計算機程式期末專題報告

B13901139 王兆國

WilliamWang941225@gmail.com

https://williamwang941225.github.io/Website/

December 26, 2024

目錄

1 專題內容

1.1 專題整體介紹與說明文案

我們製作的遊戲名稱為東方森幻記，源自於知名的遊戲 ip「東方 project」遊戲是 pixel 感的 2D 平台跳躍，目標是像馬力歐或是洛克人一樣躲避路上的互動陷阱最終打倒 boss，同時我們也加入一些開啟箱子蒐集彩蛋的隱藏要素。故事背景: 前言: 某天，魔理沙煮了一大鍋蘑菇料理，邀請靈夢去她魔法森林的木屋品嘗，但魔理沙剛吃下料理便失去了意識並向頂樓飛去，於是靈夢決定動手把她打醒並問問她到底蘑菇是在哪裡採的。後記: 兩人接下來去了永遠亭看診，在回想了當天的經過後，魔理沙說她採蘑菇時有遇見戀戀，又因為魔法森林的蘑菇常會吸收周遭的魔力，所以永琳推測當時的蘑菇可能剛好受到碰巧經過的戀戀的魔力影響，導致蘑菇本身的毒素被加強或是特性被改變，至於飛向頂樓可能是因為魔理沙有觀看星空的習慣導致下意識的行為。

1.2 功能設計

1.2.1 方塊

遊戲裡共有六種方塊，不同的方塊有不同的渲染樣式及與角色的互動方式

1. Normal：普通的方塊，構成牆壁或是地板
2. Spike：尖刺，角色碰到會掉血
3. Chest：寶箱，可以被角色打開並掉落彩蛋
4. Cannon：大砲，發射子彈攻擊玩家
5. Fake：沒有碰撞箱的假方塊，角色碰到前渲染成一般的牆壁，碰到後渲染成空氣
6. Nothing：透明的空氣方塊，沒有碰撞箱，構成背景

1.2.2 角色

1. player：遊戲角色
    - 可以跑步、跳躍、射擊子彈。
    - 碰到尖刺、大砲子彈、boss 子彈會扣血。
    - 每個 tick 會自動回血與補充子彈。
2. boss：魔王
    - 發射子彈攻擊玩家。有兩種模式：自動瞄準、隨機發射。

1.2.3 子彈

1. `player_bullets`：角色子彈
    - 射到 boss 會扣血。
2. `boss_bullets`：boss 子彈
    - 射到遊戲角色會扣血。
3. `cannon_bullets`：大砲子彈
    - 射到遊戲角色會扣血。

1.3 美工介面設計

1.3.1 圖片

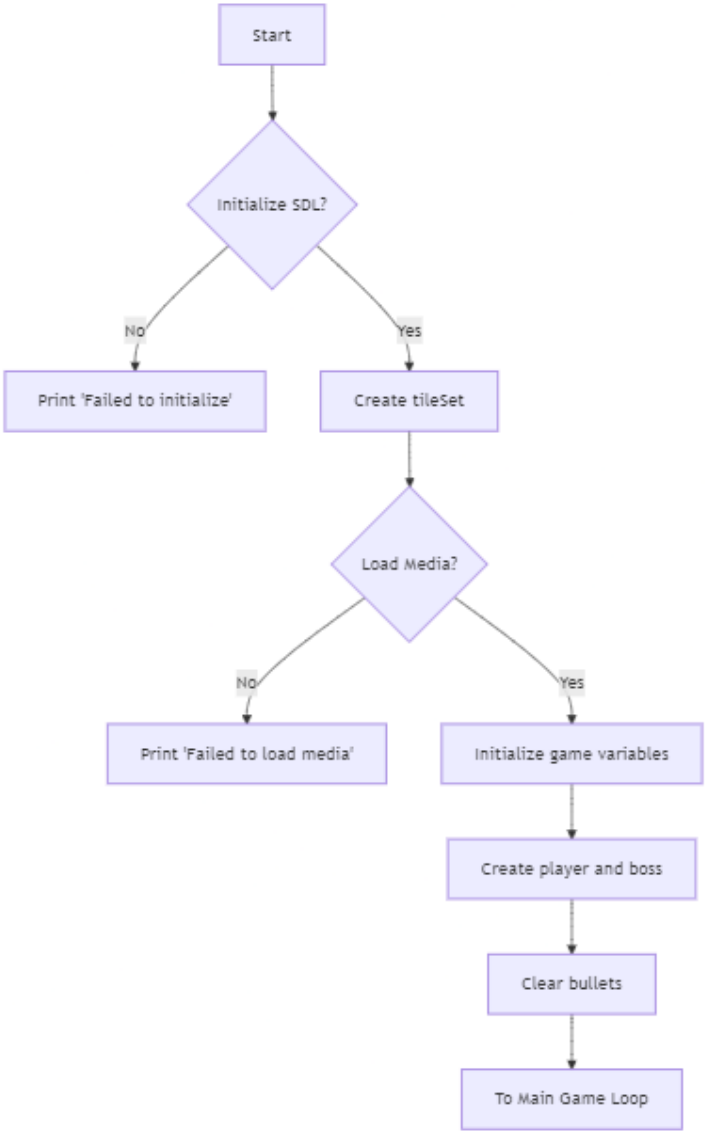
遊戲採用 pixel 風格，利用https://www.piskelapp.com/做圖片的編輯。

