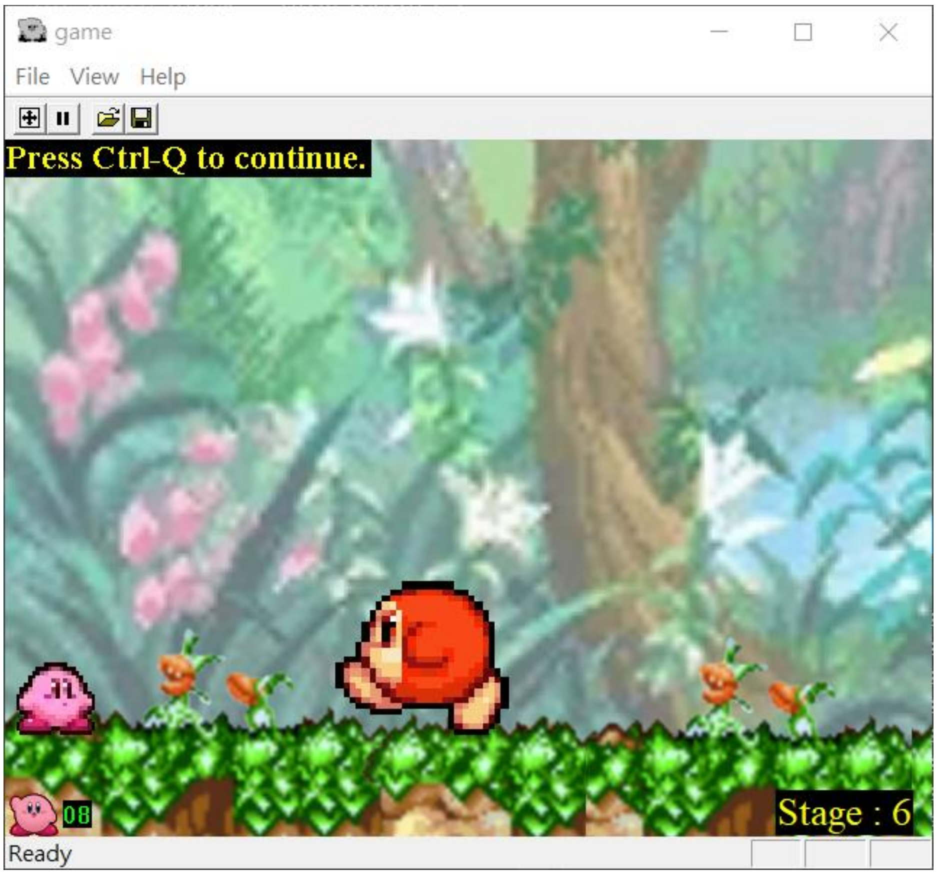
# 國立臺北科技大學
# 2021 Spring 資工系物件導向程式實習
# 期末報告

星之卡比



第 43 組

108820016 郭梓琳
108820044 蕭　　岳

# 目錄

## 1、簡介

### 1.動機

會選擇這個遊戲作為複製目標是因為蕭岳小時候很喜歡玩這個遊戲，所以在遊戲發想的時候第一個遊戲就想到它，而且遊戲方式還算直覺，最後就是遊戲的知名度還不錯。

### 2.分工

我們兩個分工的方式是大概一個禮拜會有一次的 meet，回顧上次以來的進度，提出問題並解決，之後列出接下來的進度應該是甚麼，並分為兩個部分，一人一半。每個人針對每個工作都有參與，沒有很特別的其他分工。

## 2、遊戲介紹

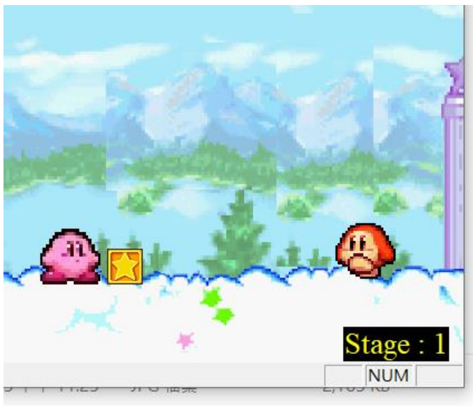
### 1. 遊戲說明

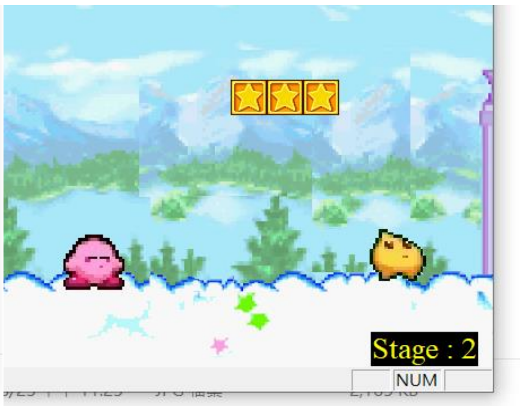
遊戲目標就是在消耗完血量之前通關，並且消滅 BOSS。

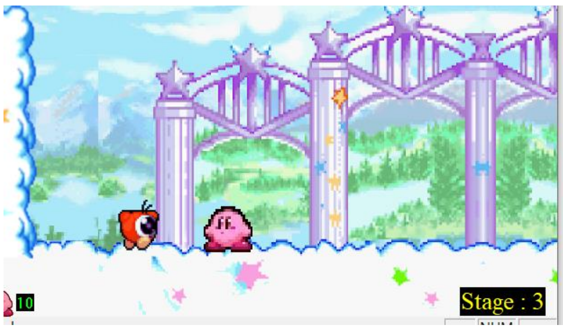
正常操作總共有六個按鍵，方向鍵上、下、左、右還有 C 鍵與空白鍵。遊戲中可以用左右鍵左右移動、按下鍵蹲下、按上鍵飛行或是按空白鍵跳躍來躲避敵人，或是用 C 鍵攻擊敵人，還可以按下鍵與 C 鍵讓卡比用踢來攻擊。在原生卡比的攻擊是吸，如果在攻擊範圍內的敵人會被吸收，而吸收的敵人如果有特殊能力可以按下鍵變身為特殊能力的卡比。特殊能力的卡比的 C 鍵攻擊每個都不同，WaddleDoo Kirby 是用武器揮擊，Burning Kirby 是噴火，Spark Kirby 是發出閃電。

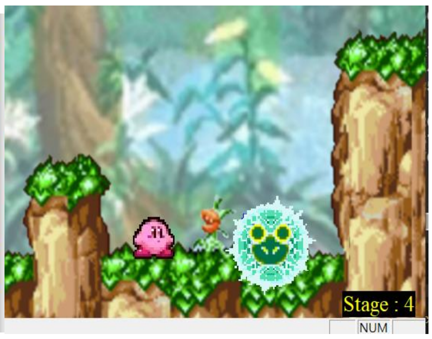
密技按鍵有 P 鍵跳到下一關，S 鍵無敵卡比不會受傷，但敵人還是會受傷。數字鍵的 1、2、3、4 直接變身。分別是 1: normal kirby，2: waddleDoo kirby，3:burning kirby，4: spark kirby。

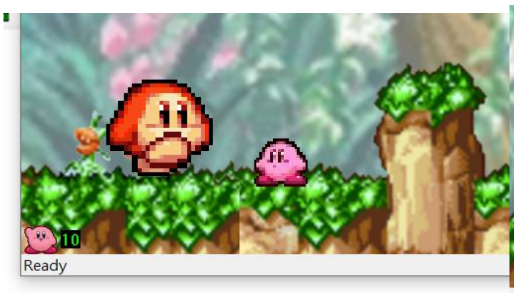
## 2. 遊戲圖形

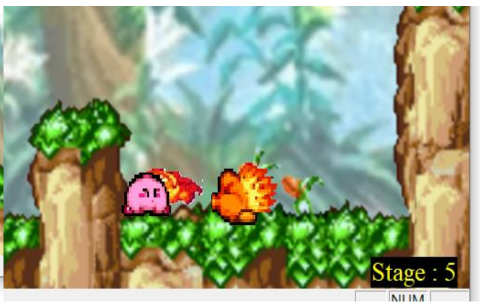

Stage : 1
NUM

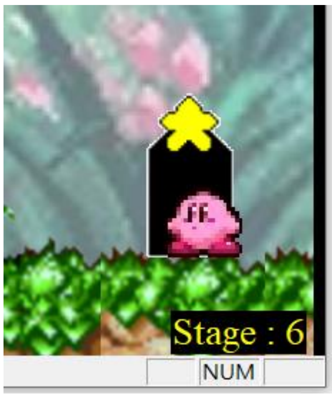

Stage : 2
NUM

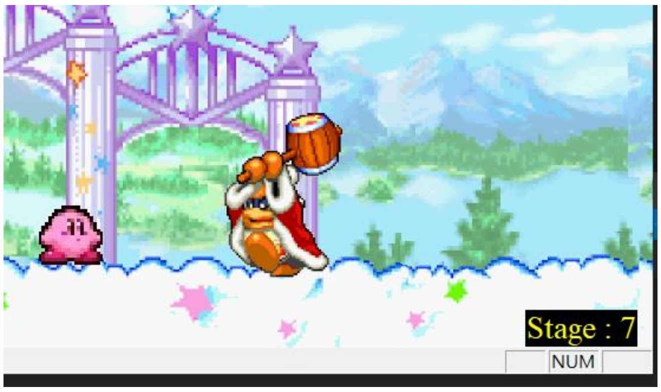

10
Stage : 3

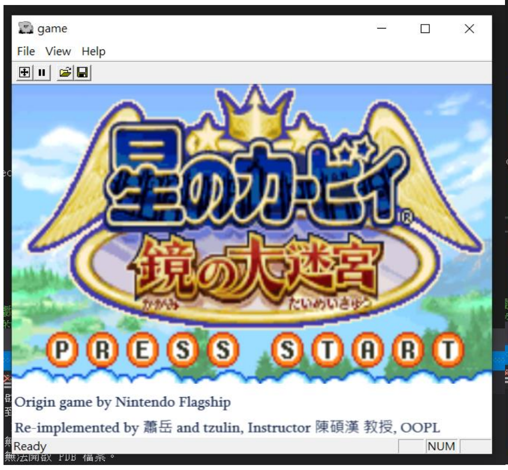

Stage : 4
NUM


10
Ready


Stage : 5

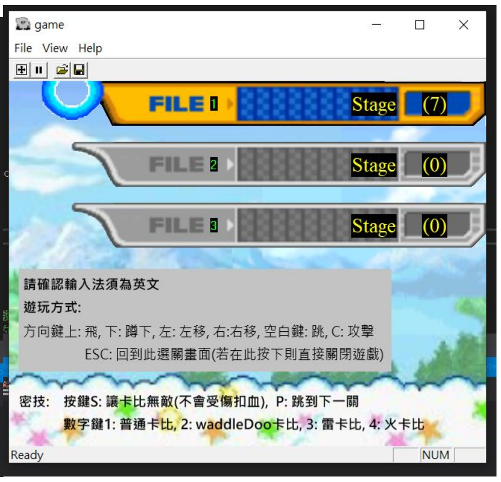

Stage : 6
NUM


Stage : 7
NUM



Origin game by Nintendo Flagship

Re-implemented by 蕭岳 and tzulin, Instructor 陳碩漢 教授, OOPL



請確認輸入法須為英文

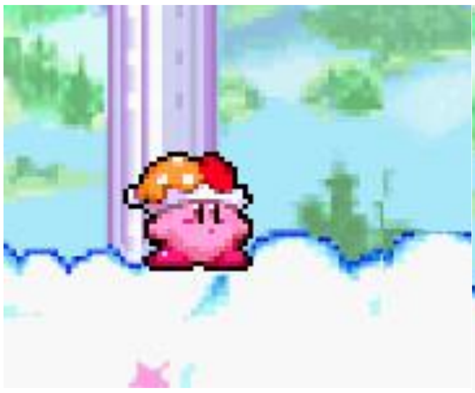
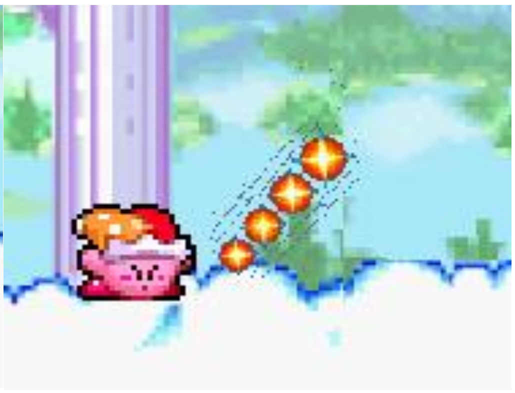
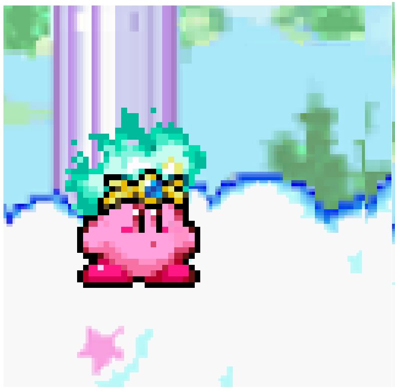
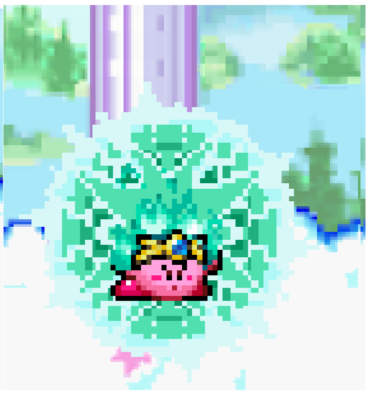
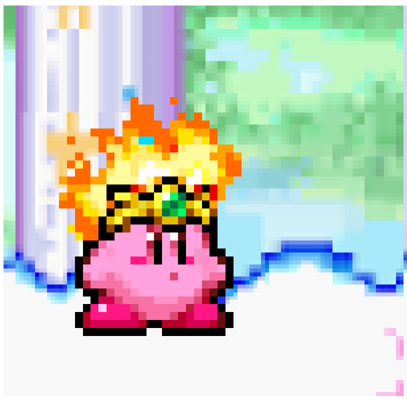
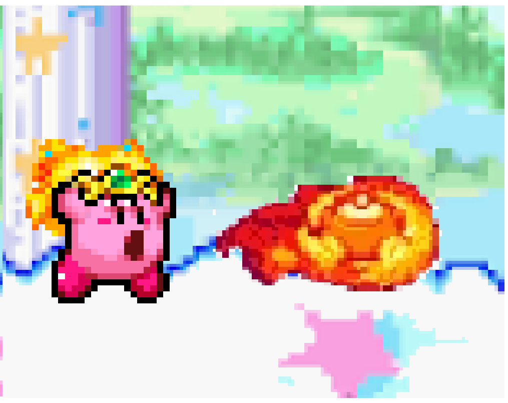
遊玩方式:

方向鍵上: 飛, 下: 蹲下, 左: 左移, 右:右移, 空白鍵: 跳, C: 攻擊

ESC: 回到此選關畫面(若在此按下則直接關閉遊戲)

密技: 按鍵S: 讓卡比無敵(不會受傷扣血), P: 跳到下一關

數字鍵1: 普通卡比, 2: waddleDoo卡比, 3: 雷卡比, 4: 火卡比

## 3. 遊戲音效

| 遊戲關卡 | 使用的音效(名稱) |
|---|---|
| 初始開場畫面 | title.mp3 |
| 選關畫面 | selection.mp3 |
| stage1-3 | starting_stage.mp3 |
| stage4-6 | forest_and_natural_area.mp3 |
| stage7 (Boss 關) | boss.mp3 |

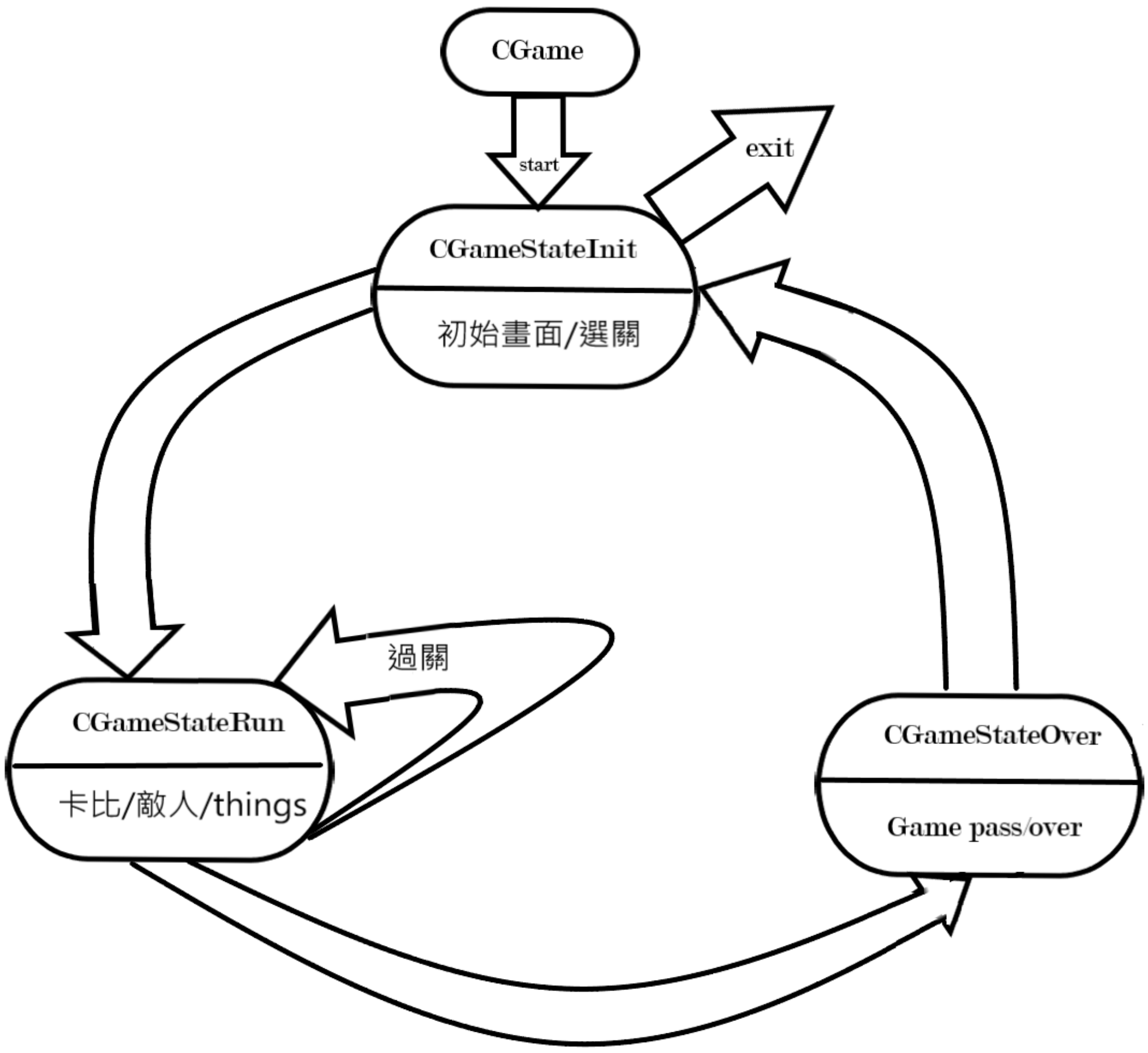
音檔來源: https://www.youtube.com/playlist?list=PLD349CFF6BB50FDDB

## 3、程式設計

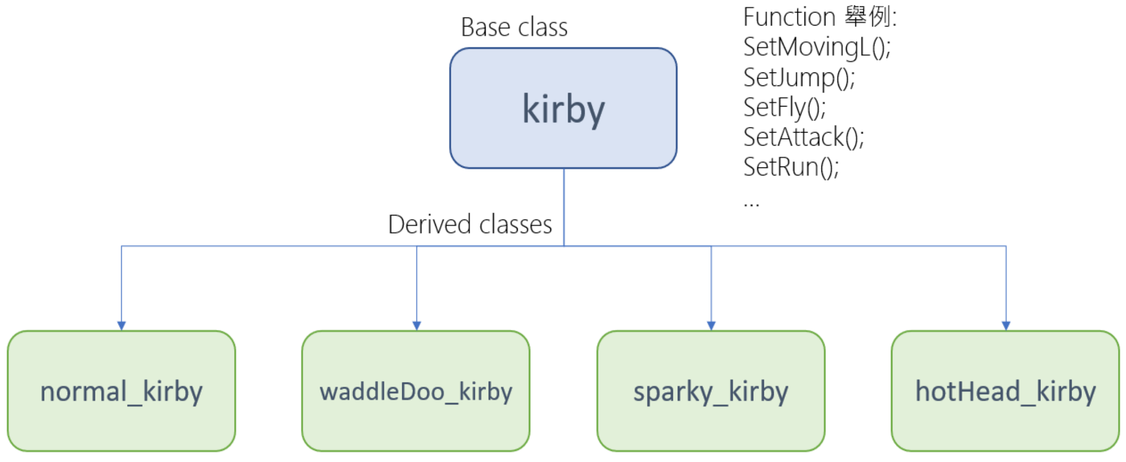
### 1. 程式架構

我們使用課堂提供的 framework 來實作。實作中新增的 class 有使用者可操作的主角人物卡比 (kirby)，會攻擊卡比的敵人(enemy)，卡比跟敵人都可以持有武器(weapon)，還有可以被卡比吃掉的星星磚塊和不能吃的牆壁磚塊(thing)。其中 kirby、enemy 和 thing 都是 base class，有使用到繼承。

遊戲流程 diagram:



繼承 class diagram:



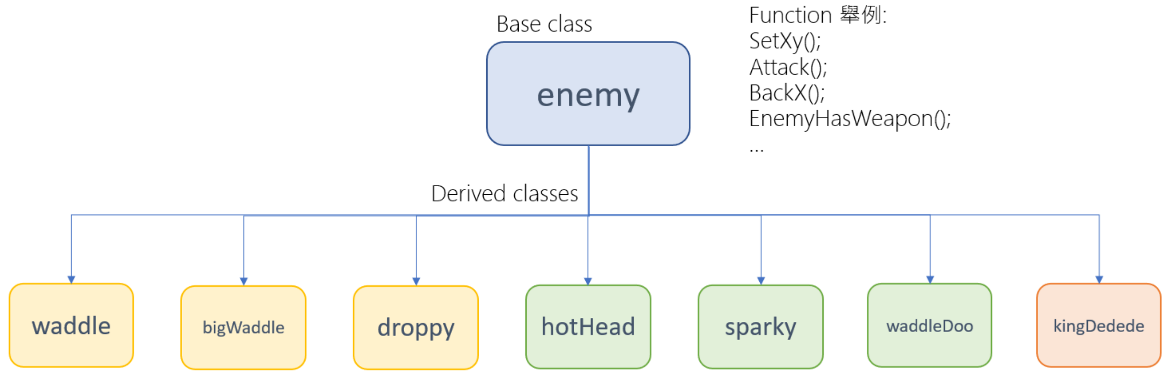
Function 舉例:
SetMovingL();
SetJump();
SetFly();
SetAttack();
SetRun();
...

