU\_EXPL2, IDB\_BLU\_EXPL3, IDB\_BLU\_EXPL4, IDB\_BLU\_EXPL5,

IDB\_BLU\_EXPL6, IDB\_BLU\_EXPL7, IDB\_BLU\_EXPL8, IDB\_BLU\_EXPL9, IDB\_BLU\_EXPL10,

IDB\_BLU\_SHAT1, IDB\_BLU\_SHAT2, IDB\_BLU\_SHAT3, IDB\_BLU\_SHAT4, IDB\_BLU\_SHAT5,

IDB\_BLU\_SHAT6, IDB\_BLU\_SHAT7, IDB\_BLU\_SHAT8, IDB\_BLU\_SHAT9, IDB\_BLU\_SHAT10,

IDB\_BLU\_SHAT11, IDB\_BLU\_SHAT12, IDB\_BLU\_SHAT13, IDB\_BLU\_SHAT14, IDB\_BLU\_SHAT15,

IDB\_BLU\_VEXPL1, IDB\_BLU\_VEXPL2, IDB\_BLU\_VEXPL3, IDB\_BLU\_VEXPL4, IDB\_BLU\_VEXPL5,

IDB\_BLU\_VEXPL6, IDB\_BLU\_VEXPL7, IDB\_BLU\_VEXPL8, IDB\_BLU\_VEXPL9, IDB\_BLU\_VEXPL10,

IDB\_BLU\_VEXPL11, IDB\_BLU\_VEXPL12, IDB\_BLU\_VEXPL13, IDB\_BLU\_VEXPL14, IDB\_BLU\_VEXPL15,

IDB\_BLU\_VEXPL16, IDB\_BLU\_VEXPL17, IDB\_BLU\_VEXPL18, IDB\_BLU\_VEXPL19, IDB\_BLU\_VEXPL20,

IDB\_BLU\_VEXPL21, IDB\_BLU\_VEXPL22, IDB\_BLU\_VEXPL23, IDB\_BLU\_VEXPL24, IDB\_BLU\_VEXPL25,

IDB\_BLU\_VEXPL26, IDB\_BLU\_VEXPL27, IDB\_BLU\_VEXPL28, IDB\_BLU\_VEXPL29, IDB\_BLU\_VEXPL30,

IDB\_BLU\_HEXPL1, IDB\_BLU\_HEXPL2, IDB\_BLU\_HEXPL3, IDB\_BLU\_HEXPL4, IDB\_BLU\_HEXPL5,

IDB\_BLU\_HEXPL6, IDB\_BLU\_HEXPL7, IDB\_BLU\_HEXPL8, IDB\_BLU\_HEXPL9, IDB\_BLU\_HEXPL10,

IDB\_BLU\_HEXPL11, IDB\_BLU\_HEXPL12, IDB\_BLU\_HEXPL13, IDB\_BLU\_HEXPL14, IDB\_BLU\_HEXPL15,

IDB\_BLU\_HEXPL16, IDB\_BLU\_HEXPL17, IDB\_BLU\_HEXPL18, IDB\_BLU\_HEXPL19, IDB\_BLU\_HEXPL20,

IDB\_BLU\_HEXPL21, IDB\_BLU\_HEXPL22, IDB\_BLU\_HEXPL23, IDB\_BLU\_HEXPL24, IDB\_BLU\_HEXPL25,

IDB\_BLU\_HEXPL26, IDB\_BLU\_HEXPL27, IDB\_BLU\_HEXPL28, IDB\_BLU\_HEXPL29, IDB\_BLU\_HEXPL30 },

purpleBmp[] = {

IDB\_PUR\_EXPL1, IDB\_PUR\_EXPL2, IDB\_PUR\_EXPL3, IDB\_PUR\_EXPL4, IDB\_PUR\_EXPL5,

IDB\_PUR\_EXPL6, IDB\_PUR\_EXPL7, IDB\_PUR\_EXPL8, IDB\_PUR\_EXPL9, IDB\_PUR\_EXPL10,

IDB\_PUR\_SHAT1, IDB\_PUR\_SHAT2, IDB\_PUR\_SHAT3, IDB\_PUR\_SHAT4, IDB\_PUR\_SHAT5,

IDB\_PUR\_SHAT6, IDB\_PUR\_SHAT7, IDB\_PUR\_SHAT8, IDB\_PUR\_SHAT9, IDB\_PUR\_SHAT10,

IDB\_PUR\_SHAT11, IDB\_PUR\_SHAT12, IDB\_PUR\_SHAT13, IDB\_PUR\_SHAT14, IDB\_PUR\_SHAT15,

