計算機程式期末專題書面報告

第七組 組員：陳新元 陳泓均 鄭人豪

遊戲名稱：打殭屍遊戲

一. 程式中的 class

我們分幾個部分討論 ：

(1) 武器 ：

Showbullet 用來處理跟剩餘子彈相關的參數及文字的投影，因為有很多種武器，所以在處理關於子彈數量的部分時，我只需要在宣告的後面傳入不同的參數，之後在程式中就可以簡單的呼叫 class 裡的 function。

Showgun 因為 pistol, UZI, shotgun 動畫的投影是用 SDL Draw Line，所以我把他們用在同一個 class，透過簡單的 class 呼叫 function 並傳入參數就可以畫出想要的線條，藉此讓程式較乾淨，此 class 中有判斷是否命中殭屍的 function 藉此 function 微調動畫線的長度，利用 class 有效的簡化 main 裡面的程式，讓 main 不會看起來太冗長。

showgenerade 是一個用來處理手榴彈的 class，裡面包括手榴彈應該投影位置的參數，還有爆炸的圖，透過物件化，在 main 裡呼叫 function 時可以有效率並且不會呼叫錯 function。

shield 用來處理跟防護罩相關的 class，包含投影跟主角是否進入防護罩的判斷。

atom 用來處理原子彈的 class，裡面有會隨時間變大的長方形，利用這個長方形把同一張圖片投影上去產生不同的大小，達到原子彈的效果。

mousebuttom 是用來處理初始介面的 function，因為開始玩遊戲時是利用滑鼠來判斷是否進入遊戲所以名稱命名為 mousebuttom，他是負責處理跟主畫面有關的 class，可以有效節省 main 程式碼的長度。

fireball 是處理地圖上魔王火球的 class，利用初始發射時主角的 x y 座標做出軌道，然後再判斷是否有碰到主角而消失。

score 是處理跟分數有關的 class，隨著分數的提高，他會秀出跑馬字條讓玩家知道自己獲得什麼武器或武器增強，武器升級分為兩個部分，第一階段是提升威力，第二階段是加快攻擊速度，讓玩家有不一樣的體驗，private 的運用保護了分數，確定分數不會被任意改動到。

(2)人物 ：

人物方面使用了 3 個 class，分別是 Dot、dot1、zombie。首先講一下為什麼會有 dot 和 dot1，因為一開始我們設計雙人遊戲，但兩隻主角其中有功能不太相同之處，因此分作兩個 class 處理比較方便，接著因為相同的部分還是很多，所

以共同繼承 Dot 這個 class，但後來因時間不足因此拿掉一個角色，所以就刪掉 dot2，而最後沒有把 Dot 和 dot1 合併是因為想到 zombie 這個人物也可以繼承 Dot class，這樣在 dot1(主角)和 zombie 互相使用對方的物件為參數時，可以看成 Dot 的物件。

(3)LTexture

其實這個是 SDL 裡面的一個 class，是 SDL tutorial 本來就寫好的 class，我們只是將它分開獨立成一個檔案，然後增加一些額外的功能。

二. 封裝(encapsulation)

此處舉 bucket 與 redbox 的 class 做說明。

a. bucket：bucket class 中

bool pos_xy_bucket[LEVEL_WIDTH1/40][LEVEL_HEIGHT1/63] 代表地圖上柱子(一種武器)所在的位置，

bool pos_xy_wall[LEVEL_WIDTH1/40][LEVEL_HEIGHT1/63] 代表地圖上柱子(一種武器)所在的位置

bool pos_xy[LEVEL_WIDTH1/40][LEVEL_HEIGHT1/63] 代表地圖上的這個座標是否放有柱子或`是油桶。

bucket class 中的 member function 皆是根據以上三種資訊而建構 member function，以上三個 bool 陣列皆為 private 型態。如此作法考量到利用此 bucket class 只有 dot class 中 cantgo member function：

```
void Dot:: cantgo(bucket & bucketwall,int camX, int camY){
  int i=0,j=0;

for(i=0;i<LEVEL_WIDTH1/40;i++){
     for(j=0;j<LEVEL_HEIGHT1/63;j++){
  if (((bucketwall.pos_xy_wall[i][j+1]==true))&&i==mPosX/40&&j==mPosY/63)
{   mPosX -= mVelX;mPosY -= mVelY;}

}
}
}
```