1. 人物: 採用網路上東方 project 的像素風圖案，再加以編輯。
2. 方塊: 採用 minecraft 的木材圖片，其餘則自繪。
3. 背景圖: 配合故事背景畫了叢林風格的地圖
4. 起始背景圖: 自繪，做出兩人對決的風格，並以方塊素材裝飾 #### 音效為了和像素風的遊戲搭配，於是特別採用了網路上的 8bit 的音樂素材作為背景音樂及魔王關音樂。角色音效則是在網路上尋找適合的素材剪輯而成。

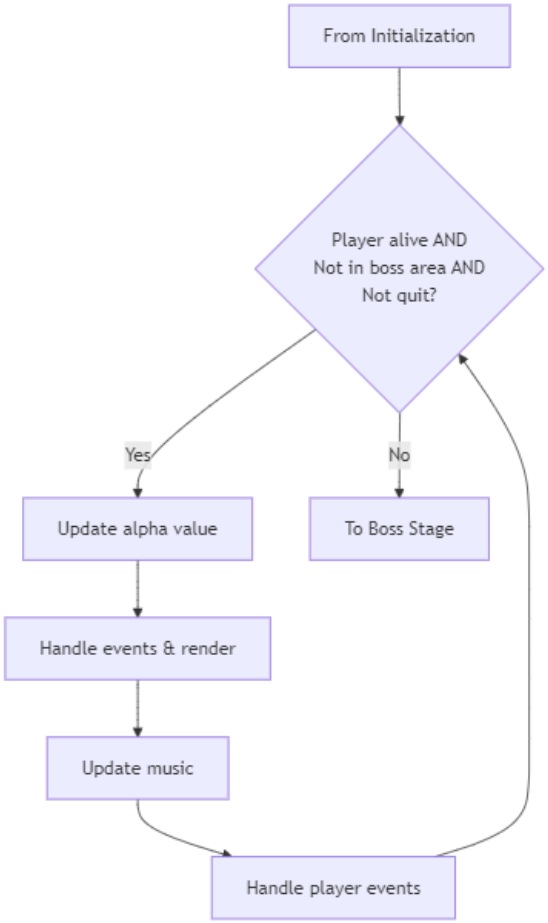
1.4 程式架構

遊戲流程:

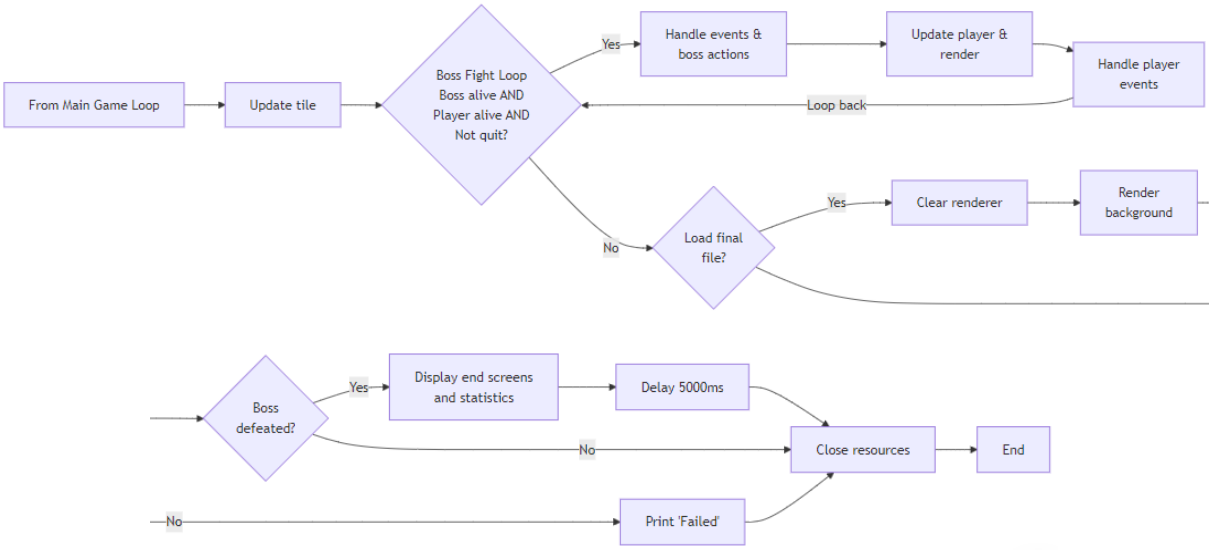
1. 初始化階段


Start
Initialize SDL?
No
Yes
Print 'Failed to initialize'
Create tileSet
Load Media?
No
Yes
Print 'Failed to load media'
Initialize game variables
Create player and boss
Clear bullets
To Main Game Loop