初始卡比
攻擊模式: 吸

吃waddleDoo變身
攻擊模式: 揮武器

吃sparky變身
攻擊模式: 發出閃電

吃hotHead變身
攻擊模式: 噴火球

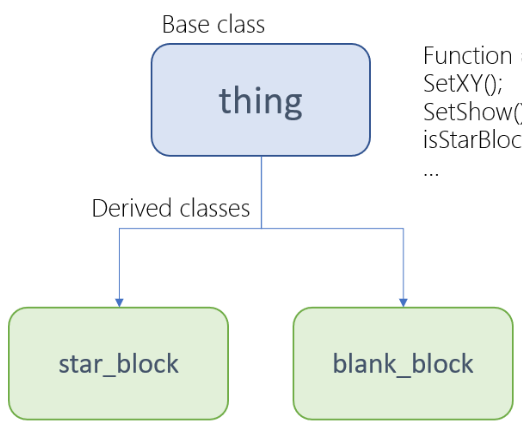


Function 舉例:
SetXy();
Attack();
BackX();
EnemyHasWeapon();
...

最普通的 enemy，沒有武器不會攻擊
攻擊力為 1，血量為 2

會攻擊的 enemy，持有武器可以攻擊
被卡比吃掉後，卡比可變身，使用它們的武器
攻擊力為 1，血量為 1

遊戲最終Boss
不可被卡比吃掉
攻擊力為 2
血量為 10



Function 舉例:
SetXY();
SetShow();
isStarBlock();
...

可以被卡比吃掉
然後吐出星星作為
攻擊手段

不能被吃掉
作為填補牆壁的磚塊

## 2. 程式類別

| 類別名稱 | .h 行數 | .cpp 行數 | 說明 |
|---|---|---|---|
| CGameStateInit | 20 | 150 | 初始畫面/選關 |
| CGameStateRun | 30 | 1097 | 判斷/跳關/遊戲進行 |
| CGameStateOver | 20 | 90 | 過關/死亡 |
| thing | 23 | 56 | 物體的判斷 |
| starBlock | 4 | 7 | 星磚 |
| blankBlock | 4 | 7 | 填充關卡的突起 |
| weapon | 34 | 126 | 武器的顯示/判斷依據 |
| enemy | 63 | 292 | 敵人的動作/受傷 |
| waddle | 5 | 41 | 敵人 1 圖片 handle |
| waddleDoo | 5 | 48 | 敵人 2 圖片 handle |
| sparky | 5 | 45 | 敵人 3 圖片 handle |
| hotHead | 5 | 48 | 敵人 4 圖片 handle |
| droppy | 5 | 45 | 敵人 5 圖片 handle |
| bigWaddle | 5 | 36 | 敵人 6 圖片 handle |
| kingDedede | 10 | 196 | BOSS |
| kirby | 115 | 1166 | 卡比的動作 |
| normal_kirby | 1 | 1 | 一般卡比 |
| sparky_kirby | 9 | 307 | 雷電卡比的圖片 handle |
| hotHead_kirby | 13 | 608 | 火卡比圖片 handle |
| waddleDoo_kirby | 14 | 615 | waddle 卡比圖片 handle |
| 總行數 | 390 | 4981 | |

### 3. 程式技術

在 CGameStateRun，主要為關卡執行的部分，我們用 std::vector 來存每一關的 enemies 及 things。std::vector 裡面放的是物件指標，在 call 它們各自的 function 時才不會 shallow copy，而造成功能不正常執行。判斷物件之間有沒有相遇（碰撞），我們是用圖片座標有沒有重疊來判定，但似乎還是有些 bug 存在。

## 4、結語

### 1. 問題及解決方法

**問題：** 找不到乾淨的背景圖素材

**解決：** 自己截遊戲畫面的圖片搭配找網路上的圖片資源，再修圖把不要的部份去掉、用對稱之類的方式把背景地圖做好

**問題：** 卡比的功能在關卡中沒辦法正常運作

**解決：** 我們用到 shallow copy，應該用 reference 或是 pointer 傳，不然會改不到原本真正的物件（p.s.感謝助教的幫忙!!）

**問題：** 卡比和敵人會走出遊戲地圖外

**解決：** 加入對地圖邊界的判斷，但是 boss 有時受傷會跳出左右邊界，kirby 還會飛出上緣，但是我們明明已經加了邊界了 QQ

**問題：** 卡比和敵人會穿過彼此和障礙物，或卡在障礙物附近

**解決：** 寫物件相遇的判斷，並讓他們再碰到後做相應的動作(轉向繼續走或受傷損血)，大致可以運作，但有時會有 bug

**問題：** 延伸遊戲地圖的範圍，卡比從初始位置走到地圖中間時要保持在中間的位置，然後移動地圖，各個物件也要有相應的移動 ，包含敵人、磚塊的上下左右移動位置（p.s.感謝助教的建議!!）

**解決：** 用各個物件的 xy 座標來做判斷及移動，大致可以實現運作，但還

是有一些 bug，敵人可能跟著飛到空中或下陷到地底

**問題：** 在加了遊戲最後的 boss 關後一直出現 memory leak，但一直找不到問題在哪裡

**解決：** 把 base class 的 destructor 加上 virtual 才會也去子類別 destruct。不太確定是否正確，但修了之後確實沒有再出現 detect memory leak 的訊息。因為是在進了 boss 關後才出現 detect memory leak 的訊息，我們一直以為是最後一關的問題，但其實是我們之前沒有寫好，這個問題我們花了蠻多時間才解決 QQ

**問題：** 遊戲打包後，存檔功能沒辦法正常運作

**解決：** 不太清楚原因且沒有找到解決辦法

**問題：** 如何把想法用 code 實現

**解決：**先和同伴討論，列出一些可能可行的想法，然後一一嘗試，最後確定使用的方法。

**問題：** 如何合作開發

**解決：**頻繁討論，遇到問題一起解決，工作或是任務明確，匯報進度。

## 2. 時間表

| 週次 | 組員-郭梓琳(hr) | 組員-蕭岳(hr) | 說明 |
|---|---|---|---|
| 1 | 0 | 0 | 第一次上課，尚未有進度<br>討論遊戲主題 |
| 2 | 4 | 2 | 完成 git 練習<br>開始做 game framework 練習 |
| 3 | 1 | 1 | 完成 game framework 練習 |
| 4 | 3 | 7 | 蒐集卡比素材，開始寫 code<br>讓卡比可以左右走動 |
| 5 | 8 | 6 | 卡比可以做左右移動(含動畫)、跳、蹲、攻擊(含動畫)的動作 |

|  |  |  |  |
|---|---|---|---|
| 6 | 3 | 5 | 做卡比"飛"的動作，尚未成功 |
| 7 | 9 | 6 | 繼續做卡比"飛"，補上卡比"跳"的動畫<br>新增第一個敵人角色 waddle (可以來回走動) |
| 8 | 7 | 0 | 卡比"飛"動作完成(含動畫)<br>卡比目前已完成的動作整合 |
| 9 | 3 | 0 | 把 code 整理得乾淨清楚一點<br>檢查 bug |
| 10 | 2 | 2 | 找地圖背景素材<br>卡比被敵人攻擊會受傷(動畫) |
| 11 | 12 | 7 | 新增敵人 waddleDoo<br>waddleDoo 攻擊動畫<br>waddleDoo 可以攻擊卡比<br>物件之間相遇的判斷(為了寫: 卡比被敵人攻擊會受傷，敵人被卡比撞到也會損血) |
| 12 | 17 | 15 | 卡比攻擊敵人<br>卡比被敵人武器攻擊會受傷(損血)<br>卡比碰到障礙物時不能穿過繼續走<br>敵人碰到地圖邊界和障礙物時不能穿過去繼續走<br>地圖的移動(遊戲畫面延伸) |
| 13 | 2 | 4.5 | 增加遊戲開場畫面 |
| 14 | 3 | 3 | 選關畫面及功能(存檔) |
| 15 | 2 | 6.5 | 存檔功能<br>換到下一關功能<br>檢查、修 bug |
| 16 | 6 | 10 | 製作新關卡 stage2<br>新增敵人 sparky 及 hotHead 且攻擊等功能正常<br>修 bug、memory leak |
| 17 | 11 | 2 | 找背景音樂素材、加背景音樂<br>卡比變身功能及切動畫圖<br>新增敵人 droppy 和 bigWaddle |
| 18 | 19 | 20 | 新增敵人 kingDedede<br>關卡 3~6、關卡 7(boss 關)完成<br>換關功能完善(game over 回到選關畫面、打贏 boss 回到選關畫面且存檔紀錄清零) |

|  |  |  |  |
|---|---|---|---|
|  |  |  | 修 bug、memory leak<br>更改遊戲 icon<br>打包遊戲 |
| 總時數 | 112 | 99 |  |

## 3. 貢獻比例

郭梓琳: 50%,　蕭岳: 50%

## 4. 自我檢核表

| 項目 | 項目 | 完成否 | 無法完成原因 |
|---|---|---|---|
| 1 | 解決 Memory leak | 已完成 |  |
| 2 | 自訂遊戲 Icon | 已完成 |  |
| 3 | 全螢幕啟動 | 未完成 | 忘記要做這個項目，最後時間來不及完成 |
| 4 | 有 About 畫面 | 已完成（在最一開始的畫面） |  |
| 5 | 初始畫面說明按鍵及滑鼠用法與密技 | 已完成（在選關畫面） |  |
| 6 | 上傳 setup/apk/source 檔 | 已完成 |  |
| 7 | setup 檔可正確執行 | 已完成 |  |
| 8 | 報告字型、點數、對齊、行距、頁碼等格式正確 | 已完成 |  |

## 5. 收獲

郭梓琳:

在這學期的這門課中學到了遊戲的流程架構應該是如何。大致是三個 state，Init、Run 和 Over state，實際實作之後有對整個概念和它是怎麼去

到各個 state 更了解。也對 c++ 語言更熟悉，尤其是物件之間的互動、繼承關係和要記得注意 memory leak 等等。透過看輸出及錯誤清單就可以解決大部分的 bug，但有時候就需要用到這次學到的設中斷點的方法，然後逐步執行，就更能找到問題點。還有學到了和別人一起共同開發的模式，以往上過的程式課都是個人作業，和別人一起開發就需要各種溝通，也一起學習、成長。有別人一起做真的可以看到一些自己發現不到的盲點。在這次過程中也因為常常使用到 git，對 git 的操作也更加熟悉，git 真的是很好用的工具啊! 以前都沒有這個習慣，就算是個人自己開發，用 git 來做版控一定會更順暢，比較知道自己每次做了什麼，也不用怕寫爛了，反正可以找回以前的版本回來改。

蕭岳:

這次的收穫條列式的來說 1.練習檢查別人的 code，不管是同伴的程式也好，frame work 的 function 也罷，都帶有不同的風格，實現目標的方式也不盡相同，經過這次的過程，我覺得自己的程式理解甚至維護都有不少進步。2.熟悉 C++ 3.練習目的導向的 coding，而不是成績導向的 coding，以前大一大多都是想著如何把分數拿到，現在一整個學期都是想著如何把遊戲完整。4.練習 debug，只要有程式，就會有 bug，這次練習非常多次，bug 的內容也讓自己印象深刻。

## 6. 心得、感想

郭梓琳:

這次寫的這個遊戲是對我來說目前寫過的最大的一個專案了。從來沒有想過自己能和同學一起寫出這樣的遊戲。雖然還是蠻粗糙的，完全比不上原本的<星之卡比>，而且也還有一些 bug，但我已經覺得蠻有成就感的了! 從一開始全黑什麼都沒有的畫面到第一次讓卡比可以出現在上面左右走動的時候，我真的超開心超感動的，跨出第一步的感覺。看到它真的能用左右鍵自己操控，感覺很新奇。但在整學期的實作過程中，其實真的還是遇到蠻多問題的，有時

候我們兩個一起討論、嘗試好幾個小時還是卡在一樣的問題真的是很挫折。有時候還會發現根本就只是少了一個判斷或什麼的，結果就造成一連串的錯，debug 很久，也會覺得自己很好笑也對自己蠻生氣的，怎麼會因為這種問題搞這麼久。這學期整個過程中，我真的非常非常感謝我的組員蕭岳，我一直覺得要這樣寫出一個遊戲來真的很困難，有時候也很沒有方向，那我們就會一起討論、構思，他總是能給我很多很棒的建議也幫助我很多，我卡住的時候，有時他很快就能找到問題的癥結點，然後我才發現其實不是那麼難的問題，都有方法解決。真的非常感謝他，讓我們一起合作做出了一個我覺得算是很不錯的成果 ！

蕭岳:

　　自己有非常嚴重得拖延症，這次的過程我還是一樣都拖到最後一天的最後一秒，可是有同伴在，總覺得應該要把程式寫到一定的程度，這心理壓力和時間壓力都有成功轉換成動力。總覺得自己適合團體活動，不過更感謝同伴對我的包容，我還真的是個幸運的人，從小都能遇到最好的夥伴。學期初有預想到這門課會花不少的時間，但是今年遇到疫情在家上課，期末不少的考試與報告擠在一起讓我的期末周只有一個字可以形容--趕!但面對即將要完成遊戲卻又有捨不得的感覺，但是看看時間又真的只能捨得!時間、精力都有限，不可能把所有想要實現的功能、角色、內容都完成，和同伴一起討論決定之後把不該分心的部分區隔開來，完成真正需要的部分更重要。

## 7. 對於本課程的建議

先感謝老師與助教們的辛勞、心累，整個學期幫助我們很多及給予我們建議，希望學弟學妹更加優秀，不會這麼多麻煩~~

具體的建議是:

1. 可能可以列出一些遊戲選擇的建議或類型，幫助大家選擇遊戲更有方向

2.如果有些 oopl gitLab 的更新可以更多的宣導，讓大家看到

3.希望步驟教學可以更新不要版本不同

## 附錄

### Header.h（我們所有 class 的宣告都放在這裡）

```
#ifndef HEADER_H
#define HEADER_H
namespace game_framework {
    class thing {
        public:
          virtual void LoadBitmap();
          virtual bool isStarBlock();
          void OnShow();
          void SetShow(bool);
          void SetXY(int, int);
          thing();
          virtual ~thing();
          int* GetHw();
          int* GetXy();
          bool GetShow();
          void YouAreLeft(bool);
          int Top();
          int Left();
          std::string GetKind();
          void SetKind(std::string input);
        protected:
          CMovingBitmap blockPic;
          int x, y;
          bool IsShow;
          std::string name = "";
    };
    class starBlock : public thing {
         void LoadBitmap() override;
          bool isStarBlock() override;
    };
    class blankBlock : public thing {
            void LoadBitmap() override;
        bool isStarBlock() override;
};
class weapon {
    public:
        weapon();
        ~weapon();
         void LoadBitmap(char** pic, int* rgb, int n);
         void LoadBitmap(int IDB_INPUT, int* rgb, int n);
         void LoadBitmap(char* pic, int* rgb, int n);
         void OnShow();
         void OnMove();
         void AnimationReset();
         void SetShow(bool);
         void SetWeapon(int enemyX, int enemyY, bool enemyFaceR);
         void SetXy(int, int);
         void SetAttackState(int attack_time, bool IsFacingR, int* input_Xy);
         int GetAttackTime();
         int* GetXy();
         void YouAreLeft(bool YouAreLeft);
         bool GetAttackFacingR();
         bool WeaponIsShow();
         bool IsFinalBitmap();
         void SetOwner(std::string which_enemy);
         int Width();
         int Height();
     protected:
      CAnimation PlayAttack;
      int x, y;
      int ImgW = 75, ImgH = 128;
      int enemyImgH = 60, enemyImgW = 60;
      int attack_time;
      bool IsShow;
      bool AttackIsFacingR;
      bool OtherFromL;
      std::string owner;
};
class kirby;
class enemy {
    public:
        enemy();
    virtual ~enemy();
    virtual void LoadBitmap();
    virtual void Reset();
    virtual void OnShow();
    virtual void Hurt(int input, int time);
    virtual void OnMove();
    int* GetXy();
    int GetHp();
    int GetWidth();
    int GetHeight();
    int GetPower();
    int Top();
    int Left();
    std::string GetKind();
    void SetXy(int input_x, int input_y);
    void SetThings(vector<thing*> input_ThingList);
    void BackX(bool fromL);
    void Attack(kirby k, int time);
    void YouAreLeft(bool YouAreLeft);
    bool EnemyFacingR();
    bool EnemyHasWeapon();
    bool GetCanAttack();
    void SetCounter(int input) {
          game_state_counter = input;
    }
    void SetMap(CMovingBitmap * Map);
    weapon GetWeapon();
protected:
    CAnimation MovingL;
    CAnimation MovingR;
    CAnimation AttackR;
    CAnimation AttackL;
    CAnimation Stand;
    CAnimation StandL;
    weapon wL;
    weapon wR;
    CMovingBitmap* Map;
    vector<thing*> StarBlockList;
    int x, y, hp;
    int power;               // 攻擊力
```

```
        int origin_x, origin_y;
        int ImgW = 64, ImgH = 60;
        int floor;              // 地板 y 座標
        int LastHurt;
        int LastAttack;
        int game_state_counter, x_old = 0, y_old = 0, m_x_old = 0, m_y_old = 0;
        bool IsFacingR;
        bool IsMovingL;
        bool IsMovingR;
        bool IsAttack;
        bool IsHurt;
        bool OtherFromL;
        bool HasWeapon;
        bool CanAttack;
        std::string kind;
};
class waddle : public enemy {
    public:
         void LoadBitmap() override;
         void Reset() override;
};
class waddleDoo : public enemy {
     public:
         void LoadBitmap() override;
         void Reset() override;
};
class sparky : public enemy {
     public:
         void LoadBitmap() override;
         void Reset() override;
};
class hotHead : public enemy {
    public:
         void LoadBitmap() override;
         void Reset() override;
};
class droppy : public enemy {
    public:
         void LoadBitmap() override;
         void Reset() override;
```

```
};
class bigWaddle : public enemy {
    public:
         void LoadBitmap() override;
         void Reset() override;
};
class kingDedede : public enemy {
    public:
         void LoadBitmap() override;
         void Reset() override;
         void OnShow() override;
         void Hurt(int input, int time) override;
         void OnMove() override;
     protected:
         CAnimation HurtR, HurtL;
};
class kirby {
    public:
      virtual void SetAttack(bool input);
      virtual void SetKindInit();
      virtual void SetEaten(bool input);
      virtual void SetEaten(bool input, std::string name);
      virtual void LoadBitmap();
      virtual void ThrowStar();
      virtual void OnMove();
      virtual void OnShow();
      virtual weapon* GetWeapon();
      kirby();
      virtual ~kirby();
      void kirby::StageReSet(int hp_left, std::string kind);
      void SetXY(int x_in, int y_in);
      void SetMovingL(bool input);
      void SetMovingR(bool input);
      void SetFacingR(bool input);
      void SetFacingL(bool input);
      void SetDown(bool input);
      void SetJump(bool input);
      void SetFly(bool input);
      void SetCounter(int);
      void SetHack(bool input);
```

```
        void SetKind(std::string input);
        void SetEnemies(vector<enemy*> input_EnemyList);
        void SetThings(vector<thing*> input_ThingList);
        void SetMap(CMovingBitmap* Map);
        void SetDoor(CMovingBitmap* door);
        void SetRun(bool input);
        bool Hurt(int input, int time);
        void YouAreLeft(bool YouAreLeft);
        void KirbyCopy(kirby* new_kirby);
        int GetCase();
        int GetHp();
        int* GetXy();
        int GetWidth();
        int GetHeight();
        std::string GetKind();
        std::string GetEatenEnemy();
        bool GetEaten();
        bool GetIsGround();
        bool IsAlive();
        bool IsScreamR();
        bool IsScreamL();
        bool GetFacingR();
protected:
        CAnimation KirbyMovingL;
        CAnimation KirbyMovingR;
        CAnimation KirbyFatMovingL;
        CAnimation KirbyFatMovingR;
        CAnimation KirbyStand;
        CAnimation KirbyStandL;
        CAnimation KirbyFatStand;
        CAnimation KirbyFatStandL;
        CAnimation KirbyJumpR;
        CAnimation KirbyJumpL;
        CAnimation KirbyScreamR;
        CAnimation KirbyScreamL;
        CAnimation KirbyDownAttackR;
        CAnimation KirbyDownAttackL;
        CAnimation KirbyFlyR;
        CAnimation KirbyFlyingR;
        CAnimation KirbyFlyL;
        CAnimation KirbyFlyingL;
        CAnimation KirbyHurtR;
        CAnimation KirbyHurtL;
        CMovingBitmap KirbyDownR;
        CMovingBitmap KirbyDownL;
        CMovingBitmap * Map;
        CMovingBitmap * Door;
        vector<enemy*> EnemyList;
        vector<thing*> StarBlockList;
        weapon StarThrow;
        std::string EatenEnemy = "";
        std::string kirby_kind;
        int x, y, hp;
        const int ImgW = 60, ImgH = 60;
        int now_img_w, now_img_h;
        int sky_top;      // 天頂 y 座標
        int floor;        // 地板 y 座標
        int init_velocity;         // 初始速度
        int velocity;     // 目前速度
        int init_fly_velocity;    // 初始飛行速度
        int fly_velocity;            // kirby inAir 時的飛行速度
        int game_state_counter;             // game state counter
        int LastHurt;              // last hurt time 初始為零
        int BoundaryTop, BoundaryRight, BoundaryLeft; // boundary
        bool IsRising;       // true 表示上升
        bool IsMovingL;
        bool IsMovingR;
        bool IsFacingR;
        bool IsDown;
        bool IsAttack;
        bool InAir;
        bool IsJumping;
        bool IsFlying;
        bool IsFat;
        bool FlyUp;
        bool OtherFromL;
        bool IsHurt;
        bool IsEaten;
        bool YouAreGround;
        bool IsRun;
```

```
	bool IsHack;
};
class normal_kirby : public kirby {};
class sparky_kirby : public kirby {
public:
	void SetEaten(bool input) override;
	void SetEaten(bool input, std::string name) override;
	void LoadBitmap() override;
	void OnMove() override;
	void ThrowStar() override;
	void SetKindInit() override;
};
class hotHead_kirby : public kirby {
public:
	void SetEaten(bool input) override;
	void SetEaten(bool input, std::string name) override;
	void LoadBitmap() override;
	void OnMove() override;
	void OnShow() override;
	void ThrowStar() override;
	void SetKindInit() override;
	weapon* GetWeapon() override;
protected:
	weapon left_side;
};
class waddleDoo_kirby : public kirby {
public:
	void SetEaten(bool input) override;
	void SetEaten(bool input, std::string name) override;
	void LoadBitmap() override;
	void OnMove() override;
	void OnShow() override;
	void ThrowStar() override;
	void SetKindInit() override;
	void SetAttack(bool input) override;
protected:
	weapon left_side;
};
inline bool EnemyCanAttack(enemy & e, kirby & k) {
	int* kirbyXy = k.GetXy();
```

```
            int* enemyXY = e.GetXy();
            // enemy 在 kirby 右邊
            if (enemyXY[0] - e.GetWidth() - 10 < kirbyXy[2] && enemyXY[0]  > kirbyXy[2]
&& !e.EnemyFacingR()) {
                    if (kirbyXy[1] - enemyXY[1] < 150) {
                            delete[] kirbyXy;
                            delete[] enemyXY;
                            e.YouAreLeft(false);
                            return true;
                    }
            }
            // enemy 在 kirby 左邊
            else if (enemyXY[2] + e.GetWidth() + 10 > kirbyXy[0] && enemyXY[2] < kirbyXy[0] &&
e.EnemyFacingR()) {                  // enemy meet kirby from right
                    if (kirbyXy[1] - enemyXY[1] < 150) {
                            delete[] kirbyXy;
                            delete[] enemyXY;
                            e.YouAreLeft(true);
                            return true;
                    }
            }
            delete[] kirbyXy;
            delete[] enemyXY;
            return false;
    }
    template < class T >
    bool KirbyCanAttack(kirby & Kirby, T * t) {
            int* StarBlockXy = t->GetXy();
            int* KirbyXy = Kirby.GetXy();
            int for_count = 0;
            for (; for_count < 10; for_count++) {
                    int x_different = (StarBlockXy[0] + StarBlockXy[2]) / 2 - (KirbyXy[0] + KirbyXy[2])
/ 2;
                    int x_different2 = (KirbyXy[0] + KirbyXy[2]) / 2 - (StarBlockXy[0] + StarBlockXy[2])
/ 2;
                    int y_different = StarBlockXy[1] - KirbyXy[1];
                    int y_different2 = KirbyXy[3] - StarBlockXy[3];
                    if (x_different >= for_count && x_different <= 150 && y_different <= for_count * 8
&& y_different2 <= for_count * 8 && Kirby.IsScreamR()) {
                            delete[] StarBlockXy;
```

```
                        delete[] KirbyXy;
                        return true;
                }
                if (x_different2 >= for_count && x_different2 <= 150 && y_different <= for_count * 8
&& y_different2 <= for_count * 8 && Kirby.IsScreamL()) {
                        delete[] StarBlockXy;
                        delete[] KirbyXy;
                        return true;
                }
        }
        delete[] StarBlockXy;
        delete[] KirbyXy;
        return false;
    }
    template <class T, class U>
    bool Meet(T & a, U & b) {
        int* aXy = a.GetXy();
        int* bXy = b.GetXy();
        int i = 0, n = 1;
        for (int count = 0; count < 2; count++) {
                for (int _count = 0; _count < 2; _count++) {
                        if (aXy[i] > bXy[0] && aXy[i] < bXy[2] && aXy[n] > bXy[1] && aXy[n] <
bXy[3]) {
                                delete[] aXy;
                                delete[] bXy;
                                if (i == 0) {
                                        // b 在 A 左邊
                                        a.YouAreLeft(false);
                                        b.YouAreLeft(true);
                                }
                                else {
                                        // a 在 b 左邊
                                        a.YouAreLeft(true);
                                        b.YouAreLeft(false);
                                }
                                return true;
                        }
                        n += 2;
                }
                n = 1;
                i += 2;
            }
            i = 0, n = 1;
            for (int count = 0; count < 2; count++) {
                for (int _count = 0; _count < 2; _count++) {
                    if (bXy[i] > aXy[0] && bXy[i] < aXy[2] && bXy[n] > aXy[1] && bXy[n] <
aXy[3]) {
                        delete[] aXy;
                        delete[] bXy;
                        if (i == 0) {
                            // a在b左邊
                            a.YouAreLeft(true);
                            b.YouAreLeft(false);
                        }
                        else {
                            // b在a左邊
                            a.YouAreLeft(false);
                            b.YouAreLeft(true);
                        }
                        return true;
                    }
                    n += 2;
                }
                n = 1;
                i += 2;
            }
            // 沒碰到
            delete[] aXy;
            delete[] bXy;
            return false;
    }
}
#endif
```