假使我們將上述三項 bool 陣列設為 public，那萬一在其他程式碼更動到此陣列中存放的值，compiler 將無法警告我們，就如同老師在課堂中介紹 encapsulation 時，提到若擔心這點，可以將 class 的 member 設為 private，確保程式中與欲保護的資訊不相干的部分不會有所更動。如果將他們設成 public，遊戲執行後出了問題，bug 將難以找出，因此在考慮到資訊的安全系，

並兼具資料獲取、更改的便利性與效率，我們令 Dot 是 bucket 的 friend class：class bucket { friend class Dot; 。

b. redbox class

redbox class 與 bucket class 具有類似的性質與功能，就是判斷地圖上有沒有此物(redbox、bucket)，根據位置 render 圖上去並成為其他 class 中掌管移動方式(Dot 的 cantgo)，控制彈藥數量(weapon 的 supplementweapon)的一樣指標。不同於 bucket class 的安排，int posxofred[16]; int posyofred[16]; 為 private 代表 16 個 rebox 的位置。 public:bool pos_xy_red [16];代表第 n 個 redbox 是否還存在在它預設的位置上。因為遊戲預設的 redbox 數量固定為 16 個，加上 main 中僅有一處修改 posxofred[n]的內容，所以考慮到方便性，並確保資料該有的安全性，我們將 bool pos_xy_red [16]設為 public 讓它在 main 裡判斷後可立即改值：

```
for (int i=0;i<=59;i++)
        {
          for (int j=0;j<=19;j++)
            bucketwall.renderwall(camera,gRenderer,wTexture,i,j);
                    }
                      for(int i=0;i<16;i++){
          hitthered=false;
          hitthered=RED.hitthered(dot1,i);

          if
(hitthered==true){if( RED.pos_xy_red[i]==true){weapons.supplementweapon(ramdo
m,uzibullet,shotgunbullet,generadebullet,wallbullet,shieldbullet,atombombbullet);}/
/start=clock();}
          RED.pos_xy_red[i]=false;

          } }if(countred==1000){for(int i=0;i<16;i++){RED.pos_xy_red[i]=true; }}
```

16 個 rebox 個別的位置都是在遊戲剛開始就固定不再更改。

三.建構子(Constructor)&解構子(Destructor)

constructor 和 destructor 的應用，此處以 weapon.h 裡的 class showbullet 作介紹

```
showbullet::showbullet(char* weapon,int a, int b,int c,int d,int f,int i)
{
  bullets[0]=a+48;
  bullets[1]=b+48;
```

```
  bullets[2]=c+48;
  speedconstant=d;
  damage=f;
  damageupgrade=i;
  for
(stringlength=0;(weaponremainging[stringlength]=weapon[stringlength])!='\0';string
length++) ;
  mTexture = NULL;
      mWidth = 0;
      mHeight = 0;
      upgrade=0;
}
```

首先這個 clsss 是用來投影子彈剩餘的數目，所以當它被 construct 的時候，會接受第一個是關於武器名稱的字元指標，接著三個參數是初始化子彈的數量，又因為武器是用方向來區分為 class，所以在這裡因為是關於武器種類的 class，所以我們把每種武器的內部性質都設在這裡，因此第五個是接收武器的攻擊速度，第六第七分別是武器升級前升級後的威力。

在 consturctor 裡，用一個 char 陣列把傳入的子彈數存起來，接著設定攻擊速度跟傷害，然後設定第一個傳入的字串長度以利將來把子彈數和字串合起來，最後是關於投影的變數初始化。

```
showbullet::~showbullet()
{
  free();
}
```

解構時會呼叫 class 的 free function

```
void showbullet::free()
{
      //Free texture if it exists
      if( mTexture != NULL )
      {
            SDL_DestroyTexture( mTexture );
            mTexture = NULL;
            mWidth = 0;
            mHeight = 0;
      }
}
```

free 會把跟投影相關的 mTexture 裡的東西毀掉並讓他指向 null。

四.組裝 (composition) &繼承 (inheritance)