2. 主遊戲循環

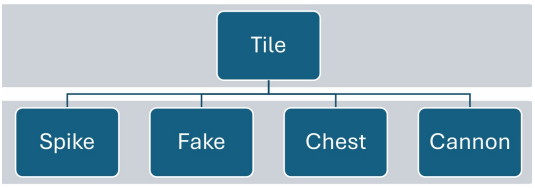


3. Boss 戰鬥階段和結束

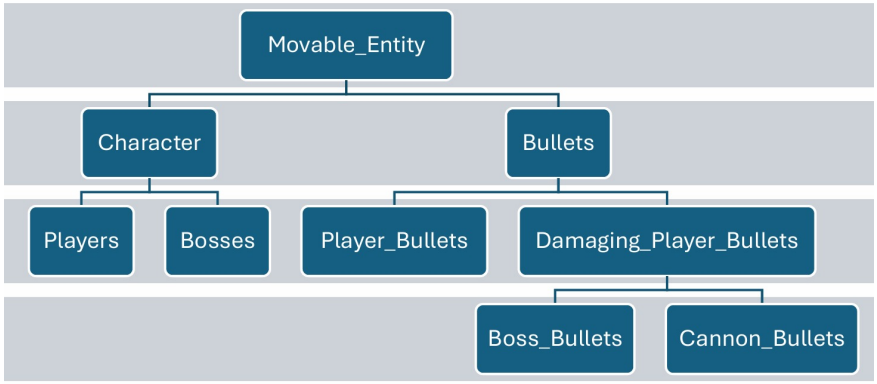


（含分工）架構圖：

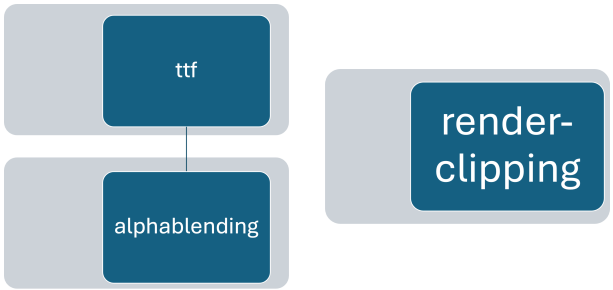
1. 林守威
   Tile 的類別及其衍生類別實作、init/close 的 Tile 及渲染相關函式 Tile 類別架構圖:



2. 王兆國
   移動的類別及其衍生類別實作，整個畫面的渲染與角色動作控制。



3. 陳毅
   部分圖片 rendering, clipping，與 ttf, alpha,blending.



1.5 心得與建議

1. 林守威
   儘管仍有許多想完成的功能因為各種問題無法實現，但是我對自己的作品感到很滿意。因為**遊戲的構想，地圖、互動機制、遊戲流程的概念發想都是由我負責**，在反覆修改的過程中，不斷思考如何在讓玩家體驗更順暢、遊戲內容更豐富的同時，盡力控制程式架構的難度讓初學程式的我們有能力在這段極為有限的期末時間內完成。

從原本企劃書的規劃到如今的架構，每晚的苦思都讓我更了解到遊戲設計的深度與難度。因此看著自己和組員合作一步步地從零開始學習 SDL 到最後完成一個完整的專案，以及**將上課學到的 polymorphism 等概念應用在 Tile 類別內**讓我有莫大的成就感。

未來展望: 放遊戲初代流程圖與後面的流程圖。

2. 王兆國

完成專題後對自己的作品感到滿意，不是因為完成多厲害的功能與設計，而是認知到自己能夠運用有限的線上教學，從原本看似功能很破爛的程式庫一一實作各種角色的精細操作與物件的相互作用。`Players` 的 `handleEvent()` 是我精心製作的程式，使用設定的跑步與跳躍參數控制 `Player` 的動作，並且讓 `Player` 能夠滿足物理規則。`Players` 的 `FullRender()` 整合組員各自分工的程式，過程中發現有多餘的 class，於是改為**利用 polymorphism 將各種繼承的 class 以巧妙簡潔的方式撰寫並渲染**。以上的渲染雖然只有限於鏡頭內可看見的物件，但這不失為一個好方法，一方面是可以節省記憶體，另一方面可以增加遊玩性，讓玩家可以藉由空檔閃躲子彈。最後，我認為我們的專題尚未達到可發揮的上限，我們的優勢在於遊戲玩家的**動作精細控制與物件之間的互動多樣性**，雖然跳躍關卡有難度，但遊戲玩家地跳躍皆是相當合理的，且遊玩過程中不會有太大的卡頓。若是再花時間實作，新增關卡與遊玩機制，相信能夠變為一款好玩的遊戲。最後**希望評分標準可以早點公布。**