## things.cpp (base class thing & derived class starBlock and blankBlock)

```
#include "stdafx.h"
#include "source/Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
```

```
#include "source/audio.h"
#include "source/gamelib.h"
#include "Header.h"
namespace game_framework {
    thing::thing() {
          IsShow = true;
    }
    thing::~thing() {}
    void thing::LoadBitmap() {}
    void thing::SetXY(int input_x, int input_y) {
          x = input_x;
          y = input_y;
    }
    void thing::YouAreLeft(bool input) {}
    int* thing::GetHw() {
          return new int[2]{ blockPic.Height(), blockPic.Width() };
    }
    int* thing::GetXy() {
          return new int[4]{ x, y , x + blockPic.Width(), y + blockPic.Height() };
    }
    bool thing::GetShow() {
          return IsShow;
    }
    void thing::OnShow() {
          blockPic.SetTopLeft(x, y);
          blockPic.ShowBitmap();
    }
    void thing::SetShow(bool input) {
          IsShow = input;
    }
    int thing::Top() {
          return y;
    }
    int thing::Left() {
          return x;
    }
    bool thing::isStarBlock() {
          return false;
    }
    std::string thing::GetKind() {
        return name;
    }
    void thing::SetKind(std::string input) {
        name = input;
    }
    void starBlock::LoadBitmap() {
        blockPic.LoadBitmap(IDB_STARBLOCK);
    }
    bool starBlock::isStarBlock() {
        return true;
    }
    void blankBlock::LoadBitmap() {
        blockPic.LoadBitmap(".\\res\\blankBlock.bmp", RGB(0, 0, 0));
    }
    bool blankBlock::isStarBlock() {
        return false;
    }
}
```

## weapon.cpp (class weapon)

```
#include "stdafx.h"
#include "source/Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "source/audio.h"
#include "source/gamelib.h"
#include "Header.h"
namespace game_framework {
    weapon::weapon() {
        IsShow = false;
        attack_time = 0;
    }
    weapon::~weapon() {}
    void weapon::LoadBitmap(char** pic, int* rgb, int n) {
        for (int i = 0; i < n; i++) {
            PlayAttack.AddBitmap(pic[i], RGB(rgb[0], rgb[1], rgb[2]));
        }
    }
    void weapon::LoadBitmap(char* pic, int* rgb, int n) {
        for (int i = 0; i < n;i++) {
            PlayAttack.AddBitmap(pic, RGB(rgb[0], rgb[1], rgb[2]));
```

```
	}
}
void weapon::LoadBitmap(int IDB_INPUT, int* rgb, int n) {
	for (int i = 0; i < n; i++) {
		PlayAttack.AddBitmap(IDB_INPUT, RGB(rgb[0], rgb[1], rgb[2]));
	}
}
void weapon::OnShow() {
	if (IsShow) {
		PlayAttack.SetDelayCount(5);
		PlayAttack.SetTopLeft(x, y);
		PlayAttack.OnShow();
	}
}
void weapon::AnimationReset() {
	PlayAttack.Reset();
}
void weapon::OnMove() {
	PlayAttack.OnMove();
	ImgW = PlayAttack.Width();
	ImgH = PlayAttack.Height();
}
void weapon::SetShow(bool input) {
	if (input) {
		IsShow = true;
	}
	else {
		IsShow = false;
	}
}
void weapon::SetOwner(std::string input) {
	owner = input;
}
void weapon::SetWeapon(int enemyX, int enemyY, bool enemyFaceR) {
	if (owner == "waddleDoo") {
		if (enemyFaceR) {
			x = enemyX + enemyImgW;
			y = enemyY - (ImgH - enemyImgH);
		}
		else {
				x = enemyX - ImgW;
				y = enemyY - (ImgH - enemyImgH);
			}
		}
		else if (owner == "sparky") {
			x = enemyX - 20;
			y = enemyY - 20;
		}
		else if (owner == "hotHead") {
			if (enemyFaceR) {
				x = enemyX + enemyImgW + 15;
				y = enemyY + 15;
			}
			else {
				x = enemyX - ImgW;
				y = enemyY + 15;
			}
		}
}
void weapon::SetXy(int input_x, int input_y) {
		x = input_x;
		y = input_y;
}
void weapon::SetAttackState(int input_attack_time, bool IsFacingR, int* input_Xy) {
		x = input_Xy[0];
		y = input_Xy[1];
		attack_time = input_attack_time;
		AttackIsFacingR = IsFacingR;
}
int* weapon::GetXy() {
		return new int[4]{ x, y, x +ImgW, y + ImgW };
}
int weapon::GetAttackTime() {
		return attack_time;
}
bool weapon::GetAttackFacingR() {
		return AttackIsFacingR;
}
bool weapon::WeaponIsShow() {
		return IsShow;
```

```
    }
    bool weapon::IsFinalBitmap() {
          return PlayAttack.IsFinalBitmap();
    }
    void weapon::YouAreLeft(bool YouAreLeft) {
          OtherFromL = !YouAreLeft;
    }
    int weapon::Width() {
          return PlayAttack.Width();
    }
    int weapon::Height() {
          return PlayAttack.Height();
    }
}
```

## enemy.cpp (base class enemy)

```
#include "stdafx.h"
#include "source/Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "source/audio.h"
#include "source/gamelib.h"
#include "Header.h"
namespace game_framework {
    const int frame_of_test = 5;
    enemy::enemy()
    {
          const int INIT_HP = 0;
          hp = INIT_HP;
          x = origin_x;
          y = origin_y;
          power = 1;
          floor = SIZE_Y - 60;
          IsFacingR = false;
          IsMovingL = true;
          IsMovingR = false;
          IsAttack = false;
          LastHurt = 0;
          LastAttack = 0;
          CanAttack = false;
    }
```

```
enemy::~enemy() {}
std::string enemy::GetKind() {
      return kind;
}
int* enemy::GetXy() {
      return new int[4]{ x, y, x + ImgW, y + ImgH };
}
int enemy::GetHp() {
      return hp;
}
int enemy::GetWidth() {
      return ImgW;
}
int enemy::GetHeight() {
      return ImgH;
}
int enemy::GetPower() {
      return power;
}
int enemy::Top() {
      return y;
}
int enemy::Left() {
      return x;
}
void enemy::BackX(bool fromL) {
      if (fromL) {
               SetXy(x + 60, y);
      }
      else {
               SetXy(x - 60, y);
      }
}
void enemy::Hurt(int input, int time) {
      if (abs(LastHurt - time) < 30) {
               return;
      }
      BackX(OtherFromL);
      LastHurt = time;
      hp -= input;
```

```
}
void enemy::Reset() {
	const int origin_x = SIZE_X - ImgW - frame_of_test;
	const int origin_y = SIZE_Y - 60 - ImgH;
	x = origin_x;
	y = origin_y;
	wL.SetShow(false);
	wR.SetShow(false);
	hp = 1;
}
void enemy::Attack(kirby k, int time) {
	if (abs(LastAttack - time) < 30) {
		return;
	}
	LastAttack = time;
	IsAttack = true;
}
void enemy::YouAreLeft(bool YouAreLeft) {
	OtherFromL = !YouAreLeft;
}
void enemy::OnShow()
{
	if (!IsAttack) {
		if (IsMovingR) {
			MovingR.SetDelayCount(3);
			MovingR.SetTopLeft(x, y);
			MovingR.OnShow();
			ImgH = MovingR.Height();
			ImgW = MovingL.Width();
		}
		else {
			MovingL.SetDelayCount(3);
			MovingL.SetTopLeft(x, y);
			MovingL.OnShow();
			ImgH = MovingL.Height();
			ImgW = MovingL.Width();
		}
	}
	else {
		if (OtherFromL) {
```

```
                        AttackL.SetDelayCount(5);
                        AttackL.SetTopLeft(x, y);
                        AttackL.OnShow();
                        wL.SetOwner(kind);
                        wL.SetWeapon(x, y, IsFacingR);
                        wL.SetShow(true);
                        wL.OnMove();
                        wL.OnShow();
                        if (AttackL.IsFinalBitmap()) {
                                AttackL.Reset();
                                wL.AnimationReset();
                                IsAttack = false;
                                wL.SetShow(false);
                        }
                        ImgH = AttackL.Height();
                        ImgW = AttackL.Width();
                }
                else {
                        AttackR.SetDelayCount(5);
                        AttackR.SetTopLeft(x, y);
                        AttackR.OnShow();
                        wR.SetOwner(kind);
                        wR.SetWeapon(x, y, IsFacingR);
                        wR.SetShow(true);
                        wR.OnMove();
                        wR.OnShow();
                        if (AttackR.IsFinalBitmap()) {
                                AttackR.Reset();
                                wR.AnimationReset();
                                IsAttack = false;
                                wR.SetShow(false);
                        }
                        ImgH = AttackR.Height();
                        ImgW = AttackR.Width();
                }
        }
}
void enemy::OnMove()
{
        // set moving XY
```

```
const int length = 2;
if (!IsAttack) {
	MovingL.OnMove();
	MovingR.OnMove();
	// set moving XY and frame of test
	if (IsMovingL && x >= Map->Left()) {
		if (IsFacingR) {
			IsFacingR = false;
		}
		SetXy(x - length, y);
	}
	else if (x <= Map->Left()) {
		IsMovingL = false;
		IsMovingR = true;
	}
	if (IsMovingR && x <= Map->Left() + Map->Width() - ImgW) {
		if (!IsFacingR) {
			IsFacingR = true;
		}
		SetXy(x + length, y);
	}
	else if (x >= Map->Left() + Map->Width() - ImgW) {
		IsMovingR = false;
		IsMovingL = true;
	}
}
else {
	AttackR.OnMove();
	AttackL.OnMove();
}
if (game_state_counter % 20 == 0) {
	int* temp = Map->GetXy();
	if (x_old - x == m_x_old - temp[0] && y_old - y == m_y_old - temp[1] && !IsAttack) {
		IsMovingR = !IsMovingR;
		IsMovingL = !IsMovingL;
	}
	else {
		x_old = x;
		y_old = y;
		m_y_old = temp[1];
```

```
                        m_x_old = temp[0];
                }
                delete[] temp;
            }
    }
    void enemy::LoadBitmap() {}
    void enemy::SetMap(CMovingBitmap * input) {
          Map = input;
    }
    void enemy::SetThings(vector<thing*> input) {
          StarBlockList = input;
    }
    bool enemy::EnemyFacingR() {
          return IsFacingR;
    }
    void enemy::SetXy(int x_in, int y_in) {
       int aXy[4] = { x_in, y_in, x_in + ImgW, y_in + ImgH };
       bool result = true;
       if (!StarBlockList.empty()) {
          for (int k = 0; k < int(StarBlockList.size()); k++) {
                if (StarBlockList[k] != nullptr && StarBlockList[k]->GetShow()) {
                     int* bXy = StarBlockList[k]->GetXy();
                     int i = 0, n = 1;
                     for (int count = 0; count < 2; count++) {
                       for (int _count = 0; _count < 2; _count++) {
                            if (aXy[i] > bXy[0] && aXy[i] < bXy[2] && aXy[n] > bXy[1] && aXy[n] <
bXy[3]) {
                                     result = false;
                                     IsMovingL = !IsMovingL;
                                     IsMovingR = !IsMovingR;
                                     IsFacingR = !IsFacingR;
                            }
                            n += 2;
                       }
                       n = 1;
                       i += 2;
                     }
                     i = 0, n = 1;
                     for (int count = 0; count < 2; count++) {
                          for (int _count = 0; _count < 2; _count++) {
                                if (bXy[i] > aXy[0] && bXy[i] < aXy[2] && bXy[n] > aXy[1] && bXy[n] <
aXy[3]) {
                                    result = false;
                                    IsMovingL = !IsMovingL;
                                    IsMovingR = !IsMovingR;
                                    IsFacingR = !IsFacingR;
                                }
                                 n += 2;
                            }
                             n = 1;
                              i += 2;
                        }
                        delete[] bXy;
                        }
            }
          }
          if (result) {
            x = x_in;
            y = y_in;
           }
      }
      weapon enemy::GetWeapon() {
            if (IsFacingR) {
                    return wR;
            }
            else {
                    return wL;
            }
      }
      bool enemy::EnemyHasWeapon() {
            return HasWeapon;
      }
      bool enemy::GetCanAttack() {
            return CanAttack;
      }
}
```