IDB\_PUR\_VEXPL1, IDB\_PUR\_VEXPL2, IDB\_PUR\_VEXPL3, IDB\_PUR\_VEXPL4, IDB\_PUR\_VEXPL5,

IDB\_PUR\_VEXPL6, IDB\_PUR\_VEXPL7, IDB\_PUR\_VEXPL8, IDB\_PUR\_VEXPL9, IDB\_PUR\_VEXPL10,

IDB\_PUR\_VEXPL11, IDB\_PUR\_VEXPL12, IDB\_PUR\_VEXPL13, IDB\_PUR\_VEXPL14, IDB\_PUR\_VEXPL15,

IDB\_PUR\_VEXPL16, IDB\_PUR\_VEXPL17, IDB\_PUR\_VEXPL18, IDB\_PUR\_VEXPL19, IDB\_PUR\_VEXPL20,

IDB\_PUR\_VEXPL21, IDB\_PUR\_VEXPL22, IDB\_PUR\_VEXPL23, IDB\_PUR\_VEXPL24, IDB\_PUR\_VEXPL25,

IDB\_PUR\_VEXPL26, IDB\_PUR\_VEXPL27, IDB\_PUR\_VEXPL28, IDB\_PUR\_VEXPL29, IDB\_PUR\_VEXPL30,

IDB\_PUR\_HEXPL1, IDB\_PUR\_HEXPL2, IDB\_PUR\_HEXPL3, IDB\_PUR\_HEXPL4, IDB\_PUR\_HEXPL5,

IDB\_PUR\_HEXPL6, IDB\_PUR\_HEXPL7, IDB\_PUR\_HEXPL8, IDB\_PUR\_HEXPL9, IDB\_PUR\_HEXPL10,

IDB\_PUR\_HEXPL11, IDB\_PUR\_HEXPL12, IDB\_PUR\_HEXPL13, IDB\_PUR\_HEXPL14, IDB\_PUR\_HEXPL15,

IDB\_PUR\_HEXPL16, IDB\_PUR\_HEXPL17, IDB\_PUR\_HEXPL18, IDB\_PUR\_HEXPL19, IDB\_PUR\_HEXPL20,

IDB\_PUR\_HEXPL21, IDB\_PUR\_HEXPL22, IDB\_PUR\_HEXPL23, IDB\_PUR\_HEXPL24, IDB\_PUR\_HEXPL25,

IDB\_PUR\_HEXPL26, IDB\_PUR\_HEXPL27, IDB\_PUR\_HEXPL28, IDB\_PUR\_HEXPL29, IDB\_PUR\_HEXPL30 };

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

{

this->x = x;

this->y = y;

}

class NormalBlast :public Blast

{

public:

NormalBlast();

NormalBlast(int style, int x, int y);

static void LoadBitmap();

void OnMove();

void OnShow();

bool IsLast();

private:

static CMovingBitmap normalBlast[6][10];

static CMovingBitmap shatter[6][15];

int curShow;

int shift[3][3]; //move direction/ moving speed/ spin direction

int shatPosition[3][2]; //every shatter current position

int shatShow[3]; //shatter current frame

const int totalShow; //total shatter spawned

double size; //current size

};

CMovingBitmap NormalBlast::normalBlast[6][10], NormalBlast::shatter[6][15];

NormalBlast::NormalBlast() :curShow(0), totalShow(0)

{

sound = &CGameState::sound;

}

NormalBlast::NormalBlast(int style, int x, int y) :curShow(0), size(1.8), totalShow(rand() % 2 + 2)

{

this->style = style;