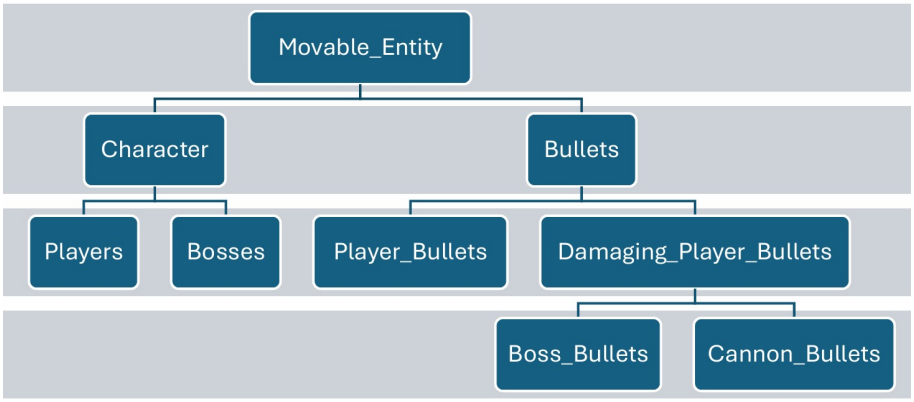
3. 陳毅

看到其它兩位組員強大的程式能力，我感到非常欽佩；再加上這學期的其他課程、系上活動過重，因此我並不對自己的程式表現感到特別滿意。但我對自己的美工設計所花費的時間感到相當滿意。**幾乎所有的圖像皆為我以游標類似手繪的方法生成**，像是螢幕左下方簡單的子彈條，這樣有漸層的樣式，必須每個 pixel，從色盤上**慢慢游標點選連續不同的色碼**，並慢慢地用游標在 sheet 上點出來，而人物的設計我也修改過，為了創作出二維但有立體感的圖像，顯現出跑步時的**側面姿態**、與移動時的**動態感**，必須每幀圖片的身體都有些微的 pixel 變化，不只是手部的圖像改變，另外簡單的尖刺與砲彈的方塊，也因圖像很小、像素很少，我們在繳交前大改了方塊設計，嘗試了很多方法才能讓方塊變得像樣，而背後也有許多沒有用到的圖片或圖像、像是打 boss 時的 boss room 背景設計，也必須使用 Picwish 線上去背與背景合成圖片，且去背方法也是用每個小範圍手繪畫出來的，無法大範圍點選，而簡單的背景圖像，也是事先生成多張圖片最後才決定好的，最後我也幫忙了 **ttf, alpha blending, music** 的部分，交給另外兩位組員幫忙合成，並將 music 的 bug 去除。

2 個人貢獻

以下紅字是我認為較為重要且與批改標準有關之處

2.1 Class



2.1.1 Movable_Entity

移動有關物件的 Base Class。

main.cpp line 117-127

```
class MovableEntity {
    public:
        SDL_Rect mBox; // Position and dimensions
        MovableEntity(int x, int y, int w, int h) {
            mBox.x = x;
            mBox.y = y;
            mBox.w = w;
            mBox.h = h;
        }
        SDL_Rect getBox() const { return mBox; }
};
```

- mVelX,mVelY設為protected是為了只讓繼承的結構可直接更改。
- mBox紀錄物件的x,y,w,h(圖片左上 x 座標、圖片左上 y 座標、圖片寬 width、圖片高 height)。設為 Public 是為了在確認是否碰撞 (collisonX(),collsionY()) 時較為方便。

- MovableEntity(int x, int y, int w, int h):main.cpp line 120-127，輸入 mBox 的參數。
- getBox():const 是在方法的末尾加上的，它保證該方法不會修改類別的成員變數。因此 getBox() 不能更動 mBox 或其他類似的屬性。

2.1.2 Character

角色物件的 Base Class。繼承Movable_Entity。
main.cpp line 130-155

```
class Character : public MovableEntity {
    protected:
        int hp;
        const int hp_max;
        bool Alive;
        int GetStatus() const { return Alive;}

    public:
        Character(int x, int y, int w, int h, int initialHealth)
        : MovableEntity(x, y, w, h), hp(initialHealth), hp_max(initialHealth)
        {Alive = true;}
        void DecreaseHp(int d){
            hp -= d;
            if(hp <= 0){
                Alive = false;
                hp = 0;
            }
        }
        void IncreaseHp(int d, int hp_max){
            hp += d;
            hp = min(hp,hp_max);
        }
        float GetHpPercentage(){
            return (float)hp/hp_max;
        }
};
```

- hp, hp_max：角色的血量與最大血量，hp_max 設為 const int 是為防止其被任何函數改變，在初始化時兩者設為相同。如此設計是因為遊戲中**只要是角色皆有血量，否則不會死亡**。

- Alive：角色是否存活。存活為true。
- GetStatus(): 型態為 const，保證函式不會變動到 Alive。
- Character(int x, int y, int w, int h, int initialHealth):main.cpp line 138-140，前四個參數為物件 MovableEntity 的初始化參數，且將 hp, hp_max 初始化為 initialHealth。注意到 hp_max 的型別為 const static int 所以必須用 member initialization 做初始化。
- DecreaseHp(),IncreaseHp()：角色增加血量與減少血量。血不大於 0 時，Alive設為false。
- GetHpPercentage()：計算hp/hp_max。

2.1.3 Players

玩家的 class。繼承 Character。
main.cpp line 300-385

```
class Players: public Character{
    public:
        //The dimensions of the player
        int Easter_Eggs_Total = 0;
        int Bullet_Total = 0;

        static const int Player_WIDTH = PLAYER_WIDTH;
        static const int Player_HEIGHT = PLAYER_HEIGHT;

        //Initializes the variables
        Players(int x, int y, int initialHealth)
        :Character(x, y, Player_WIDTH, Player_HEIGHT, initialHealth){
            Step_Run = Step_Run_Basic;
        }
        void setCamera( SDL_Rect&);

        bool EnterBossArea(Bosses& Boss){
            if(mBox.y + Player_HEIGHT <= Boss.Area_BottomY)
                return true;
            return false;
        }
        void Render(SDL_Rect& camera, Tile**& tile, KeyPressSurfaces& Key);
```

```
        void Jump(Tile**&);

        void Fall(Tile**&);

        void handleEvent(Tile**&, KeyPressSurfaces&);

        void FullRender(vector<Bullets*>&, Bosses&, Tile**&, SDL_Rect&,
            clock_t now, KeyPressSurfaces&);

        bool GetStatus(){
            return Character::GetStatus();
        }

        void IncreaseHp(int d, int hp){
            Character::IncreaseHp(d,hp);
        }

        float GetHpPercentage(){
            return Character::GetHpPercentage();
        }

    private:
    // ======================== ACTION PARAMETER
        ================================
int Vertical = 0, Horizontal = 0, cnt_Horizontal = 0;
int Step_Run_Basic = 4, Step_Run_Tick = 8, Step_Run_Acceleration = 1,
   Step_Run_Acceleration_Tick = 5, Step_Run_Acceleration_Max = 4;
int Step_Run;
int Step_Horizontal_Tick = 4;
bool Left = false, Right = false;
int Step_Jump_Basic = 8, Step_Jump_Tick_1 = 10, Step_Jump_Tick_2 = 20,
   Step_Jump_Delay = 16;
int Step_Jump_Cnt = 0;
int Step_Vertical = 0;
bool Jumped = false, Falled = false, FreeFall = false;
int accelCnt = 0;
int Fall_Natural_Vel = 4;
// ============================================================================

        int Bullet_Cnt = Bullet_Cnt_Max, Bullet_Add_Cnt = 0,
```

```
        Bullet_Add_Cnt_Max = 50;
    int Bullet_Shoot_Interval = 100;


    void Accelerator(char dir) { // dir: 'L', 'R'
        if (!Jumped && !Falled) {
            cnt_Horizontal++;
            if (cnt_Horizontal >= Step_Horizontal_Tick) {
                Horizontal = ++Horizontal % 4;
                cnt_Horizontal = 0;
            }
            accelCnt++;
            if (((dir == 'L') && Left) || ((dir == 'R') && Right)) { //
                同一方向
                if (Step_Run < Step_Run_Basic + Step_Run_Acceleration_Max) {
                    if (accelCnt >= Step_Run_Acceleration_Tick) {
                        Step_Run += Step_Run_Acceleration;
                        accelCnt = 0; // 重置加速計數器
                    }
                }
            }
        }
        if (((dir == 'L') && Right) || ((dir == 'R') && Left)) { //
            角色在空中時，方向相反
            accelCnt = 0;
            cnt_Horizontal = 0;
            Step_Run = Step_Run_Basic;
        }
    }
};
```