## waddle.cpp (derived classes of enemy)

```
#include "stdafx.h"
#include "source/Resource.h"
#include <mmsystem.h>
```

```
#include <ddraw.h>
#include "source/audio.h"
#include "source/gamelib.h"
#include "Header.h"
namespace game_framework {
    const int frame_of_test = 5;
    void waddle::LoadBitmap()
    {
        // load walk right
        char *walk_right[10] = { ".\\res\\waddledee\\walk\\walkR1.bmp",
".\\res\\waddledee\\walk\\walkR2.bmp", ".\\res\\waddledee\\walk\\walkR3.bmp",
".\\res\\waddledee\\walk\\walkR4.bmp", ".\\res\\waddledee\\walk\\walkR5.bmp",
".\\res\\waddledee\\walk\\walkR6.bmp", ".\\res\\waddledee\\walk\\walkR7.bmp",
".\\res\\waddledee\\walk\\walkR8.bmp", ".\\res\\waddledee\\walk\\walkR9.bmp",
".\\res\\waddledee\\walk\\walkR10.bmp" };
        for (int i = 0; i < 10; i++) {
                MovingR.AddBitmap(walk_right[i], RGB(255, 0, 0));
        }
        // load walk left
        char *walk_left[10] = { ".\\res\\waddledee\\walk\\walkL1.bmp",
".\\res\\waddledee\\walk\\walkL2.bmp", ".\\res\\waddledee\\walk\\walkL3.bmp",
".\\res\\waddledee\\walk\\walkL4.bmp", ".\\res\\waddledee\\walk\\walkL5.bmp",
".\\res\\waddledee\\walk\\walkL6.bmp", ".\\res\\waddledee\\walk\\walkL7.bmp",
".\\res\\waddledee\\walk\\walkL8.bmp", ".\\res\\waddledee\\walk\\walkL9.bmp",
".\\res\\waddledee\\walk\\walkL10.bmp" };
        for (int i = 0; i < 10; i++) {
                MovingL.AddBitmap(walk_left[i], RGB(255, 0, 0));
        }
        // load attack right
        for (int i = 0; i < 5; i++) {
                AttackR.AddBitmap(".\\res\\waddledee\\walk\\walkR1.bmp");
        }
        // load attack left
        for (int i = 0; i < 5; i++) {
                AttackL.AddBitmap(".\\res\\waddledee\\walk\\walkL1.bmp");
        }
    }
    void waddle::Reset() {
        hp = 2;
        power = 1;
        kind = "waddle";
        IsFacingR = false;
        IsMovingL = true;
        IsMovingR = false;
        IsAttack = false;
        LastHurt = 0;
        HasWeapon = false;
    }
    void waddleDoo::LoadBitmap()
    {
        // load walk right
        char *walk_right[9] = { ".\\res\\waddledoo\\walk\\walkR1.bmp",
".\\res\\waddledoo\\walk\\walkR2.bmp", ".\\res\\waddledoo\\walk\\walkR3.bmp",
".\\res\\waddledoo\\walk\\walkR4.bmp", ".\\res\\waddledoo\\walk\\walkR5.bmp",
".\\res\\waddledoo\\walk\\walkR6.bmp", ".\\res\\waddledoo\\walk\\walkR7.bmp",
".\\res\\waddledoo\\walk\\walkR8.bmp", ".\\res\\waddledoo\\walk\\walkR9.bmp" };
        for (int i = 0; i < 9; i++) {
            MovingR.AddBitmap(walk_right[i], RGB(84, 109, 142));
        }
        // load walk left
        char *walk_left[9] = { ".\\res\\waddledoo\\walk\\walkL1.bmp",
".\\res\\waddledoo\\walk\\walkL2.bmp", ".\\res\\waddledoo\\walk\\walkL3.bmp",
".\\res\\waddledoo\\walk\\walkL4.bmp", ".\\res\\waddledoo\\walk\\walkL5.bmp",
".\\res\\waddledoo\\walk\\walkL6.bmp", ".\\res\\waddledoo\\walk\\walkL7.bmp",
".\\res\\waddledoo\\walk\\walkL8.bmp", ".\\res\\waddledoo\\walk\\walkL9.bmp" };
        for (int i = 0; i < 9; i++) {
            MovingL.AddBitmap(walk_left[i], RGB(84, 109, 142));
        }
        // load attack right
        for (int i = 0; i < 10; i++) {
            AttackR.AddBitmap(".\\res\\waddledoo\\walk\\walkR1.bmp");
        }
        // load attack left
        for (int i = 0; i < 10; i++) {
            AttackL.AddBitmap(".\\res\\waddledoo\\walk\\walkL5.bmp");
        }
        // load weapon bitmap
        char* weapon_left[10] = { ".\\res\\weapon\\waddledoo\\attackL1.bmp",
".\\res\\weapon\\waddledoo\\attackL2.bmp", ".\\res\\weapon\\waddledoo\\attackL3.bmp",
".\\res\\weapon\\waddledoo\\attackL4.bmp", ".\\res\\weapon\\waddledoo\\attackL5.bmp",
".\\res\\weapon\\waddledoo\\attackL6.bmp", ".\\res\\weapon\\waddledoo\\attackL7.bmp",
".\\res\\weapon\\waddledoo\\attackL8.bmp", ".\\res\\weapon\\waddledoo\\attackL9.bmp",
".\\res\\weapon\\waddledoo\\attackL10.bmp" };
		char* weapon_right[10] = { ".\\res\\weapon\\waddledoo\\attackR1.bmp",
".\\res\\weapon\\waddledoo\\attackR2.bmp", ".\\res\\weapon\\waddledoo\\attackR3.bmp",
".\\res\\weapon\\waddledoo\\attackR4.bmp", ".\\res\\weapon\\waddledoo\\attackR5.bmp",
".\\res\\weapon\\waddledoo\\attackR6.bmp", ".\\res\\weapon\\waddledoo\\attackR7.bmp",
".\\res\\weapon\\waddledoo\\attackR8.bmp", ".\\res\\weapon\\waddledoo\\attackR9.bmp",
".\\res\\weapon\\waddledoo\\attackR10.bmp" };
		int rgb[3] = { 255, 255, 255 };
		wL.LoadBitmap(weapon_left, rgb, 10);
		wR.LoadBitmap(weapon_right, rgb, 10);
	}
	void waddleDoo::Reset() {
		hp = 1;
		power = 1;
		kind = "waddleDoo";
		IsFacingR = false;
		IsMovingL = true;
		IsMovingR = false;
		IsAttack = false;
		LastHurt = 0;
		HasWeapon = true;
	}
	void sparky::LoadBitmap()
	{
		// load walk right
		char *walk_right[6] = { ".\\res\\sparky\\R1.bmp", ".\\res\\sparky\\R2.bmp",
".\\res\\sparky\\R3.bmp", ".\\res\\sparky\\R4.bmp", ".\\res\\sparky\\R5.bmp",
".\\res\\sparky\\R6.bmp" };
		for (int i = 0; i < 6; i++) {
			MovingR.AddBitmap(walk_right[i], RGB(0, 0, 255));
		}
		// load walk left
		char *walk_left[6] = { ".\\res\\sparky\\L1.bmp", ".\\res\\sparky\\L2.bmp",
".\\res\\sparky\\L3.bmp", ".\\res\\sparky\\L4.bmp", ".\\res\\sparky\\L5.bmp",
".\\res\\sparky\\L6.bmp" };
		for (int i = 0; i < 6; i++) {
			MovingL.AddBitmap(walk_left[i], RGB(0, 0, 255));
		}
		// load attack right
		for (int i = 0; i < 2; i++) {
			AttackR.AddBitmap(".\\res\\sparky\\attack1.bmp", RGB(0, 0, 255));
			AttackR.AddBitmap(".\\res\\sparky\\attack2.bmp", RGB(0, 0, 255));
		}
		// load attack left
		for (int i = 0; i < 2; i++) {
			AttackL.AddBitmap(".\\res\\sparky\\attack1.bmp", RGB(0, 0, 255));
			AttackL.AddBitmap(".\\res\\sparky\\attack2.bmp", RGB(0, 0, 255));
		}
		// load weapon bitmap
		char* weapon[4] = { ".\\res\\weapon\\sparky\\attack1.bmp",
".\\res\\weapon\\sparky\\attack2.bmp", ".\\res\\weapon\\sparky\\attack3.bmp",
".\\res\\weapon\\sparky\\attack4.bmp" };
		int rgb[3] = { 0, 0, 255 };
		wL.LoadBitmap(weapon, rgb, 4);
		wR.LoadBitmap(weapon, rgb, 4);
	}
	void sparky::Reset() {
		hp = 1;
		power = 1;
		kind = "sparky";
		IsFacingR = false;
		IsMovingL = true;
		IsMovingR = false;
		IsAttack = false;
		LastHurt = 0;
		HasWeapon = true;
	}
	void hotHead::LoadBitmap()
	{
		// load walk right
		char *walk_right[8] = { ".\\res\\hotHead\\walkR1.bmp", ".\\res\\hotHead\\walkR2.bmp",
".\\res\\hotHead\\walkR3.bmp", ".\\res\\hotHead\\walkR4.bmp", ".\\res\\hotHead\\walkR5.bmp",
".\\res\\hotHead\\walkR6.bmp", ".\\res\\hotHead\\walkR7.bmp", ".\\res\\hotHead\\walkR8.bmp" };
		for (int i = 0; i < 8; i++) {
			MovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
		}
		// load walk left
		char *walk_left[8] = { ".\\res\\hotHead\\walkL1.bmp", ".\\res\\hotHead\\walkL2.bmp",
".\\res\\hotHead\\walkL3.bmp", ".\\res\\hotHead\\walkL4.bmp", ".\\res\\hotHead\\walkL5.bmp",
".\\res\\hotHead\\walkL6.bmp", ".\\res\\hotHead\\walkL7.bmp", ".\\res\\hotHead\\walkL8.bmp" };
          for (int i = 0; i < 8; i++) {
                   MovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
          }
          // load attack right
          for (int i = 0; i < 3; i++) {
                   AttackR.AddBitmap(".\\res\\hotHead\\attackR1.bmp", RGB(255, 255, 255));
                   AttackR.AddBitmap(".\\res\\hotHead\\attackR2.bmp", RGB(255, 255, 255));
                   AttackR.AddBitmap(".\\res\\hotHead\\attackR3.bmp", RGB(255, 255, 255));
          }
          // load attack left
          for (int i = 0; i < 3; i++) {
                   AttackL.AddBitmap(".\\res\\hotHead\\attackL1.bmp", RGB(255, 255, 255));
                   AttackL.AddBitmap(".\\res\\hotHead\\attackL2.bmp", RGB(255, 255, 255));
                   AttackL.AddBitmap(".\\res\\hotHead\\attackL3.bmp", RGB(255, 255, 255));
          }
          // load weapon bitmap
          char* weapon_left[4] = { ".\\res\\weapon\\hotHead\\attackL1.bmp",
".\\res\\weapon\\hotHead\\attackL2.bmp", ".\\res\\weapon\\hotHead\\attackL3.bmp",
".\\res\\weapon\\hotHead\\attackL4.bmp" };
          char* weapon_right[4] = { ".\\res\\weapon\\hotHead\\attackR1.bmp",
".\\res\\weapon\\hotHead\\attackR2.bmp", ".\\res\\weapon\\hotHead\\attackR3.bmp",
".\\res\\weapon\\hotHead\\attackR4.bmp" };
          int rgb[3] = { 255, 255, 255 };
          wL.LoadBitmap(weapon_left, rgb, 4);
          wR.LoadBitmap(weapon_right, rgb, 4);
    }
    void hotHead::Reset() {
          hp = 1;
          power = 1;
          kind = "hotHead";
          IsFacingR = false;
          IsMovingL = true;
          IsMovingR = false;
          IsAttack = false;
          LastHurt = 0;
          HasWeapon = true;
    }
    void droppy::LoadBitmap()
    {
        // load walk right
        char *walk_right[8] = { ".\\res\\droppy\\walkR1.bmp", ".\\res\\droppy\\walkR2.bmp",
".\\res\\droppy\\walkR3.bmp", ".\\res\\droppy\\walkR4.bmp", ".\\res\\droppy\\walkR5.bmp",
".\\res\\droppy\\walkR6.bmp", ".\\res\\droppy\\walkR7.bmp", ".\\res\\droppy\\walkR8.bmp" };
        for (int i = 0; i < 8; i++) {
            MovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
        }
        for (int i = 6; i >= 0; i--) {
            MovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
        }
        // load walk left
        char *walk_left[8] = { ".\\res\\droppy\\walkL1.bmp", ".\\res\\droppy\\walkL2.bmp",
".\\res\\droppy\\walkL3.bmp", ".\\res\\droppy\\walkL4.bmp", ".\\res\\droppy\\walkL5.bmp",
".\\res\\droppy\\walkL6.bmp", ".\\res\\droppy\\walkL7.bmp", ".\\res\\droppy\\walkL8.bmp" };
        for (int i = 0; i < 8; i++) {
            MovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
        }
        for (int i = 6; i >= 0; i--) {
            MovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
        }
        // load attack right
        for (int i = 0; i < 5; i++) {
            AttackR.AddBitmap(".\\res\\droppy\\walkR1.bmp");
        }
        // load attack left
        for (int i = 0; i < 5; i++) {
            AttackL.AddBitmap(".\\res\\droppy\\walkL1.bmp");
        }
    }
    void droppy::Reset() {
        hp = 2;
        power = 1;
        kind = "droppy";
        IsFacingR = false;
        IsMovingL = true;
        IsMovingR = false;
        IsAttack = false;
        LastHurt = 0;
        HasWeapon = false;
    }
    void bigWaddle::LoadBitmap() {
          // load walk right
          char *walk_right[10] = { ".\\res\\bigwaddledee\\walkR1.bmp",
".\\res\\bigwaddledee\\walkR2.bmp", ".\\res\\bigwaddledee\\walkR3.bmp",
".\\res\\bigwaddledee\\walkR4.bmp", ".\\res\\bigwaddledee\\walkR5.bmp",
".\\res\\bigwaddledee\\walkR6.bmp", ".\\res\\bigwaddledee\\walkR7.bmp",
".\\res\\bigwaddledee\\walkR8.bmp", ".\\res\\bigwaddledee\\walkR9.bmp",
".\\res\\bigwaddledee\\walkR10.bmp" };
          for (int i = 0; i < 10; i++) {
                   MovingR.AddBitmap(walk_right[i], RGB(255, 0, 0));
          }
          // load walk left
          char *walk_left[10] = { ".\\res\\bigwaddledee\\walkL1.bmp",
".\\res\\bigwaddledee\\walkL2.bmp", ".\\res\\bigwaddledee\\walkL3.bmp",
".\\res\\bigwaddledee\\walkL4.bmp", ".\\res\\bigwaddledee\\walkL5.bmp",
".\\res\\bigwaddledee\\walkL6.bmp", ".\\res\\bigwaddledee\\walkL7.bmp",
".\\res\\bigwaddledee\\walkL8.bmp", ".\\res\\bigwaddledee\\walkL9.bmp",
".\\res\\bigwaddledee\\walkL10.bmp" };
          for (int i = 0; i < 10; i++) {
                   MovingL.AddBitmap(walk_left[i], RGB(255, 0, 0));
          }
          // load attack right
          for (int i = 0; i < 5; i++) {
                   AttackR.AddBitmap(".\\res\\bigwaddledee\\walkR1.bmp");
          }
          // load attack left
          for (int i = 0; i < 5; i++) {
                   AttackL.AddBitmap(".\\res\\bigwaddledee\\walkL1.bmp");
          }
    }
    void bigWaddle::Reset() {
          hp = 2;
          power = 1;
          kind = "bigWaddle";
          IsFacingR = false;
          IsMovingL = true;
          IsMovingR = false;
          IsAttack = false;
          LastHurt = 0;
```

```
		HasWeapon = false;
	}
	void kingDedede::LoadBitmap() {
		// load walk right
		char *walk_right[7] = { ".\\res\\king_dedede\\walk\\walkR1.bmp",
".\\res\\king_dedede\\walk\\walkR2.bmp", ".\\res\\king_dedede\\walk\\walkR3.bmp",
".\\res\\king_dedede\\walk\\walkR4.bmp", ".\\res\\king_dedede\\walk\\walkR5.bmp",
".\\res\\king_dedede\\walk\\walkR6.bmp", ".\\res\\king_dedede\\walk\\walkR7.bmp"};
		for (int i = 0; i < 7; i++) {
			MovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
		}
		// load walk left
		char *walk_left[7] = { ".\\res\\king_dedede\\walk\\walkL1.bmp",
".\\res\\king_dedede\\walk\\walkL2.bmp", ".\\res\\king_dedede\\walk\\walkL3.bmp",
".\\res\\king_dedede\\walk\\walkL4.bmp", ".\\res\\king_dedede\\walk\\walkL5.bmp",
".\\res\\king_dedede\\walk\\walkL6.bmp", ".\\res\\king_dedede\\walk\\walkL7.bmp" };
		for (int i = 0; i < 7; i++) {
			MovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
		}
		// load attack right
		char *attack_right[8] = { ".\\res\\king_dedede\\attack\\attackR1.bmp",
".\\res\\king_dedede\\attack\\attackR2.bmp", ".\\res\\king_dedede\\attack\\attackR3.bmp",
".\\res\\king_dedede\\attack\\attackR4.bmp", ".\\res\\king_dedede\\attack\\attackR5.bmp",
".\\res\\king_dedede\\attack\\attackR6.bmp", ".\\res\\king_dedede\\attack\\attackR7.bmp",
".\\res\\king_dedede\\attack\\attackR8.bmp" };
		for (int i = 0; i < 8; i++) {
			AttackR.AddBitmap(attack_right[i], RGB(255, 255, 255));
		}
		// load attack left
		char *attack_left[8] = { ".\\res\\king_dedede\\attack\\attackL1.bmp",
".\\res\\king_dedede\\attack\\attackL2.bmp", ".\\res\\king_dedede\\attack\\attackL3.bmp",
".\\res\\king_dedede\\attack\\attackL4.bmp", ".\\res\\king_dedede\\attack\\attackL5.bmp",
".\\res\\king_dedede\\attack\\attackL6.bmp", ".\\res\\king_dedede\\attack\\attackL7.bmp",
".\\res\\king_dedede\\attack\\attackL8.bmp" };
		for (int i = 0; i < 8; i++) {
			AttackL.AddBitmap(attack_left[i], RGB(255, 255, 255));
		}
		// Load hurt right
		for (int i = 0;i < 5;i++) {
			HurtR.AddBitmap(".\\res\\king_dedede\\hurt\\hurtR1.bmp", RGB(255, 255, 255));
	}
	for (int i = 0;i < 5;i++) {
		HurtR.AddBitmap(".\\res\\king_dedede\\hurt\\hurtR2.bmp", RGB(255, 255, 255));
	}
	// Load hurt left
	for (int i = 0;i < 2;i++) {
		HurtL.AddBitmap(".\\res\\king_dedede\\hurt\\hurtL1.bmp", RGB(255, 255, 255));
	}
	for (int i = 0;i < 2;i++) {
		HurtL.AddBitmap(".\\res\\king_dedede\\hurt\\hurtL2.bmp", RGB(255, 255, 255));
	}
	char* weapon_left = { ".\\res\\weapon\\1.bmp" };
	int rgb[3] = { 255, 255, 255 };
	wL.LoadBitmap(weapon_left, rgb, 10);
	wR.LoadBitmap(weapon_left, rgb, 10);
}
void kingDedede::Reset() {
	hp = 10;
	power = 2;
	ImgH = MovingL.Height();
	ImgW = MovingL.Width();
	kind = "kingDedede";
	IsFacingR = false;
	IsMovingL = true;
	IsMovingR = false;
	IsAttack = false;
	IsHurt = false;
	LastHurt = 0;
	HasWeapon = true;
	CanAttack = false;
}
void kingDedede::OnShow()
{
	if (IsHurt) {
		if (!IsFacingR) {
			HurtL.SetDelayCount(3);
			HurtL.SetTopLeft(x, y);
			HurtL.OnMove();
			HurtL.OnShow();
			if (HurtL.IsFinalBitmap()) {
```

```
                IsHurt = false;
                HurtL.Reset();
            }
        }
        else {
            HurtR.SetDelayCount(3);
            HurtR.SetTopLeft(x, y);
            HurtR.OnMove();
            HurtR.OnShow();
            if (HurtR.IsFinalBitmap()) {
                IsHurt = false;
                HurtR.Reset();
            }
        }
    }
    else if (!IsAttack) {
        if (IsMovingR) {
            MovingR.SetDelayCount(3);
            MovingR.SetTopLeft(x, y);
            MovingR.OnShow();
            ImgH = MovingR.Height();
            ImgW = MovingL.Width();
        }
        else {
            MovingL.SetDelayCount(3);
            MovingL.SetTopLeft(x, y);
            MovingL.OnShow();
            ImgH = MovingL.Height();
            ImgW = MovingL.Width();
        }
    }
    else {
        if (OtherFromL) {
            AttackL.SetDelayCount(5);
            AttackL.SetTopLeft(x, y - 50);
            AttackL.OnShow();
            wL.SetOwner(kind);
            wL.SetWeapon(x, y - 50, IsFacingR);
            wL.SetShow(true);
            wL.OnMove();
				wL.OnShow();
				if (AttackL.IsFinalBitmap()) {
					AttackL.Reset();
					wL.AnimationReset();
					IsAttack = false;
					wL.SetShow(true);
				}
				ImgH = AttackL.Height();
				ImgW = AttackL.Width();
			}
			else {
				AttackR.SetDelayCount(5);
				AttackR.SetTopLeft(x, y - 50);
				AttackR.OnShow();
				wR.SetOwner(kind);
				wR.SetWeapon(x, y - 50, IsFacingR);
				wR.SetShow(true);
				wR.OnMove();
				wR.OnShow();
				if (AttackR.IsFinalBitmap()) {
					AttackR.Reset();
					wR.AnimationReset();
					IsAttack = false;
					wR.SetShow(true);
				}
				ImgH = AttackR.Height();
				ImgW = AttackR.Width();
			}
		}
	}
	void kingDedede::OnMove()
	{
		// set moving XY
		const int length = 2
		if (!IsAttack && !IsHurt) {
			MovingL.OnMove();
			MovingR.OnMove();
			// set moving XY and frame of test
			if (IsMovingL && x >= Map->Left()) {
				if (IsFacingR) {
```

```
                                IsFacingR = false;
                        }
                        SetXy(x - length, y);
                }
                else if (x <= Map->Left()) {
                        IsMovingL = false;
                        IsMovingR = true;
                }

                if (IsMovingR && x <= Map->Left() + Map->Width() - ImgW) {
                        if (!IsFacingR) {
                                IsFacingR = true;
                        }
                        SetXy(x + length, y);
                }
                else if (x >= Map->Left() + Map->Width() - ImgW) {
                        IsMovingR = false;
                        IsMovingL = true;
                }
        }
        else {
                AttackR.OnMove();
                AttackL.OnMove();
        }
    }
    void kingDedede::Hurt(int input, int time) {
        if (abs(LastHurt - time) < 30) {
                return;
        }
        if (IsAttack) {
                return;
        }
        IsHurt = true;
        BackX(OtherFromL);
        LastHurt = time;
        hp -= input;
    }
}
```