SetTopLeft(x, y);

int direction[] = { -2, -1, 0, 2, 1 }; //direction & speed for shatter

#pragma omp parallel for

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

{

//init shatters' position at center

shatPosition[i][0] = x;

shatPosition[i][1] = y;

shatShow[i] = rand() % 15; //set shatter start frame

shift[i][0] = direction[rand() % 5]; //random direction x

shift[i][1] = direction[rand() % 2 + 3];//random dirention y

shift[i][2] = rand() % 2; //set shatter rotation { 1 = clockwise, 0 = counter-clockwise}

}

}

void NormalBlast::LoadBitmap()

{

int \*bmpID[] = { blueBmp, purpleBmp, orangeBmp, greenBmp , redBmp, yellowBmp};

for (int i = 0; i < 6; i++)

{

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

normalBlast[i][j].LoadBitmap(bmpID[i][j], RGB(254, 191, 200));

for (int j = 0; j < 15; j++)

shatter[i][j].LoadBitmap(bmpID[i][j + 10], RGB(254, 191, 200));

}

}

void NormalBlast::OnMove()

{

curShow++;

if(curShow % 2) size -= 0.1; //zoom-out shatter

#pragma omp parallel for

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

{ //move shatters

shatPosition[i][0] += shift[i][0];

shatPosition[i][1] += shift[i][1];

//rotate shatters

if (!(curShow % 3))

{

if(shift[i][2]) shatShow[i] = (shatShow[i] - 1) < 0 ? (shatShow[i] - 1) + 15 : (shatShow[i] - 1); //counter-clockwise

else shatShow[i] = (shatShow[i] + 1) % 15; //clockwise

}

}

}

void NormalBlast::OnShow()

{

if (curShow < 10)

{ //show blast circle

normalBlast[style - 1][curShow].SetTopLeft(x - (normalBlast[style - 1][curShow].Width() / 2) + 25, y - (normalBlast[style - 1][curShow].Height() / 2) + 25);

normalBlast[style - 1][curShow].ShowBitmap();

}

if (curShow >= 4)

{ //show shatters

#pragma omp parallel for

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

{

shatter[style - 1][shatShow[i]].SetTopLeft(shatPosition[i][0], shatPosition[i][1]);

shatter[style - 1][shatShow[i]].ShowBitmap(size);

}

}

}

bool NormalBlast::IsLast()

{

return (curShow == 25);

}

class LineBlast :public Blast

{

public:

LineBlast(int style, int x, int y, int power);

static void LoadBitmap();

void OnMove();

void OnShow();

bool IsLast();

private:

static CMovingBitmap horizontal[6][30];

static CMovingBitmap vertical[6][30];

int curShow;

int powStyle;

};

CMovingBitmap LineBlast::horizontal[6][30], LineBlast::vertical[6][30];

LineBlast::LineBlast(int style, int x, int y, int power) :powStyle(power), curShow(0)

{

this->style = style;

SetTopLeft(x, y);

sound = &CGameState::sound;

}

void LineBlast::LoadBitmap()

{

int \*bmpID[] = { blueBmp, purpleBmp, orangeBmp, greenBmp , redBmp, yellowBmp };

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 30; j++)

horizontal[i][j].LoadBitmap(bmpID[i][j + 55], RGB(254, 191, 200));

for (int j = 0; j < 30; j++)

vertical[i][j].LoadBitmap(bmpID[i][j + 25], RGB(254, 191, 200));

}

}

void LineBlast::OnMove()

{

curShow++;

}

void LineBlast::OnShow()

{

if (\*sound && curShow == 1) CAudio::Instance()->Play(AUDIO\_LINE\_BLAST, false);

switch (powStyle)

{

case 1:

horizontal[style - 1][curShow].SetTopLeft(x - (horizontal[style - 1][curShow].Width() / 2 - 25), y);

horizontal[style - 1][curShow].ShowBitmap();

break;

case 2:

vertical[style - 1][curShow].SetTopLeft(x, y - (vertical[style - 1][curShow].Height() / 2 - 25));

vertical[style - 1][curShow].ShowBitmap();

break;

default:

GAME\_ASSERT(0, "Invalid power style");

}

}

bool LineBlast::IsLast()

{

return curShow >= 29;

}

class MagicBlast :public Blast

{

public:

MagicBlast(int x, int y);