- Easter_Eggs_Total：獲得的彩蛋數量。型態為 public，遊戲結束顯示彩蛋收集數時可以直接存取數值。
- Player_WIDTH, Player_HEIGHT：角色的寬與高，型態為 static const int，防止重新定義並確認每次呼叫時都為相同的值。
- Players(int x, int y, int initialHealth): 遊戲角色的初始化，將參數代入 Character(x, y, Player_WIDTH, Player_HEIGHT, initialHealth)，並令 Step_Run = Step_Run_Basic。

- ACTION PARAMETER：角色移動跳躍的參數，見 FullRender 的說明。
- Bullet_Cnt, Bullet_Add_Cnt, Bullet_Add_Cnt_Max, Bullet_Shoot_Interval：子彈剩餘數量、子彈填充計數器、子彈填充計數器上限 (達到上限後加一發子彈)、子彈射擊間隔 (以 tick 為單位)。
- Accelerator()：若連續兩個 tick 都按著左 (右) 鍵，則會加速。否則角色會轉向，且速度會重至基本速度 Step_Run_Basic。其中利用 Left,Right 記錄上個 tick 角色的移動方向，再利用輸入的 char 參數來判定是否與上次相同。若是，速度 Step_Run 加 Step_Run_Acceleration 而若增加的速度大於最大增加速度 (Step_Run > Step_Run_Basic + Step_Run_Acceleration_Max) 則不會再增加速度。

2.1.4 Bosses

main.cpp line 235-298

```
class Bosses: public Character{
  static const int Boss_WIDTH = BOSS_WIDTH;
  static const int Boss_HEIGHT = BOSS_HEIGHT;
  static const int Boss_Action_Tick = 400;
  int Boss_Action_Mode;
  static const int Boss_Action_Mode_Max = 2;
  static const int Boss_Mode_0_Interval = 20;
  static const int Boss_Mode_1_Speed = 5;
  static const int Boss_Mode_1_Interval = 20;

 public:
  int Area_BottomY = 640;
  int Stage_Cnt = 0;
  void Render(Players& player, SDL_Rect &);
  Bosses(int x, int y, int
     initialHealth):Character(x,y,Boss_WIDTH,Boss_HEIGHT,initialHealth){
   Boss_Action_Mode = 0;
  }
  int ModeRand(){
   return (rand()%20)-10;
  }
  void Boss_Mode_0(){
   static int cnt = 0;
   cnt++;
   //cout << "Mode0\n";
```

```
 if(cnt == Boss_Mode_0_Interval){
  Bullet.push_back(new Boss_Bullets(mBox.x,mBox.y,ModeRand(),ModeRand(),0));
  Bullet.push_back(new Boss_Bullets(mBox.x,mBox.y,ModeRand(),ModeRand(),0));
  cnt = 0;
 }
}
void Boss_Mode_1(Players& player);

void Action(Players& player){
 static int cnt = 0;
 static bool rest = true;
 cnt++;
 if(cnt >= Boss_Action_Tick/2)
  rest = false;
 if (cnt % Boss_Action_Tick == 0) {
  Boss_Action_Mode++;
  cnt = 0;
  Boss_Action_Mode %= Boss_Action_Mode_Max;
  rest = true;
 }
 if(!rest){
  switch(Boss_Action_Mode){
   case 0:
    Boss_Mode_0();
    break;
   case 1:
    Boss_Mode_1(player);
    break;
  }
 }
}
void DecreaseHp(int d){
 Character::DecreaseHp(d);
}
float GetHpPercentage(){
 return Character::GetHpPercentage();
}
bool GetStatus(){
 return Character::GetStatus();
}
```

```
};
```

main.cpp line 387-399

```
void Bosses::Boss_Mode_1(Players& player){
    static int cnt = 0;
    cnt++;
    if(cnt == Boss_Mode_1_Interval){
        int x = player.mBox.x - mBox.x;
        int y = player.mBox.y - mBox.y;
        int vx = round(double(Boss_Mode_1_Speed*x/sqrt(x*x+y*y)));
        int vy = round(double(Boss_Mode_1_Speed*y/sqrt(x*x+y*y)));
        Bullet.push_back(new Boss_Bullets(mBox.x - BULLET_WIDTH,mBox.y,vx,vy,0));
        Bullet.push_back(new Boss_Bullets(mBox.x + mBox.w ,mBox.y,vx,vy,0));
        cnt = 0;
    }
}
```