## kirby.cpp (base class kirby & derived classes)

```
#include "stdafx.h"
```

```
#include "source/Resource.h"
#include <mmsystem.h>
#include <ddraw.h>
#include "source/audio.h"
#include "source/gamelib.h"
#include "Header.h"
namespace game_framework {
    const int frame_of_test = 5;
    const int temp_floor = 60;
    // kirby constructor
    kirby::kirby() {
          StageReSet(10, "normal_kirby");
    }
    void kirby::StageReSet(int hp_left, std::string kind) {
          const int origin_x = frame_of_test;
          const int origin_y = SIZE_Y - temp_floor - ImgH;
          const int INIT_VELOCITY = 18;
          const int INIT_FLY_VELOCITY = 9;
          const int INIT_HP = hp_left;
          BoundaryTop = SIZE_Y / 2;
          BoundaryLeft = SIZE_X / 2;
          BoundaryRight = 2560 - BoundaryLeft;
          now_img_h = ImgH;
          now_img_w = ImgW;
          init_velocity = INIT_VELOCITY;
          velocity = INIT_VELOCITY;
          init_fly_velocity = INIT_FLY_VELOCITY;
          fly_velocity = INIT_FLY_VELOCITY;
          hp = INIT_HP;
          x = origin_x;
          y = origin_y;
          sky_top = 0;
          floor = SIZE_Y - temp_floor;
          IsMovingL = false;
          IsMovingR = false;
          IsFacingR = true;
          IsDown = false;
          IsAttack = false;
          InAir = false;
          IsRising = true;
```

```
            IsJumping = false;
            IsFlying = false;
            IsFat = false;
            FlyUp = false;
            LastHurt = 0;
            IsHurt = false;
            IsEaten = false;
            OtherFromL = false;
            YouAreGround = true;
            IsRun = false;
            IsHack = false;
            EatenEnemy = "";
            kirby_kind = kind;
      }
      kirby::~kirby() {}
      void kirby::LoadBitmap()
      {
            // load walk left
            char *walk_left[10] = { ".\\res\\walk\\walkL1.bmp", ".\\res\\walk\\walkL2.bmp",
".\\res\\walk\\walkL3.bmp", ".\\res\\walk\\walkL4.bmp", ".\\res\\walk\\walkL5.bmp",
".\\res\\walk\\walkL6.bmp", ".\\res\\walk\\walkL7.bmp", ".\\res\\walk\\walkL8.bmp",
".\\res\\walk\\walkL9.bmp", ".\\res\\walk\\walkL10.bmp" };
            for (int i = 0; i < 10; i++) {
                  KirbyMovingL.AddBitmap(walk_left[i], RGB(255, 0, 0));
            }
            // load walk right
            char *walk_right[10] = { ".\\res\\walk\\walkR1.bmp", ".\\res\\walk\\walkR2.bmp",
".\\res\\walk\\walkR3.bmp", ".\\res\\walk\\walkR4.bmp", ".\\res\\walk\\walkR5.bmp",
".\\res\\walk\\walkR6.bmp", ".\\res\\walk\\walkR7.bmp", ".\\res\\walk\\walkR8.bmp",
".\\res\\walk\\walkR9.bmp", ".\\res\\walk\\walkR10.bmp" };
            for (int i = 0; i < 10; i++) {
                  KirbyMovingR.AddBitmap(walk_right[i], RGB(255, 0, 0));
            }
            // load fat walk left
            char *fat_walk_left[8] = { ".\\res\\walk\\fat_walkL1.bmp", ".\\res\\walk\\fat_walkL2.bmp",
".\\res\\walk\\fat_walkL3.bmp", ".\\res\\walk\\fat_walkL4.bmp", ".\\res\\walk\\fat_walkL5.bmp",
".\\res\\walk\\fat_walkL6.bmp", ".\\res\\walk\\fat_walkL7.bmp", ".\\res\\walk\\fat_walkL8.bmp" };
            for (int i = 0; i < 8; i++) {
                  KirbyFatMovingL.AddBitmap(fat_walk_left[i], RGB(255, 255, 255));
            }
```

```
          // load fat walk right
          char *fat_walk_right[8] = { ".\\res\\walk\\fat_walkR1.bmp", ".\\res\\walk\\fat_walkR2.bmp",
".\\res\\walk\\fat_walkR3.bmp", ".\\res\\walk\\fat_walkR4.bmp", ".\\res\\walk\\fat_walkR5.bmp",
".\\res\\walk\\fat_walkR6.bmp", ".\\res\\walk\\fat_walkR7.bmp", ".\\res\\walk\\fat_walkR8.bmp" };
          for (int i = 0; i < 8; i++) {
                    KirbyFatMovingR.AddBitmap(fat_walk_right[i], RGB(255, 255, 255));
          }
          // load stand and wink right
          int count = 12;
          while (count-- > 0) {
                    KirbyStand.AddBitmap(IDB_STAND, RGB(255, 0, 0));
          }
          KirbyStand.AddBitmap(IDB_CLOSE_EYES, RGB(255, 0, 0));
          count = 12;
          while (count-- > 0) {
                    KirbyStand.AddBitmap(IDB_STAND, RGB(255, 0, 0));
          }
          KirbyStand.AddBitmap(IDB_CLOSE_EYES, RGB(255, 0, 0));
          KirbyStand.AddBitmap(IDB_STAND, RGB(255, 0, 0));
          KirbyStand.AddBitmap(IDB_CLOSE_EYES, RGB(255, 0, 0));
          // load stand and wink left
          count = 12;
          while (count-- > 0) {
                    KirbyStandL.AddBitmap(IDB_STANDL, RGB(255, 0, 0));
          }
          KirbyStandL.AddBitmap(IDB_CLOSE_EYES_L, RGB(255, 0, 0));
          count = 12;
          while (count-- > 0) {
                    KirbyStandL.AddBitmap(IDB_STANDL, RGB(255, 0, 0));
          }
          KirbyStandL.AddBitmap(IDB_CLOSE_EYES_L, RGB(255, 0, 0));
          KirbyStandL.AddBitmap(IDB_STANDL, RGB(255, 0, 0));
          KirbyStandL.AddBitmap(IDB_CLOSE_EYES_L, RGB(255, 0, 0));
          // load fat stand and wink right
          count = 12;
          while (count-- > 0) {
                    KirbyFatStand.AddBitmap(".\\res\\basic\\fat_basicR1.bmp", RGB(255, 255, 255));
          }
          KirbyFatStand.AddBitmap(".\\res\\basic\\fat_basicR2.bmp", RGB(255, 255, 255));
          count = 12;
```

```
		while (count-- > 0) {
			KirbyFatStand.AddBitmap(".\\res\\basic\\fat_basicR1.bmp", RGB(255, 255, 255));
		}
		KirbyFatStand.AddBitmap(".\\res\\basic\\fat_basicR2.bmp", RGB(255, 255, 255));
		KirbyFatStand.AddBitmap(".\\res\\basic\\fat_basicR1.bmp", RGB(255, 255, 255));
		KirbyFatStand.AddBitmap(".\\res\\basic\\fat_basicR2.bmp", RGB(255, 255, 255));
		// load fat stand and wink left
		count = 12;
		while (count-- > 0) {
			KirbyFatStandL.AddBitmap(".\\res\\basic\\fat_basicL1.bmp", RGB(255, 255, 255));
		}
		KirbyFatStandL.AddBitmap(".\\res\\basic\\fat_basicL2.bmp", RGB(255, 255, 255));
		count = 12;
		while (count-- > 0) {
			KirbyFatStandL.AddBitmap(".\\res\\basic\\fat_basicL1.bmp", RGB(255, 255, 255));
		}
		KirbyFatStandL.AddBitmap(".\\res\\basic\\fat_basicL2.bmp", RGB(255, 255, 255));
		KirbyFatStandL.AddBitmap(".\\res\\basic\\fat_basicL1.bmp", RGB(255, 255, 255));
		KirbyFatStandL.AddBitmap(".\\res\\basic\\fat_basicL2.bmp", RGB(255, 255, 255));
		// load down right and left
		KirbyDownR.LoadBitmap(IDB_DOWNR1, RGB(255, 255, 255));
		KirbyDownL.LoadBitmap(IDB_DOWNL1, RGB(255, 255, 255));
		// load down attack right and left
		count = 3;
		while (count-- > 0) {
			KirbyDownAttackR.AddBitmap(IDB_DOWNR2, RGB(255, 255, 255));
			KirbyDownAttackL.AddBitmap(IDB_DOWNL2, RGB(255, 255, 255));
		}
		// load jump right
		char *jump_right[10] = { ".\\res\\jump\\jumpR1.bmp", ".\\res\\jump\\jumpR2.bmp",
".\\res\\jump\\jumpR3.bmp", ".\\res\\jump\\jumpR4.bmp", ".\\res\\jump\\jumpR5.bmp",
".\\res\\jump\\jumpR6.bmp", ".\\res\\jump\\jumpR7.bmp", ".\\res\\jump\\jumpR8.bmp",
".\\res\\jump\\jumpR9.bmp", ".\\res\\jump\\jumpR10.bmp" };
		for (int i = 0; i < 10; i++) {
			KirbyJumpR.AddBitmap(jump_right[i], RGB(255, 0, 0));
		}
		// load jump left
		char *jump_left[10] = { ".\\res\\jump\\jumpL1.bmp", ".\\res\\jump\\jumpL2.bmp",
".\\res\\jump\\jumpL3.bmp", ".\\res\\jump\\jumpL4.bmp", ".\\res\\jump\\jumpL5.bmp",
".\\res\\jump\\jumpL6.bmp", ".\\res\\jump\\jumpL7.bmp", ".\\res\\jump\\jumpL8.bmp",
".\\res\\jump\\jumpL9.bmp", ".\\res\\jump\\jumpL10.bmp" };
          for (int i = 0; i < 10; i++) {
                    KirbyJumpL.AddBitmap(jump_left[i], RGB(255, 0, 0));
          }
          // load scream right and left
          KirbyScreamR.AddBitmap(IDB_SCREAMR1, RGB(255, 255, 255));
          count = 6;
          while (count-- > 0) {
                    KirbyScreamR.AddBitmap(IDB_SCREAMR2, RGB(255, 255, 255));
          }
          count = 6;
          while (count-- > 0) {
                    KirbyScreamR.AddBitmap(IDB_SCREAMR3, RGB(255, 255, 255));
                    KirbyScreamR.AddBitmap(IDB_SCREAMR4, RGB(255, 255, 255));
          }
          count = 5;
          while (count-- > 0) {
                    KirbyScreamR.AddBitmap(IDB_SCREAMR2, RGB(255, 255, 255));
          }
          KirbyScreamR.AddBitmap(IDB_SCREAMR5, RGB(255, 255, 255));

          KirbyScreamL.AddBitmap(IDB_SCREAML1, RGB(255, 255, 255));
          count = 6;
          while (count-- > 0) {
                    KirbyScreamL.AddBitmap(IDB_SCREAML2, RGB(255, 255, 255));
          }
          count = 6;
          while (count-- > 0) {
                    KirbyScreamL.AddBitmap(IDB_SCREAML3, RGB(255, 255, 255));
                    KirbyScreamL.AddBitmap(IDB_SCREAML4, RGB(255, 255, 255));
          }
          count = 5;
          while (count-- > 0) {
                    KirbyScreamL.AddBitmap(IDB_SCREAML2, RGB(255, 255, 255));
          }
          KirbyScreamL.AddBitmap(IDB_SCREAML5, RGB(255, 255, 255));

          // load flyR
          KirbyFlyR.AddBitmap(".\\res\\fly\\flyR1.bmp", RGB(255, 0, 0));
          KirbyFlyR.AddBitmap(".\\res\\fly\\flyR2.bmp", RGB(255, 0, 0));
          KirbyFlyR.AddBitmap(".\\res\\fly\\flyR3.bmp", RGB(255, 0, 0));
          count = 3;
          while (count-- > 0) {
                    KirbyFlyR.AddBitmap(".\\res\\fly\\flyR4.bmp", RGB(255, 0, 0));
          }
          KirbyFlyR.AddBitmap(".\\res\\fly\\flyR4.bmp", RGB(255, 0, 0));
          KirbyFlyR.AddBitmap(".\\res\\fly\\flyR5.bmp", RGB(255, 0, 0));

          // load flyingR
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR6.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR7.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR8.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR9.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR10.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR9.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR8.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR7.bmp", RGB(255, 0, 0));
          KirbyFlyingR.AddBitmap(".\\res\\fly\\flyR6.bmp", RGB(255, 0, 0));
          // load flyL
          KirbyFlyL.AddBitmap(".\\res\\fly\\flyL1.bmp", RGB(255, 0, 0));
          KirbyFlyL.AddBitmap(".\\res\\fly\\flyL2.bmp", RGB(255, 0, 0));
          KirbyFlyL.AddBitmap(".\\res\\fly\\flyL3.bmp", RGB(255, 0, 0));
          count = 3;
          while (count-- > 0) {
                    KirbyFlyL.AddBitmap(".\\res\\fly\\flyL4.bmp", RGB(255, 0, 0));
          }
          KirbyFlyL.AddBitmap(".\\res\\fly\\flyL5.bmp", RGB(255, 0, 0));
          // load flyingL
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL6.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL7.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL8.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL9.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL10.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL9.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL8.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL7.bmp", RGB(255, 0, 0));
          KirbyFlyingL.AddBitmap(".\\res\\fly\\flyL6.bmp", RGB(255, 0, 0));
          // load hurtR
          KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR1.bmp", RGB(255, 255, 255));
          KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR2.bmp", RGB(255, 255, 255));
```

```
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR3.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR5.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR6.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR7.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR8.bmp", RGB(255, 255, 255));
        // load hurtL
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL1.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL2.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL3.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL5.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL6.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL7.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL8.bmp", RGB(255, 255, 255));

        // load Star Throw
        int rgb[3] = { 0, 0, 0 };
        StarThrow.LoadBitmap(IDB_STARTHROW, rgb, 5);
}
void kirby::OnShow()
{
        if (StarThrow.WeaponIsShow()) {
                StarThrow.OnMove();
                StarThrow.OnShow();
        }
        switch (GetCase()) {
       // case jump up right
        case 1:
                KirbyJumpR.SetDelayCount(4);
                now_img_h = KirbyJumpR.Height();
                now_img_w = KirbyJumpR.Width();
                KirbyJumpR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyJumpR.OnMove();
                KirbyJumpR.OnShow();
                break;
       // case down attack right
        case 2:
                KirbyDownAttackR.SetDelayCount(5);
                now_img_h = KirbyDownAttackR.Height();
                now_img_w = KirbyDownAttackR.Width();
                KirbyDownAttackR.SetTopLeft(x, y + KirbyStand.Height() - KirbyDownR.Height());
```

```
            KirbyDownAttackR.OnMove();
            KirbyDownAttackR.OnShow();
            if (KirbyDownAttackR.IsFinalBitmap()) {
                    IsAttack = false;
                    KirbyDownAttackR.Reset();
            }
            break;
// case down right
 case 3:
            now_img_h = KirbyDownR.Height();
            now_img_w = KirbyDownR.Width();
            KirbyDownR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyDownR.ShowBitmap();
            break;
// case scream right
 case 4:
            KirbyScreamR.SetDelayCount(3);
            now_img_h = KirbyScreamR.Height();
            now_img_w = KirbyScreamR.Width();
            KirbyScreamR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyScreamR.OnMove();
            KirbyScreamR.OnShow();
            if (KirbyScreamR.IsFinalBitmap()) {
                    IsAttack = false;
                    KirbyScreamR.Reset();
            }
            break;
 // case walking right
 case 5:
            KirbyMovingR.SetDelayCount(2);
            now_img_h = KirbyMovingR.Height();
            now_img_w = KirbyMovingR.Width();
            KirbyMovingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyMovingR.OnShow();
            break;
 // case standing right
 case 6:
            KirbyStand.SetDelayCount(3);
            now_img_h = KirbyStand.Height();
            now_img_w = KirbyStand.Width();
```

```
        KirbyStand.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyStand.OnShow();
        break;
// case fly up right
case 7:
        KirbyFlyR.SetDelayCount(5);
        now_img_h = KirbyFlyR.Height();
        now_img_w = KirbyFlyR.Width();
        KirbyFlyR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyR.OnMove();
        KirbyFlyR.OnShow();
        if (KirbyFlyR.IsFinalBitmap()) {
                KirbyFlyR.Reset();
        }
        break;
// case flying right
case 8:
        KirbyFlyingR.SetDelayCount(3);
        now_img_h = KirbyFlyingR.Height();
        now_img_w = KirbyFlyingR.Width();
        KirbyFlyingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyingR.OnMove();
        KirbyFlyingR.OnShow();
        if (KirbyFlyingR.IsFinalBitmap()) {
                KirbyFlyingR.Reset();
        }
        break;
// case jump up left
case 9:
        KirbyJumpL.SetDelayCount(4);
        now_img_h = KirbyJumpL.Height();
        now_img_w = KirbyJumpL.Width();
        KirbyJumpL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyJumpL.OnMove();
        KirbyJumpL.OnShow();
        break;
// case down attack left
case 10:
        KirbyDownAttackL.SetDelayCount(5);
        now_img_h = KirbyDownAttackL.Height();
```

```
            now_img_w = KirbyDownAttackL.Width();
            KirbyDownAttackL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyDownAttackL.OnMove();
            KirbyDownAttackL.OnShow();
            if (KirbyDownAttackL.IsFinalBitmap()) {
                    IsAttack = false;
                    KirbyDownAttackL.Reset();
            }
            break;
// case down left
 case 11:
            now_img_h = KirbyDownL.Height();
            now_img_w = KirbyDownL.Width();
            KirbyDownL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyDownL.ShowBitmap();
            break;
 // case scream left
 case 12:
            KirbyScreamL.SetDelayCount(3);
            now_img_h = KirbyScreamL.Height();
            now_img_w = KirbyScreamL.Width();
            KirbyScreamL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyScreamL.OnMove();
            KirbyScreamL.OnShow();
            if (KirbyScreamL.IsFinalBitmap()) {
                    IsAttack = false;
                    KirbyScreamL.Reset();
            }
            break;
 // case walking left
 case 13:
            KirbyMovingL.SetDelayCount(2);
            now_img_h = KirbyMovingL.Height();
            now_img_w = KirbyMovingL.Width();
            KirbyMovingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyMovingL.OnShow();
            break;
 // case standing left
 case 14:
            KirbyStandL.SetDelayCount(3);
```

```
        now_img_h = KirbyStandL.Height();
        now_img_w = KirbyStandL.Width();
        KirbyStandL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyStandL.OnShow();
        break;
// case fly up left
case 15:
        KirbyFlyL.SetDelayCount(5);
        now_img_h = KirbyFlyL.Height();
        now_img_w = KirbyFlyL.Width();
        KirbyFlyL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyL.OnMove();
        KirbyFlyL.OnShow();
        if (KirbyFlyL.IsFinalBitmap()) {
                KirbyFlyL.Reset();
        }
        break;
// case flying left
case 16:
        KirbyFlyingL.SetDelayCount(3);
        now_img_h = KirbyFlyingL.Height();
        now_img_w = KirbyFlyingL.Width();
        KirbyFlyingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyingL.OnMove();
        KirbyFlyingL.OnShow();
        if (KirbyFlyingL.IsFinalBitmap()) {
                KirbyFlyingL.Reset();
        }
        break;
// case hurt right
case 17:
        KirbyHurtR.SetDelayCount(2);
        now_img_h = KirbyHurtR.Height();
        now_img_w = KirbyHurtR.Width();
        KirbyHurtR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyHurtR.OnMove();
        KirbyHurtR.OnShow();
        if (KirbyHurtR.IsFinalBitmap()) {
                IsHurt = false;
                KirbyHurtR.Reset();
```

```
          }
          break;
 // case hurt left
 case 18:
          KirbyHurtL.SetDelayCount(2);
          now_img_h = KirbyHurtL.Height();
          now_img_w = KirbyHurtL.Width();
          KirbyHurtL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
          KirbyHurtL.OnMove();
          KirbyHurtL.OnShow();
          if (KirbyHurtL.IsFinalBitmap()) {
                   IsHurt = false;
                   KirbyHurtL.Reset();
          }
          break;
// case IsEaten standing right
 case 19:
          KirbyFatStand.SetDelayCount(3);
          now_img_h = KirbyFatStand.Height();
          now_img_w = KirbyFatStand.Width();
          KirbyFatStand.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
          KirbyFatStand.OnMove();
          KirbyFatStand.OnShow();
          break;
 // case IsEaten standing left
 case 20:
          KirbyFatStandL.SetDelayCount(3);
          now_img_h = KirbyFatStandL.Height();
          now_img_w = KirbyFatStandL.Width();
          KirbyFatStandL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
          KirbyFatStandL.OnMove();
          KirbyFatStandL.OnShow();
          break;
 // case fat walk right 21
 case 21:
          KirbyFatMovingR.SetDelayCount(2);
          now_img_h = KirbyFatMovingR.Height();
          now_img_w = KirbyFatMovingR.Width();
          KirbyFatMovingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
          KirbyFatMovingR.OnMove();
```

```
                KirbyFatMovingR.OnShow();
                break;
        // case fat walk left 22
        case 22:
                KirbyFatMovingL.SetDelayCount(2);
                now_img_h = KirbyFatMovingL.Height();
                now_img_w = KirbyFatMovingL.Width();
                KirbyFatMovingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyFatMovingL.OnMove();
                KirbyFatMovingL.OnShow();
                break;
        default:
                CMovingBitmap temp;
                temp.LoadBitmap(IDB_KIRBY);
                temp.ShowBitmap();
                break;
        }
}
void kirby::OnMove()
{
        // set moving XY
        const int length = 4;
        // set moving XY and frame of test
        if (IsMovingL && !IsDown && !IsAttack && !IsHurt && x > frame_of_test) {
                if (IsFacingR) {
                        IsFacingR = false;
                }
                if (IsRun && !InAir) {
                        SetXY(x - length * 3, y);
                }
                else if (IsRun && IsHack) {
                        SetXY(x - length * 3, y);
                }
                else {
                        SetXY(x - length, y);
                }
        }
        if (IsMovingR && !IsDown && !IsAttack && !IsHurt && x < SIZE_X - ImgW - frame_of_test) {
                if (!IsFacingR) {
                        IsFacingR = true;
```

```
		}
		if (IsRun && !InAir) {
			SetXY(x + length * 3, y);
		}
		else if (IsRun && IsHack) {
			SetXY(x + length * 3, y);
		}
		else {
			SetXY(x + length, y);
		}
	}
	// set down attack right and left
	if (IsDown && IsAttack) {
		if (IsFacingR && x < SIZE_X - ImgW - frame_of_test) {
			SetXY(x + length * 3, y);
		}
		else if (x > frame_of_test) {
			SetXY(x - length * 3, y);
		}
	}
	// set jump and fly
	if (IsRising && InAir && !IsAttack && !IsDown && !IsHurt) {
		if (y < sky_top) {		// 碰到天花板
			IsRising = false;
		}
		if (IsJumping) {
			if (velocity > 0) {
				SetXY(x, y - velocity);
				velocity--;
			}
			else {
				IsRising = false;
				velocity = 1;
			}
		}
		else if (FlyUp && IsFat) {
			if (fly_velocity > 0) {
				SetXY(x, y - fly_velocity);
				fly_velocity--;
			}
```

```
                else {
                        IsRising = false;
                        FlyUp = false;
                        fly_velocity = init_fly_velocity;
                }
        }
}
else {
        // falling
        if (y < floor - frame_of_test - ImgH) {
                if (IsJumping || !IsFlying) {
                        SetXY(x, y + velocity);
                        velocity++;
                }
                else if (IsFlying) {
                        SetXY(x, y + 1);
                }
                if (YouAreGround) {
                        velocity = init_velocity;
                        InAir = false;
                        IsRising = true;
                        IsJumping = false;
                        KirbyJumpR.Reset();
                        KirbyJumpL.Reset();
                        KirbyFlyR.Reset();
                        KirbyFlyL.Reset();
                }
        }
        else {
                y = floor - frame_of_test - ImgH;
                velocity = init_velocity;
                InAir = false;
                IsRising = true;
                IsJumping = false;
                KirbyJumpR.Reset();
                KirbyJumpL.Reset();
                KirbyFlyR.Reset();
                KirbyFlyL.Reset();
        }
}
```

```
		// kirby is hurt
		if (IsHurt) {
			if (!OtherFromL && x - 4 > 0) {
				SetXY(x - 4, y);
			}
			else if (x + 4 < SIZE_X - now_img_w) {
				SetXY(x + 4, y);