MagicBlast(CPoint);

bool IsLast();

static void LoadBitmap();

void OnMove();

void OnShow();

bool operator==(const MagicBlast& rhs);

private:

static CMovingBitmap bmp[4];

int curShow;

int delay;

int x, y;

};

CMovingBitmap MagicBlast::bmp[4];

MagicBlast::MagicBlast(int x, int y) :x(x), y(y), curShow(0)

{

sound = &CGameState::sound;

}

MagicBlast::MagicBlast(CPoint p) :x(p.x), y(p.y), curShow(0)

{ }

bool MagicBlast::IsLast()

{

return curShow == 3;

}

void MagicBlast::LoadBitmap()

{

bmp[0].LoadBitmap("Bitmaps\\MagicBlast1.bmp", RGB(255, 255, 255));

bmp[1].LoadBitmap("Bitmaps\\MagicBlast2.bmp", RGB(255, 255, 255));

bmp[2].LoadBitmap("Bitmaps\\MagicBlast3.bmp", RGB(255, 255, 255));

bmp[3].LoadBitmap("Bitmaps\\MagicBlast4.bmp", RGB(255, 255, 255));

}

void MagicBlast::OnMove()

{

delay = delay == 1 ? 0 : 1;

if (delay) curShow++;

}

void MagicBlast::OnShow()

{

if (delay)

{

bmp[curShow].SetTopLeft(x - bmp[curShow].Width() / 2, y - bmp[curShow].Height() / 2);

bmp[curShow].ShowBitmap();

}

}

bool MagicBlast::operator==(const MagicBlast & rhs)

{

return rhs.x == x && rhs.y == y;

}

class SuperBlast :public Blast

{

public:

SuperBlast(int x, int y, int delay = 0, bool showAll = false);

~SuperBlast();

void AddPoint(int x, int y);

bool IsLast();

static void LoadBitmap();

void OnMove();

void OnShow();

private:

void ShowLightning(bool showAll = false);

void DrawLine(CDC\*, const CPoint&, const CPoint&);

list<CPoint>\* GetRoutePoints(CPoint, CPoint);

static CAnimation chocalate;

vector<CPoint> target;

list<MagicBlast> magicBlasts;

int curShow;

int lightningDelay; //delay time of lightning exist

bool showAll; //Show all lightning in one-time

};

CAnimation SuperBlast::chocalate;

SuperBlast::SuperBlast(int x, int y, int delay, bool showAll) :curShow(0), lightningDelay(delay), showAll(showAll)

{

this->x = x;

this->y = y;

chocalate.SetDelayCount(3);

sound = &CGameState::sound;

}

SuperBlast::~SuperBlast()

{

CAudio::Instance()->Stop(AUDIO\_SUPER\_REMOVE);

}

void SuperBlast::OnMove()

{

curShow++;

chocalate.OnMove();

for (auto i = magicBlasts.begin(); i != magicBlasts.end();)

{

if ((\*i).IsLast()) i = magicBlasts.erase(i);

else

{

(\*i).OnMove();

i++;

}

}

}

void SuperBlast::OnShow()

{

if (!target.size()) return;

showAll ? ShowLightning(true) : ShowLightning();

for (auto i = magicBlasts.begin(); i != magicBlasts.end(); i++)

{ //Show magic blast effect on every target

(\*i).OnShow();

}

}

void SuperBlast::ShowLightning(bool showAll)

{

chocalate.SetTopLeft(x, y);

chocalate.OnShow(); //Show chocalate

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

CPen penLighting;

CPen \*pPen;

penLighting.CreatePen(PS\_SOLID | PS\_COSMETIC, showAll ? 5 : rand() % 10 + 1, RGB(207, 249, 245));

pPen = pDC->SelectObject(&penLighting);

if (showAll)

{ // Show all lightning effect at one time

for (auto i = target.begin(); i != target.end(); i++)

{

DrawLine(pDC, CPoint(x + 25, y + 25), \*i); //Draw a lightning from current position to target

}

}

else

{

for (int i = lightningDelay; i >= 0; i--)

{

if (curShow - i >= 0 && curShow - i < target.size())

{

DrawLine(pDC, CPoint(x + 25, y + 25), target[curShow - i]);//Draw a lightning from current position to target

}

}

}

// Restore the previous pen.

pDC->SelectObject(pPen);