在程式中，我們以人物來實作繼承。利用繼承的幾項優點：
(1)以一個 class 為基底，讓多個 class 作為其擴充
(2)縮減程式碼的撰寫
(3)兩繼承於相同 base class 的 member 可以相互影響
以下為各點說明和舉例：
(1)我們的遊戲有兩種角色，主角與殭屍。由於操控兩種角色移動模式、判斷障礙物、生命值等 function 以及他們具備的速度、方向和生命值在撰寫程式碼上具有高度的相同性。因此我們以名為 Dot 的 class 為基底，zombie(僵屍的 class)與 dot1(主角的 class)繼承 Dot，如此可大大縮減重複程式碼的撰寫。
Base Class：

```
class Dot
{
  friend class bucket;
  friend class zombie;

  public:
     int    get_relative_pos_x(SDL_Rect&);
     int    get_relative_pos_y(SDL_Rect&);
     int rposx;
     int rposy;
            //The dimensions of the dot
            static const int DOT_WIDTH = 20;
            static const int DOT_HEIGHT = 20;
            //Maximum axis velocity of the dot
            static const int DOT_VEL = 4;
            //Initializes the variables
            Dot();
            //Takes key presses and adjusts the dot's velocity

            //related to health
            void hurt(int damage){ characterhealth -= damage; }
//        void heal(int restore){ changeHealth()health += restore; }
     void changeHealth(int h){ characterhealth = h; }
     int getHealth(){ return characterhealth; }
```

```
        //Moves the dot
        void move(int camX, int camY,bucket & bucketwall ,Dot&);
    void cantgo(bucket & bucketwall,int camX, int camY );

        //Shows the dot on the screen relative to the camera
        void render( int camX, int camY,LTexture&gDotTexture, SDL_Renderer*
gRenderer   );
        int getPosX();
        int getPosY();
        //related to judging moving
        void setmoving(bool a){ moving = a; }
    bool getmoving(){ return moving ; }
    void setDirection(int a){ direction = a ;}
    int getDirection(){ return direction; }
    //related to counters
    void addCounter(){ counter++;}
    int getCounter(){ return counter ;}
    void addHitCounter(){ hitCounter++;}
    int getHitCounter(){ return hitCounter ;}
    void addDieCounter(){ dieCounter++;}
    int getDieCounter(){ return dieCounter ;}
    void setDieCounter(int a){ dieCounter = a;}
     // related to hit or die(hurt)
    bool getHit(){ return hit;}
    void setHit(bool a){ hit = a;}
    void setDie(bool a){ die = a;}
    bool getDie(){ return die;}
    //related to loading media
    bool loadMediaMove(SDL_Renderer* gRenderer , LTexture[4][10][3]);
protected:
        int mPosX,mPosY;
        int characterhealth;
        int mVelX, mVelY;
        bool hit ;
        bool die;
        int direction;
        bool moving;
        int counter;
```

```
        int weapon_count ;
        int dieCounter;
        int hitCounter; };
```

以 Base Class 的擴充：

```
class Dot1 : public Dot{
public :
  void handleEvent( SDL_Rect &camera,SDL_Renderer* gRenderer,LTexture&
wTexture,SDL_Event& e, bucket& bucketwall );
};
class zombie : public Dot
```

(2)利用 class 大量縮減程式碼為程式帶來執行上的效率，更實用的是當眼前有幾千行程式碼的時候出了一個 bug，因較簡潔有力的撰寫為我們帶來 debug 的便利性。

(3)我們在 zombie　class 中利用繼承與複寫的功能提高一個 class 的可塑性與利用價值。例如：

```
bool zombie::loadMediaMove(SDL_Renderer* gRenderer, LTexture
oveTexture[10][3])

bool Dot::loadMediaMove(SDL_Renderer* gRenderer,LTexture
moveTexture[4][10][3])
```

兩個名稱相同的 function 同樣是處理圖片的上載，但因角色不同故上載不同的圖。

實作上述的三個 class 達成利用現有的基本或基礎類別來擴充、實作新的類別與功能，達到 software reuse 的目標。

五.分工

(1) 泓均

map.h 裡面 dot class 裡面的 public

(1)related to judging moving

(2) //related to counters

(3)// related to hit or die(hurt)