			}
		}
		// kirby throw star
		if (IsEaten && IsAttack) {
			ThrowStar();
		}
		StarThrow.OnMove();
		if (StarThrow.GetAttackTime() > 0 && game_state_counter - StarThrow.GetAttackTime() < 50) {
			int* temp = StarThrow.GetXy();
			if (StarThrow.GetAttackFacingR()) {
				StarThrow.SetXy(temp[0] + 20, temp[1]);
			}
			else {
				StarThrow.SetXy(temp[0] - 20, temp[1]);
			}
			delete[] temp;
		}
		else {
			StarThrow.SetShow(false);
		}
		// animation OnMove
		KirbyMovingL.OnMove();
		KirbyMovingR.OnMove();
		KirbyStand.OnMove();
		KirbyStandL.OnMove();
	}
	void kirby::SetXY(int x_in, int y_in) {
		int aXy[4] = { x_in, y_in, x_in + now_img_w, y_in + now_img_h };
		bool result = true;
		if (!StarBlockList.empty()) {
		for (int k = 0; k < int(StarBlockList.size()); k++) {
			if (StarBlockList[k] != nullptr && StarBlockList[k]->GetShow()) {
				int* bXy = StarBlockList[k]->GetXy();
```

```
                        int i = 0, n = 1;
                        for (int count = 0; count < 2; count++) {
                         for (int _count = 0; _count < 2; _count++) {
                                 if (aXy[i] > bXy[0] && aXy[i] < bXy[2] && aXy[n] > bXy[1] && aXy[n]
< bXy[3]) {
                                          result = false;
                                          if (y_in - y > 0) {
                                          // go down
                                                   YouAreGround = true;
                                          }
                                 }
                                 n += 2;
                           }
                         n = 1;
                         i += 2;
                }
                i = 0, n = 1;
                for (int count = 0; count < 2; count++) {
                         for (int _count = 0; _count < 2; _count++) {
                                 if (bXy[i] > aXy[0] && bXy[i] < aXy[2] && bXy[n] > aXy[1] && bXy[n]
< aXy[3]) {
                                          result = false;
                                          if (y_in - y > 0) {
                                                   // go down
                                                   YouAreGround = true;
                                          }
                                  }
                                  n += 2;
                                 }
                                 n = 1;
                                 i += 2;
                        }
                         delete[] bXy;
                         }
                  }
         }
         if (result) {
                  if (x_in - x > 0) {
                           // right
                           if (x_in > BoundaryLeft - now_img_w && Map->Left() > -(BoundaryRight -
SIZE_X / 2)) {
                                  Map->SetTopLeft(Map->Left() - (x_in - x), Map->Top());
                                  Door->SetTopLeft(Door->Left() - (x_in - x), Door->Top());
                                  int* temp = StarThrow.GetXy();
                                  StarThrow.SetXy(temp[0] - (x_in - x), temp[1]);
                                  delete[] temp;
                                  if (!StarBlockList.empty()) {
                                          for (auto n : StarBlockList) {
                                                   n->SetXY(n->Left() - (x_in - x), n->Top());
                                          }
                                  }
                                  if (!EnemyList.empty()) {
                                          for (auto n : EnemyList) {
                                                   n->SetXy(n->Left() - (x_in - x), n->Top());
                                          }
                                  }
                                  return;
                          }
                  }
                  else if (x_in - x < 0) {
                          // left
                          if (x_in < BoundaryLeft && Map->Left() < 0) {
                                  Map->SetTopLeft(Map->Left() - (x_in - x), Map->Top());
                                  Door->SetTopLeft(Door->Left() - (x_in - x), Door->Top());
                                  int* temp = StarThrow.GetXy();
                                  StarThrow.SetXy(temp[0] - (x_in - x), temp[1]);
                                  delete[] temp;
                                  if (!StarBlockList.empty()) {
                                          for (auto n : StarBlockList) {
                                                   n->SetXY(n->Left() - (x_in - x), n->Top());
                                          }
                                  }
                                  if (!EnemyList.empty()) {
                                          for (auto n : EnemyList) {
                                                   n->SetXy(n->Left() - (x_in - x), n->Top());
                                          }
                                  }
                                  return;
                          }
                  }
```

```
if (y_in - y > 0) {
        // go down
        if (y_in > BoundaryTop && Map->Top() > -260) {
                Map->SetTopLeft(Map->Left(), Map->Top() - (y_in - y));
                Door->SetTopLeft(Door->Left(), Door->Top() - (y_in - y));
                int* temp = StarThrow.GetXy();
                StarThrow.SetXy(temp[0], temp[1] - (y_in - y));
                delete[] temp;
                if (!StarBlockList.empty()) {
                        for (auto n : StarBlockList) {
                                n->SetXY(n->Left(), n->Top() - (y_in - y));
                        }
                }
                if (!EnemyList.empty()) {
                        for (auto n : EnemyList) {
                                n->SetXy(n->Left(), n->Top() - (y_in - y));
                        }
                }
                return;
        }
}
else if (y_in - y < 0) {
        // go up
        if (y_in < BoundaryTop && Map->Top() < 0) {
                Map->SetTopLeft(Map->Left(), Map->Top() - (y_in - y));
                Door->SetTopLeft(Door->Left(), Door->Top() - (y_in - y));
                int* temp = StarThrow.GetXy();
                StarThrow.SetXy(temp[0], temp[1] - (y_in - y));
                delete[] temp;
                if (!StarBlockList.empty()) {
                        for (auto n : StarBlockList) {
                                n->SetXY(n->Left(), n->Top() - (y_in - y));
                        }
                }
                if (!EnemyList.empty()) {
                        for (auto n : EnemyList) {
                                n->SetXy(n->Left(), n->Top() - (y_in - y));
                        }
                }
                return;
```

```
                        }
                }
                x = x_in;
                y = y_in;
        }
}
void kirby::SetKindInit() {
        kirby_kind = "normal_kirby";
}
void  kirby::SetMovingL(bool input) {
        IsMovingL = input;
}
void kirby::SetMovingR(bool input) {
        IsMovingR = input;
}
void kirby::SetFacingR(bool input) {
        IsFacingR = input;
}
void kirby::SetFacingL(bool input) {
        IsFacingR = !input;
}
void kirby::SetDown(bool input) {
        if (!IsAttack && !InAir && !IsHurt) {
                IsDown = input;
        }
}
void kirby::SetAttack(bool input) {
        if (!IsHurt) {
                IsAttack = input;
        }
        if (IsFlying) {
                SetFly(false);
        }
}
void kirby::SetJump(bool input) {
        YouAreGround = false;
        if (IsJumping && !IsHurt) {
                SetFly(true);
        }
        if (!IsFlying && !IsHurt) {
```

```
			InAir = input;
			IsJumping = input;
		}
		else {
			SetFly(true);
		}
	}
	void kirby::SetFly(bool input) {
		YouAreGround = false;
		if (!IsDown || !IsHurt) {
			InAir = input;
			IsFat = input;
			FlyUp = input;
			IsRising = input;
			IsFlying = input;
		}
		if (IsJumping) {
			IsJumping = false;
		}
	}
	void kirby::SetEaten(bool input) {
		IsEaten = input;
	}
	void kirby::SetEaten(bool input, std::string name) {
		IsEaten = input;
		EatenEnemy = name;
	}
	void kirby::SetKind(std::string input) {
		kirby_kind = input;
	}
	std::string kirby::GetEatenEnemy() {
		return EatenEnemy;
	}
	void kirby::YouAreLeft(bool YouAreLeft) {
		OtherFromL = !YouAreLeft;
	}
	void kirby::SetCounter(int input_counter) {
		game_state_counter = input_counter;
	}
	void kirby::SetMap(CMovingBitmap* input_Map) {
```

```
	Map = input_Map;
}
void kirby::SetDoor(CMovingBitmap* input_door) {
	Door = input_door;
}
void kirby::SetThings(vector<thing*> input) {
	StarBlockList = input;
}
void kirby::SetEnemies(vector<enemy*> input_EnemyList) {
	EnemyList = input_EnemyList;
}
void kirby::SetRun(bool input) {
	IsRun = input;
}
void kirby::SetHack(bool input) {
	IsHack = input;
}
bool kirby::Hurt(int input, int time) {
	if (IsHack) {
		return false;
	}
	if (abs(LastHurt - time) < 30) {
		return false;
	}
	if (IsDown && IsAttack) {
		return false;
	}
	kirby_kind = "normal_kirby";
	LastHurt = time;
	hp -= input;
	IsHurt = true;
	SetDown(false);
	SetAttack(false);
	SetJump(false);
	SetFly(false);
	return true;
}
void kirby::ThrowStar() {
	int* temp = GetXy();
	StarThrow.SetAttackState(game_state_counter, IsFacingR, temp);
```

```
        delete[] temp;
        StarThrow.SetShow(true);
        IsEaten = false;
        IsAttack = false;
        EatenEnemy = "";
}
int kirby::GetCase() {
        if (IsHurt) {
                if (OtherFromL) {
                        if (IsFacingR) {
                                IsFacingR = false;
                        }
                        return 18;
                }
                else {
                        if (!IsFacingR) {
                                IsFacingR = true;
                        }
                        return 17;
                }
        }
        else {
                if (IsFacingR) {
                        if (IsJumping && !IsFat && !IsAttack) {
                                // case jump up right
                                return 1;
                        }
                        else if (IsDown && !IsJumping && !IsFlying && !IsFat) {
                                if (IsAttack) {
                                        // case down attack right
                                        return 2;
                                }
                                else {
                                        // case down right
                                        return 3;
                                }
                        }
                        else if (IsAttack && !IsJumping) {
                                // case scream right
                                return 4;
```

```
		}
		else if (FlyUp && !IsFat && !IsAttack && !IsDown) {
			// case fly up right
			return 7;
		}
		else if (IsFat && IsFlying && !IsAttack) {
			// case flying right
			return 8;
		}
		else if (IsMovingR && !IsEaten) {
			// case walking right
			return 5;
		}
		else if (IsEaten && IsMovingR) {
			// case IsEaten  walking right
			return 21;
		}
		else if (IsEaten) {
			// case IsEaten standing right
			return 19;
		}
		else {
			// case standing right
			return 6;
		}
	}
	else {
		// facing left
		if (IsJumping && !IsFat && !IsAttack) {
			// case jump up left
			return 9;
		}
		else if (IsDown && !IsJumping && !IsFlying) {
			if (IsAttack) {
				// case down attack left
				return 10;
			}
			else {
				// case down left
				return 11;
```

```
                }
            }
            else if (IsAttack && !IsJumping) {
                // case scream left
                return 12;
            }
            else if (FlyUp && !IsFat && !IsAttack && !IsDown) {
                // case fly up left
                return 15;
            }
            else if (IsFat && IsFlying && !IsAttack) {
                // case flying left
                return 16;
            }
            else if (IsMovingL && !IsEaten) {
                // case walking left
                return 13;
            }
            else if (IsEaten && IsMovingL) {
                // case IsEaten walking left
                return 22;
            }
            else if (IsEaten) {
                // case IsEaten standing left
                return 20;
            }
            else {
                // case standing left
                return 14;
            }
        }
    }
}
int kirby::GetHp() {
    return hp;
}
int* kirby::GetXy() {
    return new int[4]{ x, y , x + ImgW, y + ImgH };
}
int kirby::GetWidth() {
	return ImgW;
}
int kirby::GetHeight() {
	return ImgH;
}
std::string kirby::GetKind() {
	return kirby_kind;
}
bool kirby::GetEaten() {
	return IsEaten;
}
bool kirby::GetIsGround() {
	return YouAreGround;
}
bool kirby::IsAlive() {
	if (hp > 0) {
		return true;
	}
	else {
		return false;
	}
}
bool kirby::IsScreamR() {
	return IsAttack && IsFacingR && !IsDown;
}
bool kirby::IsScreamL() {
	return IsAttack && !IsFacingR && !IsDown;
}
bool kirby::GetFacingR() {
	return IsFacingR;
}
weapon* kirby::GetWeapon() {
	return &StarThrow;
}
void kirby::KirbyCopy(kirby* copy_kirby) {
	copy_kirby->EatenEnemy = "";
	copy_kirby->x = x;
	copy_kirby->y = y;
	copy_kirby->hp = hp;
	copy_kirby->velocity = velocity;
```

```
		copy_kirby->fly_velocity = fly_velocity;
		copy_kirby->game_state_counter = game_state_counter;
		copy_kirby->LastHurt = LastHurt;

		copy_kirby->IsRising = IsRising;
		copy_kirby->IsMovingL = IsMovingL;
		copy_kirby->IsMovingR = IsMovingR;
		copy_kirby->IsFacingR = IsFacingR;
		copy_kirby->InAir = InAir;
		copy_kirby->OtherFromL = OtherFromL;
		copy_kirby->IsDown = IsDown;
		copy_kirby->IsAttack = false;
		copy_kirby->IsJumping = IsJumping;
		copy_kirby->IsFlying = IsFlying;
		copy_kirby->FlyUp = FlyUp;
		copy_kirby->IsHurt = IsHurt;
		copy_kirby->YouAreGround = YouAreGround;
		copy_kirby->IsRun = IsRun;
		copy_kirby->IsHack = IsHack;
	}
	void sparky_kirby::SetEaten(bool input) {}
	void sparky_kirby::SetEaten(bool input, std::string name) {}
	void sparky_kirby::ThrowStar() {}
	void sparky_kirby::LoadBitmap()
	{
		// load walk left
		char *walk_left[10] = { ".\\res\\kirby_sparky\\walk\\walkL1.bmp",
".\\res\\kirby_sparky\\walk\\walkL2.bmp", ".\\res\\kirby_sparky\\walk\\walkL3.bmp",
".\\res\\kirby_sparky\\walk\\walkL4.bmp", ".\\res\\kirby_sparky\\walk\\walkL5.bmp",
".\\res\\kirby_sparky\\walk\\walkL6.bmp", ".\\res\\kirby_sparky\\walk\\walkL7.bmp",
".\\res\\kirby_sparky\\walk\\walkL8.bmp", ".\\res\\kirby_sparky\\walk\\walkL9.bmp",
".\\res\\kirby_sparky\\walk\\walkL10.bmp" };
		for (int i = 0; i < 10; i++)
		{
			KirbyMovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
		}
		// load walk right
		char *walk_right[10] = { ".\\res\\kirby_sparky\\walk\\walkR1.bmp",
".\\res\\kirby_sparky\\walk\\walkR2.bmp", ".\\res\\kirby_sparky\\walk\\walkR3.bmp",
".\\res\\kirby_sparky\\walk\\walkR4.bmp", ".\\res\\kirby_sparky\\walk\\walkR5.bmp",
".\\res\\kirby_sparky\\walk\\walkR6.bmp", ".\\res\\kirby_sparky\\walk\\walkR7.bmp",
".\\res\\kirby_sparky\\walk\\walkR8.bmp", ".\\res\\kirby_sparky\\walk\\walkR9.bmp",
".\\res\\kirby_sparky\\walk\\walkR10.bmp" };
        for (int i = 0; i < 10; i++)
        {
                KirbyMovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
        }
        // load stand and wink right
        int count = 12;
        while (count-- > 0) {
                KirbyStand.AddBitmap(".\\res\\kirby_sparky\\basic\\basicR1.bmp", RGB(255, 255,
255));
        }
        KirbyStand.AddBitmap(".\\res\\kirby_sparky\\basic\\basicR3.bmp", RGB(255, 255, 255));
        count = 12;
        while (count-- > 0) {
                KirbyStand.AddBitmap(".\\res\\kirby_sparky\\basic\\basicR1.bmp", RGB(255, 255,
255));
        }
        KirbyStand.AddBitmap(".\\res\\kirby_sparky\\basic\\basicR3.bmp", RGB(255, 255, 255));
        KirbyStand.AddBitmap(".\\res\\kirby_sparky\\basic\\basicR1.bmp", RGB(255, 255, 255));
        KirbyStand.AddBitmap(".\\res\\kirby_sparky\\basic\\basicR3.bmp", RGB(255, 255, 255));
        // load stand and wink left
        count = 12;
        while (count-- > 0) {
                KirbyStandL.AddBitmap(".\\res\\kirby_sparky\\basic\\basicL1.bmp", RGB(255, 255,
255));
        }
        KirbyStandL.AddBitmap(".\\res\\kirby_sparky\\basic\\basicL3.bmp", RGB(255, 255, 255));
        count = 12;
        while (count-- > 0) {
                KirbyStandL.AddBitmap(".\\res\\kirby_sparky\\basic\\basicL1.bmp", RGB(255, 255,
255));
        }
        KirbyStandL.AddBitmap(".\\res\\kirby_sparky\\basic\\basicL3.bmp", RGB(255, 255, 255));
        KirbyStandL.AddBitmap(".\\res\\kirby_sparky\\basic\\basicL1.bmp", RGB(255, 255, 255));
        KirbyStandL.AddBitmap(".\\res\\kirby_sparky\\basic\\basicL3.bmp", RGB(255, 255, 255));
```

```
	// load down right and left
	KirbyDownR.LoadBitmap(".\\res\\kirby_sparky\\down\\downR1.bmp", RGB(255, 255, 255));
	KirbyDownL.LoadBitmap(".\\res\\kirby_sparky\\down\\downL1.bmp", RGB(255, 255, 255));
	// load down attack right and left
	count = 3;
	while (count-- > 0) {
		KirbyDownAttackR.AddBitmap(".\\res\\kirby_sparky\\down\\downR2.bmp", RGB(255, 255,
255));
		KirbyDownAttackL.AddBitmap(".\\res\\kirby_sparky\\down\\downL2.bmp", RGB(255, 255,
255));
	}
	// load jump right
	char *jump_right[10] = { ".\\res\\kirby_sparky\\jump\\jumpR1.bmp",
".\\res\\kirby_sparky\\jump\\jumpR2.bmp", ".\\res\\kirby_sparky\\jump\\jumpR3.bmp",
".\\res\\kirby_sparky\\jump\\jumpR4.bmp", ".\\res\\kirby_sparky\\jump\\jumpR5.bmp",
".\\res\\kirby_sparky\\jump\\jumpR6.bmp", ".\\res\\kirby_sparky\\jump\\jumpR7.bmp",
".\\res\\kirby_sparky\\jump\\jumpR8.bmp", ".\\res\\kirby_sparky\\jump\\jumpR9.bmp",
".\\res\\kirby_sparky\\jump\\jumpR10.bmp" };
	for (int i = 0; i < 10; i++) {
		KirbyJumpR.AddBitmap(jump_right[i], RGB(255, 255, 255));
	}
	// load jump left
	char *jump_left[10] = { ".\\res\\kirby_sparky\\jump\\jumpL1.bmp",
".\\res\\kirby_sparky\\jump\\jumpL2.bmp", ".\\res\\kirby_sparky\\jump\\jumpL3.bmp",
".\\res\\kirby_sparky\\jump\\jumpL4.bmp", ".\\res\\kirby_sparky\\jump\\jumpL5.bmp",
".\\res\\kirby_sparky\\jump\\jumpL6.bmp", ".\\res\\kirby_sparky\\jump\\jumpL7.bmp",
".\\res\\kirby_sparky\\jump\\jumpL8.bmp", ".\\res\\kirby_sparky\\jump\\jumpL9.bmp",
".\\res\\kirby_sparky\\jump\\jumpL10.bmp" };
	for (int i = 0; i < 10; i++)
	{
		KirbyJumpL.AddBitmap(jump_left[i], RGB(255, 255, 255));
	}
	// load scream right and left
	count = 3;
	while (count-- > 0) {
		KirbyScreamR.AddBitmap(".\\res\\kirby_sparky\\scream\\screamR2.bmp", RGB(255, 255,
255));
	}
	count = 10;
	while (count-- > 0) {
			KirbyScreamR.AddBitmap(".\\res\\kirby_sparky\\scream\\screamR1.bmp", RGB(255, 255,
255));
		}
		count = 3;
		while (count-- > 0) {
			KirbyScreamL.AddBitmap(".\\res\\kirby_sparky\\scream\\screamL2.bmp", RGB(255, 255,
255));
		}
		count = 10;
		while (count-- > 0) {
			KirbyScreamL.AddBitmap(".\\res\\kirby_sparky\\scream\\screamL1.bmp", RGB(255, 255,
255));
		}
		// load flyR
		KirbyFlyR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR1.bmp", RGB(255, 255, 255));
		KirbyFlyR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR2.bmp", RGB(255, 255, 255));
		KirbyFlyR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR3.bmp", RGB(255, 255, 255));
		count = 3;
		while (count-- > 0) {
			KirbyFlyR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR4.bmp", RGB(255, 255, 255));
		}
		KirbyFlyR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR4.bmp", RGB(255, 255, 255));
		KirbyFlyR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR5.bmp", RGB(255, 255, 255));
		// load flyingR
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR6.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR7.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR8.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR9.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR10.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR9.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR8.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR7.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_sparky\\fly\\flyR6.bmp", RGB(255, 255, 255));
		// load flyL
		KirbyFlyL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL1.bmp", RGB(255, 255, 255));
		KirbyFlyL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL2.bmp", RGB(255, 255, 255));
		KirbyFlyL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL3.bmp", RGB(255, 255, 255));
		count = 3;
		while (count-- > 0) {
			KirbyFlyL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL4.bmp", RGB(255, 255, 255));
```

```
        }
        KirbyFlyL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL5.bmp", RGB(255, 255, 255));
        // load flyingL
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL6.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL7.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL8.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL9.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL10.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL9.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL8.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL7.bmp", RGB(255, 255, 255));
        KirbyFlyingL.AddBitmap(".\\res\\kirby_sparky\\fly\\flyL6.bmp", RGB(255, 255, 255));
        // load hurtR
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR1.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR2.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR3.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR5.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR6.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR7.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR8.bmp", RGB(255, 255, 255));
        // load hurtL
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL1.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL2.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL3.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL5.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL6.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL7.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL8.bmp", RGB(255, 255, 255));
        // load Star Throw
        int rgb[3] = { 0, 0, 255 };
        StarThrow.LoadBitmap(".\\res\\weapon\\attack1.bmp", rgb, 2);
        StarThrow.LoadBitmap(".\\res\\weapon\\attack2.bmp", rgb, 2);
}
void sparky_kirby::OnMove()
{
        // set moving XY
        const int length = 4;
        // set moving XY and frame of test
        if (IsMovingL && !IsDown && !IsAttack && !IsHurt && x > frame_of_test) {
                if (IsFacingR) {
```

```
                IsFacingR = false;
        }
        if (IsRun && !InAir) {
                SetXY(x - length * 3, y);
        }
        else if (IsRun && IsHack) {
                SetXY(x - length * 3, y);
        }
        else {
                SetXY(x - length, y);
        }
}
if (IsMovingR && !IsDown && !IsAttack && !IsHurt && x < SIZE_X - ImgW - frame_of_test) {
        if (!IsFacingR) {
                IsFacingR = true;
        }
        if (IsRun && !InAir) {
                SetXY(x + length * 3, y);
        }
        else if (IsRun && IsHack) {
                SetXY(x + length * 3, y);