- Boss_WIDTH, Boss_HEIGHT：Boss 的寬與高，型態為 static const int，防止重新定義並確認每次呼叫時都為相同的值。
- Boss_Action_Tick, Boss_Action_Mode, Boss_Action_Mode_Max：Boss 模式的變化間隔、參數、參數最大值。前後兩者皆設定為定值，型態設為 static const int。
- Boss_Mode_0_Interval, Boss_Mode_1_Interval, Boss_Mode_1_Speed：Boss 模式的各種參數。由於參數皆設定為定值，型態設為 static const int。
- ModeRand():Boss_Mode_0() 的隨機函數。設定子彈速度絕對值不超過 10。
- Boss_Mode_0()：隨機方向速度發射子彈。使用 cnt 計數 (使用 static 型別 Boss_Mode_0() 結束後 cnt 不會被刪除)，當 cnt = Boss_Mode_0_Interval 時才會發射，以免子彈發射過多，玩家無法躲避。
- Boss_Mode_1()：朝玩家方向發射子彈。cnt 的計數功能同上。
- Action:Boss 切換模式並呼叫函數 Boss_Mode_0()、Boss_Mode_1()。cnt 的計數功能同上。rest 的功能為使得 Boss 只有在後半的時間能發射子彈 (初始化為 false)。當 cnt 大於 Boss_Action_Tick/2 時，rest = true。 當 cnt 大於 Boss_Action_Tick 時，Boss_Action_Mode 會加一，再取其除以 Boss_Action_Mode_Max 的餘數以確保不會大於總模式數，且 rest = false。
- DecreaseHp()、GetHpPercentage()、GetStatus(): 繼承至 Character 的對應函數。

2.1.5 Bullets

main.cpp line 157-176

```
class Bullets:public MovableEntity{
    protected:
        const int Bullets_Damage;
    public:
        int velX = 0;
        int velY = 0;
        const double angle = 0;
        Bullets(int a, int b, int velx, int vely,int
            BULLET_DAMAGE):MovableEntity(a, b, BULLET_WIDTH,
            BULLET_HEIGHT),velX(velx),velY(vely),Bullets_Damage(BULLET_DAMAGE),angle((double)180
        virtual void Update(){
   mBox.x += velX;
   mBox.y += velY;
  };
        virtual void Render(SDL_Rect&){};
        virtual bool DetectPlayerCollsion(Players&){
            return false;
        };
        virtual bool DetectBossCollision(Bosses&){
            return false;
        };
};
```

- Bullets_Damage: 子彈射擊的傷害。設為 protected 是為了只讓繼承的結構可直接更改。
- velX,velY: 子彈在 x, y 方向上的速度。
- angle: 子彈傾斜的角度。繪製的子彈圖片為向左，因此在渲染時需要旋轉角度。型態為 const，因為在子彈確認初始速度後，方向不會改變。**使用到 cmath 函式庫的 atan2(velX,velY) 函數 (結果為 $\tan^{-1}(\text{velY}/\text{velX})$) 來計算反正切值，以防止會產生 mVelX = 0 而無法計算的情況。**
- Bullet_WIDTH, Bullet_HEIGHT：子彈的寬與高，型態為 static const int，防止重新定義並確認每次呼叫時都為相同的值。
- Update(): 更新子彈位置的函數。x 座標增加 velX, y 座標增加 velY。
- Render(): 渲染子彈的函數，作為多形函數。

- DetectPlayerCollision(), DetectBossCollision(): 偵測子彈使否與角色碰撞的函數，作為多形函數。由於子彈型別的設計，各種 Derived Class 對應的多形函數皆不同。

2.1.6 Player_Bullets

main.cpp line 223-231

```
class Player_Bullets:public Bullets{
    public:
        Player_Bullets(int x,int y,int
            velx):Bullets(x,y,velx,0,PLAYER_BULLET_DAMAGE){}
        virtual void Render (SDL_Rect& camera) override{
            gKeyPressSurfaces[KEY_PRESS_SURFACE_BULLET_RIGHT].render(mBox.x -
                camera.x, mBox.y - camera.y,NULL,angle);
        };
        virtual bool DetectBossCollision(Bosses&) override;

};
```

Bullet_Character_Collision.hpp line 9-15

```
bool Player_Bullets::DetectBossCollision(Bosses& boss){
    if(checkCollision(mBox,boss.mBox)){
        boss.DecreaseHp(Bullets_Damage);
        return true;
    }
    return false;
}
```

- Render(): 呼叫 LTexture class 的 render 函數，渲染子彈的圖 (原本有設定不同子彈圖案不同，但迫於時間而無法達成，因此 function 的 polymorphism 在此沒實質作用)。

- DetectBossCollision(): 傳入 Boss，以偵測 Boss 是否與子彈碰撞 (重疊)。**注意到由於這個 class 只與玩家的子彈有關，所以只會對 Boss 造成傷害，故 DetectBossCollision() 需改變程式，DetectPlayerCollision() 則否。**

2.1.7 Damaging_Player_Bullets

main.cpp line 178-182

```
class Damaging_Player_Bullets:public Bullets{
    public:
```

```
        Damaging_Player_Bullets(int x, int y,int velX, int
            velY):Bullets(x,y,velX,velY,BOSS_BULLET_DAMAGE){}
        bool DetectPlayerCollsion(Players&) override;
};
```