(4)//related to loading media

protected 當中大部分

map.cpp 裡面， dot 的 constructor 和 loadMediaMove

zombie.h 和 zombie.cpp 也是差不多的情況

main 當中讓殭屍跟著主角走，控制主角的八個方向，以及 render 主角和殭屍的圖。

(2) 新元

weapon.h 裡的所有 class，大部分是跟武器相關，還有初始介面跟魔王的火球。

(3) 人豪

map.h bucket class

map.h Dot class 其中的 move、 cantgo、render、

map.h redbox class

和一些 zombie class 裡控制殭屍移動，判斷碰撞(bool collision())

在 main 中控制主角移動的 sdl event，判斷圖上的角色是否碰上障礙物與殭屍間的碰撞，和一些有關地圖上紅色箱子(redbox)的放置，以及地圖跟隨主角而移動 camera 的部分。

六.個人補充與設計特色

a. 合併上遇到的困難

泓均：

因為當初是我負責主角，人豪負責地圖，新元負責武器，因此想說地圖和主角先合起來，最後再把武器加上去，但是我們地圖和主角一開始合起來後，主角卻變得卡住不會動，也找不出原因，因為原本是把地圖合過來主角這邊，所以後來我就決定把主角合過去(因為如果人物用點顯示是可以跑的，表示程式沒有問題)，整個重來又多花了三個多小時，但是最後終於成功了。

新元：
因為當初沒有拆檔案，全部都寫在同一個 cpp 裡，所以當自己把它拆成三個檔案時，忘記 global 變數不能名稱一樣，就這樣卡了很久，最後經過老師的點悟後，就勢如破竹，沒有遇到什麼困難，但是在最終和大家的遊戲程式併在一起時，因為時間不夠充裕，所以在 main 的 global 裡設了兩個 global 變數去存取主角 class 裡的 xy 座標，再利用這個 xy 座標去設定我的武器，有種小作弊的感覺。
人豪：
剛開始研究如何讓視窗中的底圖跟隨著主角的移動碰上了相對位置和原理的問題，我是靠著繪製了一張角色和 camera 之間相對位置關係圖才解決這方面的疑惑。再來就是為了解決碰撞的問題，剛開始嘗試在 main 裡做兩物相撞的判斷，藉此區隔重疊的兩物，但是成效不佳，於是改在每個 class 中為不同的 class 量身打造最適合的判斷碰撞的 function，如 zombie 跟 Dot 的 cantgo ，zombie 的 collision 、redbox 的 hitthered 和 bucket 的 judgethebarrier。

b.設計特色

這邊我要說一下為甚麼我們光合併就花那麼多時間，因為主角和殭屍的圖都有八個方向，每個方向又有三張圖，而且當初我們是設計主角和殭屍被攻擊、攻擊別人、死掉都有不同的圖，因此圖片量很大，因此程式碼也相對變的很多(一張圖就要多一行)，所以只要動到角色相關的修改，一點小修改就需要很多的時間，雖然最後沒有時間，我們的圖就沒有那麼多變(看起來也沒那麼生動)，不過我們還是堅持八個方向，因為這樣畢竟操作起來比較順手，而這也是我們的一大賣點(好像很少組有八個方向的移動，我記得是沒有拉)。

多樣化的武器，而每一項武器都是需要時間去完成的，武器最難的部分就是判斷，判斷範圍大了會讓別人覺得這個遊戲不精緻，但當範圍小了會讓人覺得這個遊戲有 bug，所以當我在打程式時，對武器的判斷特別要求，常常用投影線的方式去測量武器跟命中僵屍的部分，還有武器的創新，因為不能只用現有的武器，還要外加自己和組員的創意，所以推出原子彈跟防護罩，增加遊戲好玩的程度，而魔王的火球也是精心設計，為了要讓遊戲有挑戰度，會從兩個地方分別射出三顆火球，其中一顆會朝原本主角所在的位子放下，另外兩顆會是一上一下隨機發射，並且隨著分數的提高，火球的速度會變快。

我們這組是少數利用 camera 來實現大地圖的組別。因為光是處理座標就必須考慮絕對座標、相對座標。經過多次的實驗，要能在這種大地圖伴隨著 camera 移動而改變視域的遊戲架構下，為每個物件設置絕對位置與相對位置。先用絕對位置判斷碰撞，如：兩物相撞、子彈擊中、障礙物的觸發和補包(redbox)與主角的接觸。再將絕對座標與 camera 座標相減得到相對座標，依照相對座標 render 到畫面中。