        }
        else {
                SetXY(x + length, y);
        }
}
// set down attack right and left
if (IsDown && IsAttack) {
        if (IsFacingR && x < SIZE_X - ImgW - frame_of_test) {
                SetXY(x + length * 3, y);
        }
        else if (x > frame_of_test) {
                SetXY(x - length * 3, y);
        }
}
// set jump and fly
if (IsRising && InAir && !IsAttack && !IsDown && !IsHurt) {
        if (y < sky_top) {          // 碰到天花板
                IsRising = false;
        }
        if (IsJumping) {
                if (velocity > 0) {
                        SetXY(x, y - velocity);
                        velocity--;
                }
                else {
                        IsRising = false;
                        velocity = 1;
                }
        }
        else if (FlyUp && IsFat) {
                if (fly_velocity > 0) {
                        SetXY(x, y - fly_velocity);
                        fly_velocity--;
                }
                else {
                        IsRising = false;
                        FlyUp = false;
                        fly_velocity = init_fly_velocity;
                }
        }
}
else {
        // falling
        if (y < floor - frame_of_test - ImgH) {
                if (IsJumping || !IsFlying) {
                        SetXY(x, y + velocity);
                        velocity++;
                }
                else if (IsFlying) {
                        SetXY(x, y + 1);
                }
                if (YouAreGround) {
                        velocity = init_velocity;
                        InAir = false;
                        IsRising = true;
                        IsJumping = false;
                        KirbyJumpR.Reset();
                        KirbyJumpL.Reset();
                        KirbyFlyR.Reset();
                        KirbyFlyL.Reset();
                    }
                }
                else {
                    y = floor - frame_of_test - ImgH;
                    velocity = init_velocity;
                    InAir = false;
                    IsRising = true;
                    IsJumping = false;
                    KirbyJumpR.Reset();
                    KirbyJumpL.Reset();
                    KirbyFlyR.Reset();
                    KirbyFlyL.Reset();
                }
        }
        // kirby is hurt
        if (IsHurt) {
                if (!OtherFromL && x - 4 > 0) {
                    SetXY(x - 4, y);
                }
                else if (x + 4 < SIZE_X - now_img_w) {
                    SetXY(x + 4, y);
                }
        }
        if (IsAttack && !IsDown && &InAir) {
                StarThrow.SetShow(true);
                StarThrow.SetXy(x - 26, y - 10);
        }
        else {
                StarThrow.SetShow(false);
        }

        // animation OnMove
        KirbyMovingL.OnMove();
        KirbyMovingR.OnMove();
        KirbyStand.OnMove();
        KirbyStandL.OnMove();
}
void sparky_kirby::SetKindInit() {
        kirby_kind = "sparky_kirby";
```

```
    }
    void hotHead_kirby::SetEaten(bool input) {}
    void hotHead_kirby::SetEaten(bool input, std::string name) {}
    void hotHead_kirby::ThrowStar(){}
    void hotHead_kirby::LoadBitmap()
    {
        // load walk left
        char *walk_left[10] = { ".\\res\\kirby_hotHead\\walk\\walkL1.bmp",
".\\res\\kirby_hotHead\\walk\\walkL2.bmp", ".\\res\\kirby_hotHead\\walk\\walkL3.bmp",
".\\res\\kirby_hotHead\\walk\\walkL4.bmp", ".\\res\\kirby_hotHead\\walk\\walkL5.bmp",
".\\res\\kirby_hotHead\\walk\\walkL6.bmp", ".\\res\\kirby_hotHead\\walk\\walkL7.bmp",
".\\res\\kirby_hotHead\\walk\\walkL8.bmp", ".\\res\\kirby_hotHead\\walk\\walkL9.bmp",
".\\res\\kirby_hotHead\\walk\\walkL10.bmp" };
        for (int i = 0; i < 10; i++)
        {
            KirbyMovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
        }
        // load walk right
        char *walk_right[10] = { ".\\res\\kirby_hotHead\\walk\\walkR1.bmp",
".\\res\\kirby_hotHead\\walk\\walkR2.bmp", ".\\res\\kirby_hotHead\\walk\\walkR3.bmp",
".\\res\\kirby_hotHead\\walk\\walkR4.bmp", ".\\res\\kirby_hotHead\\walk\\walkR5.bmp",
".\\res\\kirby_hotHead\\walk\\walkR6.bmp", ".\\res\\kirby_hotHead\\walk\\walkR7.bmp",
".\\res\\kirby_hotHead\\walk\\walkR8.bmp", ".\\res\\kirby_hotHead\\walk\\walkR9.bmp",
".\\res\\kirby_hotHead\\walk\\walkR10.bmp" };
        for (int i = 0; i < 10; i++)
        {
            KirbyMovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
        }
        // load stand and wink right
        int count = 12;
        while (count-- > 0) {
            KirbyStand.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicR1.bmp", RGB(255, 255,
255));
        }
        KirbyStand.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicR3.bmp", RGB(255, 255, 255));
        count = 12;
        while (count-- > 0) {
            KirbyStand.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicR1.bmp", RGB(255, 255,
255));
        }
```

```
		KirbyStand.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicR3.bmp", RGB(255, 255, 255));
		KirbyStand.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicR1.bmp", RGB(255, 255, 255));
		KirbyStand.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicR3.bmp", RGB(255, 255, 255));
		// load stand and wink left
		count = 12;
		while (count-- > 0) {
			KirbyStandL.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicL1.bmp", RGB(255, 255,
255));
		}
		KirbyStandL.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicL3.bmp", RGB(255, 255, 255));
		count = 12;
		while (count-- > 0) {
			KirbyStandL.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicL1.bmp", RGB(255, 255,
255));
		}
		KirbyStandL.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicL3.bmp", RGB(255, 255, 255));
		KirbyStandL.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicL1.bmp", RGB(255, 255, 255));
		KirbyStandL.AddBitmap(".\\res\\kirby_hotHead\\basic\\basicL3.bmp", RGB(255, 255, 255));
		// load down right and left
		KirbyDownR.LoadBitmap(".\\res\\kirby_hotHead\\down\\downR1.bmp", RGB(255, 255, 255));
		KirbyDownL.LoadBitmap(".\\res\\kirby_hotHead\\down\\downL1.bmp", RGB(255, 255, 255));
		// load down attack right and left
		count = 3;
		while (count-- > 0) {
			KirbyDownAttackR.AddBitmap(".\\res\\kirby_hotHead\\down\\downR3.bmp", RGB(255, 255,
255));
			KirbyDownAttackL.AddBitmap(".\\res\\kirby_hotHead\\down\\downL3.bmp", RGB(255, 255,
255));
		}
		// load jump right
		char *jump_right[10] = { ".\\res\\kirby_hotHead\\jump\\jumpR1.bmp",
".\\res\\kirby_hotHead\\jump\\jumpR2.bmp", ".\\res\\kirby_hotHead\\jump\\jumpR3.bmp",
".\\res\\kirby_hotHead\\jump\\jumpR4.bmp", ".\\res\\kirby_hotHead\\jump\\jumpR5.bmp",
".\\res\\kirby_hotHead\\jump\\jumpR6.bmp", ".\\res\\kirby_hotHead\\jump\\jumpR7.bmp",
".\\res\\kirby_hotHead\\jump\\jumpR8.bmp", ".\\res\\kirby_hotHead\\jump\\jumpR9.bmp",
".\\res\\kirby_hotHead\\jump\\jumpR10.bmp" };
		for (int i = 0; i < 10; i++) {
			KirbyJumpR.AddBitmap(jump_right[i], RGB(255, 255, 255));
		}
		// load jump left
```

```
	char *jump_left[10] = { ".\\res\\kirby_hotHead\\jump\\jumpL1.bmp",
".\\res\\kirby_hotHead\\jump\\jumpL2.bmp", ".\\res\\kirby_hotHead\\jump\\jumpL3.bmp",
".\\res\\kirby_hotHead\\jump\\jumpL4.bmp", ".\\res\\kirby_hotHead\\jump\\jumpL5.bmp",
".\\res\\kirby_hotHead\\jump\\jumpL6.bmp", ".\\res\\kirby_hotHead\\jump\\jumpL7.bmp",
".\\res\\kirby_hotHead\\jump\\jumpL8.bmp", ".\\res\\kirby_hotHead\\jump\\jumpL9.bmp",
".\\res\\kirby_hotHead\\jump\\jumpL10.bmp" };
	for (int i = 0; i < 10; i++) {
		KirbyJumpL.AddBitmap(jump_left[i], RGB(255, 255, 255));
	}
	// load scream right and left
	count = 3;
	while(count-- > 0) {
		KirbyScreamR.AddBitmap(".\\res\\kirby_hotHead\\scream\\screamR3.bmp", RGB(255, 255,
255));
	}
	count = 10;
	while (count-- > 0) {
		KirbyScreamR.AddBitmap(".\\res\\kirby_hotHead\\scream\\screamR2.bmp", RGB(255, 255,
255));
	}
	count = 5;
	while (count-- > 0) {
		KirbyScreamR.AddBitmap(".\\res\\kirby_hotHead\\scream\\screamR1.bmp", RGB(255, 255,
255));
	}
	count = 3;
	while (count-- > 0) {
		KirbyScreamL.AddBitmap(".\\res\\kirby_hotHead\\scream\\screamL3.bmp", RGB(255, 255,
255));
	}
	count = 10;
	while (count-- > 0) {
		KirbyScreamL.AddBitmap(".\\res\\kirby_hotHead\\scream\\screamL2.bmp", RGB(255, 255,
255));
	}
	count = 5;
	while (count-- > 0) {
		KirbyScreamL.AddBitmap(".\\res\\kirby_hotHead\\scream\\screamL1.bmp", RGB(255, 255,
255));
	}
```

```
// load flyR
KirbyFlyR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR1.bmp", RGB(255, 255, 255));
KirbyFlyR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR2.bmp", RGB(255, 255, 255));
KirbyFlyR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR3.bmp", RGB(255, 255, 255));
count = 3;
while (count-- > 0) {
        KirbyFlyR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR4.bmp", RGB(255, 255, 255));
}
KirbyFlyR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR4.bmp", RGB(255, 255, 255));
KirbyFlyR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR5.bmp", RGB(255, 255, 255));
// load flyingR
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR6.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR7.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR8.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR9.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR10.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR9.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR8.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR7.bmp", RGB(255, 255, 255));
KirbyFlyingR.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyR6.bmp", RGB(255, 255, 255));
// load flyL
KirbyFlyL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL1.bmp", RGB(255, 255, 255));
KirbyFlyL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL2.bmp", RGB(255, 255, 255));
KirbyFlyL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL3.bmp", RGB(255, 255, 255));
count = 3;
while (count-- > 0) {
        KirbyFlyL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL4.bmp", RGB(255, 255, 255));
}
KirbyFlyL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL5.bmp", RGB(255, 255, 255));
// load flyingL
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL6.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL7.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL8.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL9.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL10.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL9.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL8.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL7.bmp", RGB(255, 255, 255));
KirbyFlyingL.AddBitmap(".\\res\\kirby_hotHead\\fly\\flyL6.bmp", RGB(255, 255, 255));
// load hurtR
```

```
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR1.bmp", RGB(255, 255, 255));
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR2.bmp", RGB(255, 255, 255));
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR3.bmp", RGB(255, 255, 255));
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR5.bmp", RGB(255, 255, 255));
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR6.bmp", RGB(255, 255, 255));
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR7.bmp", RGB(255, 255, 255));
	KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR8.bmp", RGB(255, 255, 255));
	// load hurtL
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL1.bmp", RGB(255, 255, 255));
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL2.bmp", RGB(255, 255, 255));
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL3.bmp", RGB(255, 255, 255));
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL5.bmp", RGB(255, 255, 255));
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL6.bmp", RGB(255, 255, 255));
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL7.bmp", RGB(255, 255, 255));
	KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL8.bmp", RGB(255, 255, 255));
	// load Star Throw
	int rgb[3] = {255, 255, 255};
	StarThrow.LoadBitmap(".\\res\\weapon\\hotHead\\attackR1.bmp", rgb, 2);
	StarThrow.LoadBitmap(".\\res\\weapon\\hotHead\\attackR2.bmp", rgb, 2);
	StarThrow.LoadBitmap(".\\res\\weapon\\hotHead\\attackR3.bmp", rgb, 2);
	StarThrow.LoadBitmap(".\\res\\weapon\\hotHead\\attackR4.bmp", rgb, 2);
	left_side.LoadBitmap(".\\res\\weapon\\hotHead\\attackL1.bmp", rgb, 2);
	left_side.LoadBitmap(".\\res\\weapon\\hotHead\\attackL2.bmp", rgb, 2);
	left_side.LoadBitmap(".\\res\\weapon\\hotHead\\attackL3.bmp", rgb, 2);
	left_side.LoadBitmap(".\\res\\weapon\\hotHead\\attackL4.bmp", rgb, 2);
}
void hotHead_kirby::OnMove()
{
	// set moving XY
	const int length = 4;
	// set moving XY and frame of test
	if (IsMovingL && !IsDown && !IsAttack && !IsHurt && x > frame_of_test) {
		if (IsFacingR) {
			IsFacingR = false;
		}
		if (IsRun && !InAir) {
			SetXY(x - length * 3, y);
		}
		else if (IsRun && IsHack) {
			SetXY(x - length * 3, y);
		}
		else {
			SetXY(x - length, y);
		}
	}
	if (IsMovingR && !IsDown && !IsAttack && !IsHurt && x < SIZE_X - ImgW - frame_of_test) {
		if (!IsFacingR) {
			IsFacingR = true;
		}
		if (IsRun && !InAir) {
			SetXY(x + length * 3, y);
		}
		else if (IsRun && IsHack) {
			SetXY(x + length * 3, y);
		}
		else {
			SetXY(x + length, y);
		}
	}
	// set down attack right and left
	if (IsDown && IsAttack) {
		if (IsFacingR && x < SIZE_X - ImgW - frame_of_test) {
			SetXY(x + length * 3, y);
		}
		else if (x > frame_of_test) {
			SetXY(x - length * 3, y);
		}
	}
	// set jump and fly
	if (IsRising && InAir && !IsAttack && !IsDown && !IsHurt) {
		if (y < sky_top) {		// 碰到天花板
			IsRising = false;
		}
		if (IsJumping) {
			if (velocity > 0) {
				SetXY(x, y - velocity);
				velocity--;
			}
			else {
				IsRising = false;
```

```
                    velocity = 1;
                }
            }
            else if (FlyUp && IsFat) {
                if (fly_velocity > 0) {
                    SetXY(x, y - fly_velocity);
                    fly_velocity--;
                }
                else {
                    IsRising = false;
                    FlyUp = false;
                    fly_velocity = init_fly_velocity;
                }
            }
        }
        else {
            // falling
            if (y < floor - frame_of_test - ImgH) {
                if (IsJumping || !IsFlying) {
                    SetXY(x, y + velocity);
                    velocity++;
                }
                else if (IsFlying) {
                    SetXY(x, y + 1);
                }
                if (YouAreGround) {
                    velocity = init_velocity;
                    InAir = false;
                    IsRising = true;
                    IsJumping = false;
                    KirbyJumpR.Reset();
                    KirbyJumpL.Reset();
                    KirbyFlyR.Reset();
                    KirbyFlyL.Reset();
                }
            }
            else {
                y = floor - frame_of_test - ImgH;
                velocity = init_velocity;
                InAir = false;
```

```
				IsRising = true;
				IsJumping = false;
				KirbyJumpR.Reset();
				KirbyJumpL.Reset();
				KirbyFlyR.Reset();
				KirbyFlyL.Reset();
			}
		}
		// kirby is hurt
		if (IsHurt) {
			if (!OtherFromL && x - 4 > 0) {
				SetXY(x - 4, y);
			}
			else if (x + 4 < SIZE_X - now_img_w) {
				SetXY(x + 4, y);
			}
		}
		if (IsAttack && !IsDown) {
			if (IsFacingR) {
				StarThrow.SetShow(true);
				left_side.SetShow(false);
				StarThrow.SetXy(x + KirbyScreamR.Width(), y + 20);
			}
			else {
				StarThrow.SetShow(false);
				left_side.SetShow(true);
				left_side.SetXy(x - left_side.Width(), y + 20);
			}
		}
		else {
			StarThrow.SetShow(false);
			left_side.SetShow(false);
		}
		// animation OnMove
		KirbyMovingL.OnMove();
		KirbyMovingR.OnMove();
		KirbyStand.OnMove();
		KirbyStandL.OnMove();
	}
	void hotHead_kirby::OnShow()
```

```
{
      if (StarThrow.WeaponIsShow()) {
               StarThrow.OnMove();
               StarThrow.OnShow();
      }
      else if (left_side.WeaponIsShow()) {
               left_side.OnMove();
               left_side.OnShow();
      }
      switch (GetCase()) {
      // case jump up right
      case 1:
               KirbyJumpR.SetDelayCount(4);
               now_img_h = KirbyJumpR.Height();
               now_img_w = KirbyJumpR.Width();
               KirbyJumpR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
               KirbyJumpR.OnMove();
               KirbyJumpR.OnShow();
               break;
      // case down attack right
      case 2:
               KirbyDownAttackR.SetDelayCount(5);
               now_img_h = KirbyDownAttackR.Height();
               now_img_w = KirbyDownAttackR.Width();
               KirbyDownAttackR.SetTopLeft(x, y + KirbyStand.Height() - KirbyDownR.Height());
               KirbyDownAttackR.OnMove();
               KirbyDownAttackR.OnShow();
               if (KirbyDownAttackR.IsFinalBitmap()) {
                        IsAttack = false;
                        KirbyDownAttackR.Reset();
               }
               break;
      // case down right
      case 3:
               now_img_h = KirbyDownR.Height();
               now_img_w = KirbyDownR.Width();
               KirbyDownR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
               KirbyDownR.ShowBitmap();
               break;
      // case scream right
```

```
case 4:
        KirbyScreamR.SetDelayCount(3);
        now_img_h = KirbyScreamR.Height();
        now_img_w = KirbyScreamR.Width();
        KirbyScreamR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyScreamR.OnMove();
        KirbyScreamR.OnShow();
        if (KirbyScreamR.IsFinalBitmap()) {
                IsAttack = false;
                KirbyScreamR.Reset();
        }
        break;
// case walking right
case 5:
        KirbyMovingR.SetDelayCount(2);
        now_img_h = KirbyMovingR.Height();
        now_img_w = KirbyMovingR.Width();
        KirbyMovingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyMovingR.OnShow();
        break;
// case standing right
case 6:
        KirbyStand.SetDelayCount(3);
        now_img_h = KirbyStand.Height();
        now_img_w = KirbyStand.Width();
        KirbyStand.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyStand.OnShow();
        break;
// case fly up right
case 7:
        KirbyFlyR.SetDelayCount(5);
        now_img_h = KirbyFlyR.Height();
        now_img_w = KirbyFlyR.Width();
        KirbyFlyR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyR.OnMove();
        KirbyFlyR.OnShow();
        if (KirbyFlyR.IsFinalBitmap()) {
                KirbyFlyR.Reset();
        }
        break;
```

```
// case flying right
case 8:
        KirbyFlyingR.SetDelayCount(3);
        now_img_h = KirbyFlyingR.Height();
        now_img_w = KirbyFlyingR.Width();
        KirbyFlyingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyingR.OnMove();
        KirbyFlyingR.OnShow();
        if (KirbyFlyingR.IsFinalBitmap()) {
                KirbyFlyingR.Reset();
        }
        break;
// case jump up left
case 9:
        KirbyJumpL.SetDelayCount(4);
        now_img_h = KirbyJumpL.Height();
        now_img_w = KirbyJumpL.Width();
        KirbyJumpL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyJumpL.OnMove();
        KirbyJumpL.OnShow();
        break;
// case down attack left
case 10:
        KirbyDownAttackL.SetDelayCount(5);
        now_img_h = KirbyDownAttackL.Height();
        now_img_w = KirbyDownAttackL.Width();
        KirbyDownAttackL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyDownAttackL.OnMove();
        KirbyDownAttackL.OnShow();
        if (KirbyDownAttackL.IsFinalBitmap()) {
                IsAttack = false;
                KirbyDownAttackL.Reset();
        }
        break;
// case down left
case 11:
        now_img_h = KirbyDownL.Height();
        now_img_w = KirbyDownL.Width();
        KirbyDownL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyDownL.ShowBitmap();
        break;
// case scream left
case 12:
        KirbyScreamL.SetDelayCount(3);
        now_img_h = KirbyScreamL.Height();
        now_img_w = KirbyScreamL.Width();
        KirbyScreamL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyScreamL.OnMove();
        KirbyScreamL.OnShow();
        if (KirbyScreamL.IsFinalBitmap()) {
                IsAttack = false;
                KirbyScreamL.Reset();
        }
        break;
// case walking left
case 13:
        KirbyMovingL.SetDelayCount(2);
        now_img_h = KirbyMovingL.Height();
        now_img_w = KirbyMovingL.Width();
        KirbyMovingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyMovingL.OnShow();
        break;
// case standing left
case 14:
        KirbyStandL.SetDelayCount(3);
        now_img_h = KirbyStandL.Height();
        now_img_w = KirbyStandL.Width();
        KirbyStandL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyStandL.OnShow();
        break;
// case fly up left
case 15:
        KirbyFlyL.SetDelayCount(5);
        now_img_h = KirbyFlyL.Height();
        now_img_w = KirbyFlyL.Width();
        KirbyFlyL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyL.OnMove();
        KirbyFlyL.OnShow();
        if (KirbyFlyL.IsFinalBitmap()) {
                KirbyFlyL.Reset();
                }
                break;
        // case flying left
        case 16:
                KirbyFlyingL.SetDelayCount(3);
                now_img_h = KirbyFlyingL.Height();
                now_img_w = KirbyFlyingL.Width();
                KirbyFlyingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyFlyingL.OnMove();
                KirbyFlyingL.OnShow();
                if (KirbyFlyingL.IsFinalBitmap()) {
                        KirbyFlyingL.Reset();
                }
                break;
        // case hurt right
        case 17:
                KirbyHurtR.SetDelayCount(2);
                now_img_h = KirbyHurtR.Height();
                now_img_w = KirbyHurtR.Width();
                KirbyHurtR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyHurtR.OnMove();
                KirbyHurtR.OnShow();
                if (KirbyHurtR.IsFinalBitmap()) {
                        IsHurt = false;
                        KirbyHurtR.Reset();
                }
                break;
       // case hurt left
        case 18:
                KirbyHurtL.SetDelayCount(2);
                now_img_h = KirbyHurtL.Height();
                now_img_w = KirbyHurtL.Width();
                KirbyHurtL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyHurtL.OnMove();
                KirbyHurtL.OnShow();
                if (KirbyHurtL.IsFinalBitmap()) {
                        IsHurt = false;
                        KirbyHurtL.Reset();
                }
                break;
```

```
// case IsEaten standing right
case 19:
        KirbyFatStand.SetDelayCount(3);
        now_img_h = KirbyFatStand.Height();
        now_img_w = KirbyFatStand.Width();
        KirbyFatStand.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFatStand.OnMove();
        KirbyFatStand.OnShow();
        break;
// case IsEaten standing left
case 20:
        KirbyFatStandL.SetDelayCount(3);
        now_img_h = KirbyFatStandL.Height();
        now_img_w = KirbyFatStandL.Width();