Bullet_Character_Collision.hpp line 1-7

```
bool Damaging_Player_Bullets::DetectPlayerCollsion(Players& player){
    if(checkCollision(mBox,player.mBox)){
        player.DecreaseHp(Bullets_Damage);
        return true;
    }
    return false;
}
```

- DetectPlayerCollision(): 傳入 Player，以偵測 Player 是否與子彈碰撞 (重疊)。**注意到由於這個 class 只與 Boss 與大砲的子彈有關，所以只會對玩家造成傷害，故 DetectBossCollision() 不需更動，DetectPlayerCollision() 需改變程式。**

2.1.8 Boss_Bullets

main.cpp line 184-190

```
class Boss_Bullets:public Damaging_Player_Bullets{
    public:
        Boss_Bullets(int x, int y,int velX, int
            velY):Damaging_Player_Bullets(x,y,velX,velY){}
        virtual void Render (SDL_Rect& camera) override{
            gKeyPressSurfaces[KEY_PRESS_SURFACE_BULLET_RIGHT].render(mBox.x -
                camera.x, mBox.y - camera.y,NULL,angle);
        }
};
```

- Render(): 同上。

2.1.9 Cannon_Bullets

main.cpp line 192-198

```
class Cannon_Bullets: public Damaging_Player_Bullets{
    public:
        Cannon_Bullets(int x, int y,int velX, int
            velY):Damaging_Player_Bullets(x,y,velX,velY){}
        virtual void Render (SDL_Rect& camera) override{
```

```
            gKeyPressSurfaces[KEY_PRESS_SURFACE_BULLET_RIGHT].render(mBox.x -
                camera.x, mBox.y - camera.y,NULL,angle);
        }
};
```

- Render(): 同上。

Damaging_Player_Bullets,Player_Bullets 的繼承與其多形函數很好的分開敵方我方子彈渲染與角色扣血的問題。(main.cpp line 157-176, 178-182, 184-190, 192-198, 223-231)

2.2 主程式

2.2.1 基本機制

以下說明基本機制與其程式碼。

- 當按下 Enter 鍵後遊戲開始並撥放背景音樂。

```
while(true){
    while( SDL_PollEvent( &e ) != 0 ){
        if( e.type == SDL_QUIT ){
            quit = true;
        }
    }
    const Uint8* BeginningKeyStates = SDL_GetKeyboardState( NULL );
    if(BeginningKeyStates[SDL_SCANCODE_KP_ENTER])
        break;
}
Mix_PlayMusic( gMusic, -1 );
Mix_VolumeMusic(16);
```

- 玩家尚未進入 Boss 房前的控制程式。

```
while(player.GetStatus() && !player.EnterBossArea(boss) && !quit){
    do{
        clock_t now = clock();
        if( e.type == SDL_QUIT ){
            quit = true;
        }
        player.FullRender(Bullet,boss,tileSet,camera,now,Key);
```

```
        while(double(clock() - now) < DELAY){}
    }while( SDL_PollEvent( &e ) != 0 );
    player.handleEvent(tileSet,Key);
}
```

- 玩家尚未進入 Boss 房後的控制程式。更改為 Boss 的背景音樂，並且通往 Boss 房的通道關閉。

```
Mix_HaltMusic();
Mix_PlayMusic( gBossMusic, -1 );

tileSet[169] = new Tile(80, 640, 0);

while(boss.GetStatus() && player.GetStatus() && !quit){
    do{
        clock_t now = clock();
        if( e.type == SDL_QUIT ){
            quit = true;
        }
        boss.Action(player);
        player.FullRender(Bullet,boss,tileSet,camera,now,Key);
        while(double(clock() - now) < DELAY){}
    }while( SDL_PollEvent( &e ) != 0 );
    player.handleEvent(tileSet,Key);
}
```

- 結束畫面顯示 (玩家死亡/Boss 死亡)。

```
SDL_RenderClear( gRenderer );
gBackgroundBlurred.render(- camera.x, - camera.y);
if(!boss.GetStatus()){
    // Boss Died
    ShowThxforplaying.render( ( SCREEN_WIDTH - ShowThxforplaying.getWidth()
        ) / 2, ( SCREEN_HEIGHT - ShowThxforplaying.getHeight() ) / 2 - 150);
    ShowEasterEggs.render( ( SCREEN_WIDTH - ShowEasterEggs.getWidth() ) / 2,
        ( SCREEN_HEIGHT - ShowEasterEggs.getHeight() ) / 2 );
    ShowBulletcalculates.render( ( SCREEN_WIDTH -
        ShowBulletcalculates.getWidth() ) / 2, ( SCREEN_HEIGHT -
        ShowBulletcalculates.getHeight() ) / 2 + 150);
```

```
        SDL_RenderPresent( gRenderer );

    }
    else if(!player.GetStatus()){
        // Player Died
        ShowPlayerDie.render( ( SCREEN_WIDTH - ShowPlayerDie.getWidth() ) / 2, (
            SCREEN_HEIGHT - ShowPlayerDie.getHeight() ) / 2 );
        SDL_RenderPresent( gRenderer );
    }
    SDL_Delay(5000);
    Mix_HaltMusic();
```