CDDraw::ReleaseBackCDC();

Sleep(5);

}

void SuperBlast::DrawLine(CDC\* pDC, const CPoint& start, const CPoint& end)

{

list<CPoint>\* route = GetRoutePoints(start, end); //Get points on lighting route

//DrawLine

pDC->MoveTo(\*route->begin());

for (auto j = route->begin()++; j != route->end(); j++)

{

pDC->LineTo(\*j);

}

delete route;

MagicBlast blast(end);

bool blastExists = false;

for (auto i = magicBlasts.begin(); i != magicBlasts.end(); i++)

{ //to avoid show magic blast repeatly

if ((\*i) == blast)

{

blastExists = true;

break;

}

}

if (!blastExists) magicBlasts.push\_back(blast);//push a new magic blast if it doesn't exists

}

list<CPoint>\* SuperBlast::GetRoutePoints(CPoint start, CPoint end)

{

// Get points on line from start to end by straight line equation

list<CPoint>\* route = new list<CPoint>();

int interval = abs(start.x - end.x) / (abs(start.x - end.x) > 10 ? abs(start.x - end.x) / 10 : abs(start.x - end.x) > 0 ? abs(start.x - end.x) : 1);

int totalPoint = abs(start.x - end.x) > 10 ? abs(start.x - end.x) / 10 : abs(start.x - end.x);

int reverse = start.x > end.x ? -1 : 1;

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

{

CPoint point;

point.x = start.x + interval \* i \* reverse;

point.y = (point.x \* end.y - point.x \* start.y - start.x \* end.y + end.x \* start.y) / (end.x - start.x);

route->push\_back(point);

}

route->push\_back(end);

// move position of points in a stable range randomly

int j = 0;

for (auto i = route->begin(); i != route->end(); i++, j++)

{

if (j % 2) i->x += rand() % 20 - 10;

else i->y += rand() % 20 - 10;

}

return route;

}

void SuperBlast::AddPoint(int x, int y)

{

target.push\_back(CPoint(x, y));

}

bool SuperBlast::IsLast()

{

return ((curShow >= target.size() + lightningDelay) || (showAll && curShow >= 10));

}

void SuperBlast::LoadBitmap()

{

chocalate.AddBitmap("Bitmaps\\BlastSuperCandy1.bmp", RGB(254, 191, 200));

chocalate.AddBitmap("Bitmaps\\BlastSuperCandy2.bmp", RGB(254, 191, 200));

chocalate.AddBitmap("Bitmaps\\BlastSuperCandy3.bmp", RGB(254, 191, 200));

chocalate.AddBitmap("Bitmaps\\BlastSuperCandy4.bmp", RGB(254, 191, 200));

chocalate.AddBitmap("Bitmaps\\BlastSuperCandy5.bmp", RGB(254, 191, 200));

}

#ifndef SCOREBOARD\_H

#define SCOREBOARD\_H

namespace game\_framework

{

class ScoreBoard

{

friend class GameArea;

public:

ScoreBoard();

void OnShow();

void LoadBitmap();

bool IsReachedTarget();

private:

void ShowStars();

void ShowMoves();

void ShowScoreBar();

void ShowScore();

void ShowTarget();

CMovingBitmap scoreBar, blackBar, scoreBoard;

CMovingBitmap yellowStar, greenStar, redStar;

CMovingBitmap emptyStar2, emptyStar1;

CMovingBitmap target1, target2;

CInteger score, moves, target;

double oneStar, twoStar, threeStar, lastHighScore;

int mode;

};

}

#endif

#include "stdafx.h"

#include "Resource.h"

#include <mmsystem.h>

#include <ddraw.h>

#include "audio.h"

#include "gamelib.h"

#include "ScoreBoard.h"

namespace game\_framework

{

ScoreBoard::ScoreBoard()

: score(0), moves(0), oneStar(0), twoStar(0), threeStar(0), lastHighScore(0), mode(0)

{}

void ScoreBoard::OnShow()

{

ShowScoreBar();

scoreBoard.SetTopLeft(0, 0);

scoreBoard.ShowBitmap();

ShowMoves();

ShowScore();

ShowStars();

ShowTarget();

}

void ScoreBoard::LoadBitmap()