        KirbyFatStandL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFatStandL.OnMove();
        KirbyFatStandL.OnShow();
        break;
// case fat walk right 21
case 21:
        KirbyFatMovingR.SetDelayCount(2);
        now_img_h = KirbyFatMovingR.Height();
        now_img_w = KirbyFatMovingR.Width();
        KirbyFatMovingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFatMovingR.OnMove();
        KirbyFatMovingR.OnShow();
        break;
// case fat walk left 22
case 22:
        KirbyFatMovingL.SetDelayCount(2);
        now_img_h = KirbyFatMovingL.Height();
        now_img_w = KirbyFatMovingL.Width();
        KirbyFatMovingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFatMovingL.OnMove();
        KirbyFatMovingL.OnShow();
        break;
default:
        CMovingBitmap temp;
        temp.LoadBitmap(IDB_KIRBY);
        temp.ShowBitmap();
```

```
			break;
		}
	}
	weapon* hotHead_kirby::GetWeapon() {
		if (IsFacingR) {
			return &StarThrow;
		}
		else {
			return &left_side;
		}
	}
	void hotHead_kirby::SetKindInit() {
		kirby_kind = "hotHead_kirby";
	}
	void waddleDoo_kirby::SetEaten(bool input) {}
	void waddleDoo_kirby::SetEaten(bool input, std::string name) {}
	void waddleDoo_kirby::ThrowStar() {}
	void waddleDoo_kirby::LoadBitmap()
	{
		// load walk left
		char *walk_left[10] = { ".\\res\\kirby_waddleDoo\\walk\\walkL1.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkL2.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkL3.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkL4.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkL5.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkL6.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkL7.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkL8.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkL9.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkL10.bmp" };
		for (int i = 0; i < 10; i++) {
			KirbyMovingL.AddBitmap(walk_left[i], RGB(255, 255, 255));
		}

		// load walk right
		char *walk_right[10] = { ".\\res\\kirby_waddleDoo\\walk\\walkR1.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkR2.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkR3.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkR4.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkR5.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkR6.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkR7.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkR8.bmp", ".\\res\\kirby_waddleDoo\\walk\\walkR9.bmp",
".\\res\\kirby_waddleDoo\\walk\\walkR10.bmp" };
		for (int i = 0; i < 10; i++) {
			KirbyMovingR.AddBitmap(walk_right[i], RGB(255, 255, 255));
		}
		// load stand and wink right
		int count = 12;
		while (count-- > 0) {
			KirbyStand.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicR1.bmp", RGB(255, 255, 
255));
		}
		KirbyStand.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicR2.bmp", RGB(255, 255, 255));
		count = 12;
		while (count-- > 0) {
			KirbyStand.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicR1.bmp", RGB(255, 255, 
255));
		}
		KirbyStand.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicR2.bmp", RGB(255, 255, 255));
		KirbyStand.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicR1.bmp", RGB(255, 255, 255));
		KirbyStand.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicR2.bmp", RGB(255, 255, 255));
		// load stand and wink left
		count = 12;
		while (count-- > 0) {
			KirbyStandL.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicL1.bmp", RGB(255, 255, 
255));
		}
		KirbyStandL.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicL2.bmp", RGB(255, 255, 255));
		count = 12;
		while (count-- > 0) {
			KirbyStandL.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicL1.bmp", RGB(255, 255, 
255));
		}
		KirbyStandL.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicL2.bmp", RGB(255, 255, 255));
		KirbyStandL.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicL1.bmp", RGB(255, 255, 255));
		KirbyStandL.AddBitmap(".\\res\\kirby_waddleDoo\\basic\\basicL2.bmp", RGB(255, 255, 255));
		// load down right and left
		KirbyDownR.LoadBitmap(".\\res\\kirby_waddleDoo\\down\\downR1.bmp", RGB(255, 255, 255));
		KirbyDownL.LoadBitmap(".\\res\\kirby_waddleDoo\\down\\downL1.bmp", RGB(255, 255, 255));
		// load down attack right and left
		count = 3;
		while (count-- > 0) {
			KirbyDownAttackR.AddBitmap(".\\res\\kirby_waddleDoo\\down\\downR2.bmp", RGB(255, 
255, 255));
			KirbyDownAttackL.AddBitmap(".\\res\\kirby_waddleDoo\\down\\downL2.bmp", RGB(255, 
255, 255));
```

```
	}
	// load jump right
	char *jump_right[10] = { ".\\res\\kirby_waddleDoo\\jump\\jumpR1.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpR2.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpR3.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpR4.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpR5.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpR6.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpR7.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpR8.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpR9.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpR10.bmp" };
	for (int i = 0; i < 10; i++) {
		KirbyJumpR.AddBitmap(jump_right[i], RGB(255, 255, 255));
	}
	// load jump left
	char *jump_left[10] = { ".\\res\\kirby_waddleDoo\\jump\\jumpL1.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpL2.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpL3.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpL4.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpL5.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpL6.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpL7.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpL8.bmp", ".\\res\\kirby_waddleDoo\\jump\\jumpL9.bmp",
".\\res\\kirby_waddleDoo\\jump\\jumpL10.bmp" };
	for (int i = 0; i < 10; i++) {
		KirbyJumpL.AddBitmap(jump_left[i], RGB(255, 255, 255));
	}
	// load scream right and left
	count = 6;
	while (count-- > 0) {
		KirbyScreamR.AddBitmap(".\\res\\kirby_waddleDoo\\scream\\screamR2.bmp", RGB(255,
255, 255));
	}
	count = 15;
	while (count-- > 0) {
		KirbyScreamR.AddBitmap(".\\res\\kirby_waddleDoo\\scream\\screamR1.bmp", RGB(255,
255, 255));
	}
	count = 6;
	while (count-- > 0) {
		KirbyScreamL.AddBitmap(".\\res\\kirby_waddleDoo\\scream\\screamL2.bmp", RGB(255,
255, 255));
	}
	count = 15;
	while (count-- > 0) {
		KirbyScreamL.AddBitmap(".\\res\\kirby_waddleDoo\\scream\\screamL1.bmp", RGB(255,
255, 255));
		}
		// load flyR
		KirbyFlyR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR1.bmp", RGB(255, 255, 255));
		KirbyFlyR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR2.bmp", RGB(255, 255, 255));
		KirbyFlyR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR3.bmp", RGB(255, 255, 255));
		count = 3;
		while (count-- > 0) {
			KirbyFlyR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR4.bmp", RGB(255, 255, 255));
		}
		KirbyFlyR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR4.bmp", RGB(255, 255, 255));
		KirbyFlyR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR5.bmp", RGB(255, 255, 255));
		// load flyingR
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR6.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR7.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR8.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR9.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR9.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR8.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR7.bmp", RGB(255, 255, 255));
		KirbyFlyingR.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyR6.bmp", RGB(255, 255, 255));
		// load flyL
		KirbyFlyL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL1.bmp", RGB(255, 255, 255));
		KirbyFlyL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL2.bmp", RGB(255, 255, 255));
		KirbyFlyL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL3.bmp", RGB(255, 255, 255));
		count = 3;
		while (count-- > 0) {
			KirbyFlyL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL4.bmp", RGB(255, 255, 255));
		}
		KirbyFlyL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL5.bmp", RGB(255, 255, 255));
		// load flyingL
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL6.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL7.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL8.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL9.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL9.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL8.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL7.bmp", RGB(255, 255, 255));
		KirbyFlyingL.AddBitmap(".\\res\\kirby_waddleDoo\\fly\\flyL6.bmp", RGB(255, 255, 255));
		// load hurtR
```

```
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR1.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR2.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR3.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR5.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR6.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR7.bmp", RGB(255, 255, 255));
        KirbyHurtR.AddBitmap(".\\res\\hurt\\hurtR8.bmp", RGB(255, 255, 255));
        // load hurtL
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL1.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL2.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL3.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL5.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL6.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL7.bmp", RGB(255, 255, 255));
        KirbyHurtL.AddBitmap(".\\res\\hurt\\hurtL8.bmp", RGB(255, 255, 255));
        // load Star Throw
        int rgb[3] = { 255, 255, 255 };
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR1.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR2.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR3.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR4.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR5.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR6.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR7.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR8.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR9.bmp", rgb, 1);
        StarThrow.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackR10.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL1.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL2.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL3.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL4.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL5.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL6.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL7.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL8.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL9.bmp", rgb, 1);
        left_side.LoadBitmap(".\\res\\weapon\\waddleDoo\\attackL10.bmp", rgb, 1);
}
void waddleDoo_kirby::OnMove()
{
```

```
// set moving XY
const int length = 4;
// set moving XY and frame of test
if (IsMovingL && !IsDown && !IsAttack && !IsHurt && x > frame_of_test) {
        if (IsFacingR) {
                IsFacingR = false;
        }
        if (IsRun && !InAir) {
                SetXY(x - length * 3, y);
        }
        else if (IsRun && IsHack) {
                SetXY(x - length * 3, y);
        }
        else {
                SetXY(x - length, y);
        }
}
if (IsMovingR && !IsDown && !IsAttack && !IsHurt && x < SIZE_X - ImgW - frame_of_test) {
        if (!IsFacingR) {
                IsFacingR = true;
        }
        if (IsRun && !InAir) {
                SetXY(x + length * 3, y);
        }
        else if (IsRun && IsHack) {
                SetXY(x + length * 3, y);
        }
        else {
                SetXY(x + length, y);
        }
}
// set down attack right and left
if (IsDown && IsAttack) {
        if (IsFacingR && x < SIZE_X - ImgW - frame_of_test) {
                SetXY(x + length * 3, y);
        }
        else if (x > frame_of_test) {
                SetXY(x - length * 3, y);
        }
}
```

```
// set jump and fly
if (IsRising && InAir && !IsAttack && !IsDown && !IsHurt) {
        if (y < sky_top) {          // 碰到天花板
                IsRising = false;
        }
        if (IsJumping) {
                if (velocity > 0) {
                        SetXY(x, y - velocity);
                        velocity--;
                }
                else {
                        IsRising = false;
                        velocity = 1;
                }
        }
        else if (FlyUp && IsFat) {
                if (fly_velocity > 0) {
                        SetXY(x, y - fly_velocity);
                        fly_velocity--;
                }
                else {
                        IsRising = false;
                        FlyUp = false;
                        fly_velocity = init_fly_velocity;
                }
        }
}
else {
       // falling
        if (y < floor - frame_of_test - ImgH) {
                if (IsJumping || !IsFlying) {
                        SetXY(x, y + velocity);
                        velocity++;
                }
                else if (IsFlying) {
                        SetXY(x, y + 1);
                }
                if (YouAreGround) {
                        velocity = init_velocity;
                        InAir = false;
```

```
                    IsRising = true;
                    IsJumping = false;
                    KirbyJumpR.Reset();
                    KirbyJumpL.Reset();
                    KirbyFlyR.Reset();
                    KirbyFlyL.Reset();
                }
            }
            else {
                y = floor - frame_of_test - ImgH;
                velocity = init_velocity;
                InAir = false;
                IsRising = true;
                IsJumping = false;
                KirbyJumpR.Reset();
                KirbyJumpL.Reset();
                KirbyFlyR.Reset();
                KirbyFlyL.Reset();
            }
        }
        // kirby is hurt
        if (IsHurt) {
            if (!OtherFromL && x - 4 > 0) {
                SetXY(x - 4, y);
            }
            else if (x + 4 < SIZE_X - now_img_w) {
                SetXY(x + 4, y);
            }
        }
        if (IsAttack && !IsDown) {
            if (IsFacingR) {
                StarThrow.SetShow(true);
                left_side.SetShow(false);
                StarThrow.SetXy(x + KirbyScreamR.Width() - 5, y - 55);
            }
            else {
                StarThrow.SetShow(false);
                left_side.SetShow(true);
                left_side.SetXy(x - left_side.Width() + 5, y - 55);
            }
```

```
	}
	else {
		StarThrow.SetShow(false);
		left_side.SetShow(false);
	}
	// animation OnMove
	KirbyMovingL.OnMove();
	KirbyMovingR.OnMove();
	KirbyStand.OnMove();
	KirbyStandL.OnMove();
}
void waddleDoo_kirby::OnShow()
{
	if (StarThrow.WeaponIsShow()) {
		StarThrow.OnMove();
		StarThrow.OnShow();
	}
	else if (left_side.WeaponIsShow()) {
		left_side.OnMove();
		left_side.OnShow();
	}
	switch (GetCase()) {
	// case jump up right
	case 1:
		KirbyJumpR.SetDelayCount(4);
		now_img_h = KirbyJumpR.Height();
		now_img_w = KirbyJumpR.Width();
		KirbyJumpR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
		KirbyJumpR.OnMove();
		KirbyJumpR.OnShow();
		break;
      // case down attack right
	case 2:
		KirbyDownAttackR.SetDelayCount(5);
		now_img_h = KirbyDownAttackR.Height();
		now_img_w = KirbyDownAttackR.Width();
		KirbyDownAttackR.SetTopLeft(x, y + KirbyStand.Height() - KirbyDownR.Height());
		KirbyDownAttackR.OnMove();
		KirbyDownAttackR.OnShow();
		if (KirbyDownAttackR.IsFinalBitmap()) {
			IsAttack = false;
			KirbyDownAttackR.Reset();
		}
		break;
	// case down right
	case 3:
		now_img_h = KirbyDownR.Height();
		now_img_w = KirbyDownR.Width();
		KirbyDownR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
		KirbyDownR.ShowBitmap();
		break;
	// case scream right
	case 4:
		KirbyScreamR.SetDelayCount(3);
		now_img_h = KirbyScreamR.Height();
		now_img_w = KirbyScreamR.Width();
		KirbyScreamR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
		KirbyScreamR.OnMove();
		KirbyScreamR.OnShow();
		if (KirbyScreamR.IsFinalBitmap() || StarThrow.IsFinalBitmap()) {
			SetAttack(false);
			KirbyScreamR.Reset();
		}
		break;
	// case walking right
	case 5:
		KirbyMovingR.SetDelayCount(2);
		now_img_h = KirbyMovingR.Height();
		now_img_w = KirbyMovingR.Width();
		KirbyMovingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
		KirbyMovingR.OnShow();
		break;
	// case standing right
	case 6:
		KirbyStand.SetDelayCount(3);
		now_img_h = KirbyStand.Height();
		now_img_w = KirbyStand.Width();
		KirbyStand.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
		KirbyStand.OnShow();
		break;
```

```
// case fly up right
case 7:
        KirbyFlyR.SetDelayCount(5);
        now_img_h = KirbyFlyR.Height();
        now_img_w = KirbyFlyR.Width();
        KirbyFlyR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyR.OnMove();
        KirbyFlyR.OnShow();
        if (KirbyFlyR.IsFinalBitmap()) {
                KirbyFlyR.Reset();
        }
        break;
// case flying right
case 8:
        KirbyFlyingR.SetDelayCount(3);
        now_img_h = KirbyFlyingR.Height();
        now_img_w = KirbyFlyingR.Width();
        KirbyFlyingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyFlyingR.OnMove();
        KirbyFlyingR.OnShow();
        if (KirbyFlyingR.IsFinalBitmap()) {
                KirbyFlyingR.Reset();
        }
        break;
// case jump up left
case 9:
        KirbyJumpL.SetDelayCount(4);
        now_img_h = KirbyJumpL.Height();
        now_img_w = KirbyJumpL.Width();
        KirbyJumpL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyJumpL.OnMove();
        KirbyJumpL.OnShow();
        break;
// case down attack left
case 10:
        KirbyDownAttackL.SetDelayCount(5);
        now_img_h = KirbyDownAttackL.Height();
        now_img_w = KirbyDownAttackL.Width();
        KirbyDownAttackL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
        KirbyDownAttackL.OnMove();
```

```
            KirbyDownAttackL.OnShow();
            if (KirbyDownAttackL.IsFinalBitmap()) {
                    IsAttack = false;
                    KirbyDownAttackL.Reset();
            }
            break;
    // case down left
    case 11:
            now_img_h = KirbyDownL.Height();
            now_img_w = KirbyDownL.Width();
            KirbyDownL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyDownL.ShowBitmap();
            break;
    // case scream left
    case 12:
            KirbyScreamL.SetDelayCount(3);
            now_img_h = KirbyScreamL.Height();
            now_img_w = KirbyScreamL.Width();
            KirbyScreamL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyScreamL.OnMove();
            KirbyScreamL.OnShow();
            if (KirbyScreamL.IsFinalBitmap() || left_side.IsFinalBitmap()) {
                    SetAttack(false);
                    KirbyScreamL.Reset();
            }
            break;
   // case walking left
    case 13:
            KirbyMovingL.SetDelayCount(2);
            now_img_h = KirbyMovingL.Height();
            now_img_w = KirbyMovingL.Width();
            KirbyMovingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
            KirbyMovingL.OnShow();
            break;
    // case standing left
    case 14:
            KirbyStandL.SetDelayCount(3);
            now_img_h = KirbyStandL.Height();
            now_img_w = KirbyStandL.Width();
            KirbyStandL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
```

```
                KirbyStandL.OnShow();
                break;
        // case fly up left
        case 15:
                KirbyFlyL.SetDelayCount(5);
                now_img_h = KirbyFlyL.Height();
                now_img_w = KirbyFlyL.Width();
                KirbyFlyL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyFlyL.OnMove();
                KirbyFlyL.OnShow();
                if (KirbyFlyL.IsFinalBitmap()) {
                        KirbyFlyL.Reset();
                }
                break;
        // case flying left
        case 16:
                KirbyFlyingL.SetDelayCount(3);
                now_img_h = KirbyFlyingL.Height();
                now_img_w = KirbyFlyingL.Width();
                KirbyFlyingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyFlyingL.OnMove();
                KirbyFlyingL.OnShow();
                if (KirbyFlyingL.IsFinalBitmap()) {
                        KirbyFlyingL.Reset();
                }
                break;
       // case hurt right
        case 17:
                KirbyHurtR.SetDelayCount(2);
                now_img_h = KirbyHurtR.Height();
                now_img_w = KirbyHurtR.Width();
                KirbyHurtR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
                KirbyHurtR.OnMove();
                KirbyHurtR.OnShow();
                if (KirbyHurtR.IsFinalBitmap()) {
                        IsHurt = false;
                        KirbyHurtR.Reset();
                }
                break;
       // case hurt left
```

```
 case 18:
         KirbyHurtL.SetDelayCount(2);
         now_img_h = KirbyHurtL.Height();
         now_img_w = KirbyHurtL.Width();
         KirbyHurtL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
         KirbyHurtL.OnMove();
         KirbyHurtL.OnShow();
         if (KirbyHurtL.IsFinalBitmap()) {
                 IsHurt = false;
                 KirbyHurtL.Reset();
         }
         break;
 // case IsEaten standing right
 case 19:
         KirbyFatStand.SetDelayCount(3);
         now_img_h = KirbyFatStand.Height();
         now_img_w = KirbyFatStand.Width();
         KirbyFatStand.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
         KirbyFatStand.OnMove();
         KirbyFatStand.OnShow();
         break;
// case IsEaten standing left
 case 20:
         KirbyFatStandL.SetDelayCount(3);
         now_img_h = KirbyFatStandL.Height();
         now_img_w = KirbyFatStandL.Width();
         KirbyFatStandL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
         KirbyFatStandL.OnMove();
         KirbyFatStandL.OnShow();
         break;
// case fat walk right 21
 case 21:
         KirbyFatMovingR.SetDelayCount(2);
         now_img_h = KirbyFatMovingR.Height();
         now_img_w = KirbyFatMovingR.Width();
         KirbyFatMovingR.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
         KirbyFatMovingR.OnMove();
         KirbyFatMovingR.OnShow();
         break;
 // case fat walk left 22
```

```
		case 22:
			KirbyFatMovingL.SetDelayCount(2);
			now_img_h = KirbyFatMovingL.Height();
			now_img_w = KirbyFatMovingL.Width();
			KirbyFatMovingL.SetTopLeft(x, y + KirbyStand.Height() - now_img_h);
			KirbyFatMovingL.OnMove();
			KirbyFatMovingL.OnShow();
			break;
		default:
			CMovingBitmap temp;
			temp.LoadBitmap(IDB_KIRBY);
			temp.ShowBitmap();
			break;
		}
	}
	void waddleDoo_kirby::SetAttack(bool input) {
		if (!IsHurt) {
			IsAttack = input;
		}
		if (IsFlying) {
			SetFly(false);
		}
		if (!input) {
			StarThrow.AnimationReset();
			left_side.AnimationReset();
		}
	}
	void waddleDoo_kirby::SetKindInit() {
		kirby_kind = "waddleDoo_kirby";
	}
}
```