{

score.LoadBitmap();

scoreBoard.LoadBitmap("Bitmaps\\score\_board.bmp", RGB(0, 0, 0));

scoreBar.LoadBitmap("Bitmaps\\ScoreBar.bmp");

blackBar.LoadBitmap(IDB\_BLACK\_BAR);

yellowStar.LoadBitmap("Bitmaps\\YellowStar1.bmp", RGB(0, 43, 255));

greenStar.LoadBitmap("Bitmaps\\GreenStar1.bmp", RGB(0, 43, 255));

redStar.LoadBitmap("Bitmaps\\RedStar1.bmp", RGB(0, 43, 255));

emptyStar1.LoadBitmap("Bitmaps\\EmptyStar1.bmp", RGB(0, 43, 255));

emptyStar2.LoadBitmap("Bitmaps\\EmptyStar2.bmp", RGB(0, 43, 255));

target1.LoadBitmap("Bitmaps\\Target1.bmp", RGB(251, 200, 201));

target2.LoadBitmap("Bitmaps\\Target2.bmp", RGB(251, 200, 201));

}

bool ScoreBoard::IsReachedTarget()

{

switch (mode)

{

case 1: //Target reached when score higher than target

if (score >= target) return true;

case 2: //Target reached when all jelly cleared

if (!target.GetInteger()) return true;

default:

return false;

}

}

void ScoreBoard::ShowStars()

{

//Show empty star if score is lower than each relative score

CMovingBitmap \*thirdStar = score >= threeStar ? &yellowStar : &emptyStar2;

CMovingBitmap \*secondStar = score >= twoStar ? &greenStar : &emptyStar1;

CMovingBitmap \*firstStar = score >= oneStar ? &redStar : &emptyStar1;

int star\_X = scoreBoard.Left() + 159;

thirdStar->SetTopLeft(star\_X, scoreBoard.Top() + (281 - yellowStar.Height() / 2));

thirdStar->ShowBitmap();

secondStar->SetTopLeft(star\_X, (int)(scoreBoard.Top() + 281 + (((threeStar - twoStar) / threeStar) \* 254 - greenStar.Height() / 2)));

secondStar->ShowBitmap();

firstStar->SetTopLeft(star\_X, (int)(scoreBoard.Top() + 281 + (((threeStar - oneStar) / threeStar) \* 254 - redStar.Height() / 2)));

firstStar->ShowBitmap();

}

void ScoreBoard::ShowMoves()

{

int CurrentMoves = moves.GetInteger();

int size = 1;

while (CurrentMoves > 9)

{

CurrentMoves /= 10;

size++;

}

if (size <= 7)

{

moves.SetDigit(size);

moves.SetTopLeft(scoreBoard.Left() + 119 - 9 \* size, scoreBoard.Top() + 240);

moves.ShowBitmap();

}

}

void ScoreBoard::ShowScoreBar()

{

int X\_point = scoreBoard.Left() + 159, Y\_point = scoreBoard.Top() + 539;

double currentLevel = (score.GetInteger() / threeStar) \* 129;

currentLevel = currentLevel > 129 ? 129 : currentLevel;

for (int i = 0; i < 129; i++)

{

if (i < currentLevel)

{

scoreBar.SetTopLeft(X\_point, Y\_point);

scoreBar.ShowBitmap();

}

else

{

blackBar.SetTopLeft(X\_point, Y\_point);

blackBar.ShowBitmap();

}

Y\_point -= 2;

}

}

void ScoreBoard::ShowScore()

{

int CurrentScore = score.GetInteger();

int size = 1;

while (CurrentScore > 9)

{

CurrentScore /= 10;

size++;

}

if (size <= 7)

{

score.SetDigit(size);

score.SetTopLeft((scoreBoard.Left() + 144 - (18 \* size)), scoreBoard.Top() + 325);

score.ShowBitmap();

}

}

void ScoreBoard::ShowTarget()

{

switch (mode)

{

case 1:

target1.SetTopLeft(10, 90);

target1.ShowBitmap();

break;

case 2:

target2.SetTopLeft(10, 90);

target2.ShowBitmap();

break;

}

target.SetTopLeft(110, 95);

target.ShowBitmap();

}

}