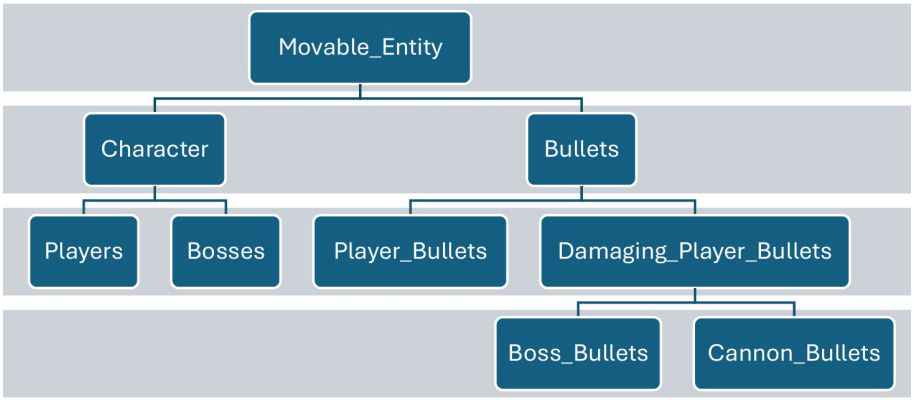
以上主程式包括但不限於有關的角色程式為我實作的。

2.2.2 組裝關係

TileSet 包含 Tile** 之成員變數 (Global_Init.hpp line 208-216)。
在 TileSet 使用組裝可以更方便我們呼叫 TileSet 去更改 Tile**，且可以避免外在函式更變 Tile**。

2.2.3 繼承關係

我實作的程式之繼承關係如下



2.3 解構子

TileSet 的解構子：刪除 tile** 的內容。
main.cpp line 958-970

```
TileSet::~TileSet(){
 for( int i = 0; i < TOTAL_TILES[lvid]; ++i )
```

```
 {
   if( tiles[ i ] != NULL )
   {
   delete [] tiles[ i ];
   tiles[ i ] = NULL;
   }
 }
 delete []tiles;
}
```

2.3.1 static member

見 2.1.4 Boss 的 Action() 中的 cnt、rest (main.cpp line 267-268) 與 Boss_Mode_1 中的 cnt(main.cpp line 388)。

2.3.2 const member

1. main.cpp line 1035 - 1044

```
while(true){
    while( SDL_PollEvent( &e ) != 0 ){
        if( e.type == SDL_QUIT ){
            quit = true;
        }
    }
    const Uint8* BeginningKeyStates = SDL_GetKeyboardState( NULL );
    if(BeginningKeyStates[SDL_SCANCODE_KP_ENTER])
        break;
}
```

   BeginningKeyStates 設為 const Uint8* 的作用是防止在程式跑動過程中 SDL_GetKeyboardState 讀值改變而造成 Bug。

2. 各種 Character 的 Width 與 Height(main.cpp line 236-237 306-307)。
3. 2.1.1 Movable_Entity 的 GetBox()(main.cpp line 126)。
4. 2.1.2 Character 的 hp_max, GetStatus()(main.cpp line 133 135)。
5. 2.1.5 Bullets 的 Bullet_Damage、angle(main.cpp line 159 163)。
6. 2.1.4 Boss 的 Boss_Mode_0_Interval, Boss_Mode_1_Interval, Boss_Mode_1_Speed ,Boss_Mode_0_Interval, Boss_Mode_1_Speed(main.cpp line 236-238 240 - 243)。

2.3.3 Polymorphism

1. Character_FullRender.hpp line 15-34

```
void RenderUpdateBullets(Players& player, Bosses& boss, vector<Bullets*>&
    Bullet, Tile**& tile, SDL_Rect& camera){
 for (vector<Bullets*>::iterator it = Bullet.begin(); it != Bullet.end();) {
       (*it)->Update();
       if (BulletsCollisionY((*it)->mBox, tile) ||
           BulletsCollisionX((*it)->mBox, tile)) {
           it = Bullet.erase(it);
       }
  else{
           if ((*it)->DetectPlayerCollsion(player)) {
               it = Bullet.erase(it);
               continue;
           }
   if ((*it)->DetectBossCollision(boss)) {
               it = Bullet.erase(it);
               continue;
           }
           (*it)->Render(camera);
           ++it;
  }
    }
}
```

在 Bullet 與其 Derived class 中有撰寫 Render() 的 polymorphism，各自都有不同的渲染。而 DetectPlayerCollsion(),DetectBossCollision() 可見2.1.5,2.1.6,2.1.7的說明。

2. Character_FullRender.hpp line 78-81

```
for (int i = 0; i < id.size(); ++i){
    if(tile[id[i]]->getType()!=5)
        tile[id[i]]->render( camera );
}
```

在 tile 已實作各種 render 的 polymorphism，而我們刻意用指標去儲存方塊 tile，因此可利用指標的處理使不同方塊有不同的渲染方式。

2.4 特別技巧或巧思

2.4.1 渲染優化

Character_FullRender.hpp line 76-77

```
vector<int> id;
FindSurroundingTileID(id,camera);
```

其中 FindSurroundingTileID() 則是傳入一個 vector 與鏡頭 camera，再將鏡頭內可以看到的方塊 tile 的 id 傳回 vector。可以減少渲染圖片的量，讓畫面不會卡。
在此部分速度差異約為 $651/96 - 1 \approx 5.78$ 倍，具體幀率差異則未測量。
以上的渲染雖然只有限於鏡頭內可看見的物件，但這不失為一個好方法，一方面是可以節省記憶體，另一方面可以增加遊玩性，讓玩家可以藉由空檔閃躲子彈。

2.4.2 handleEvent(tileSet,Key)

handleEvent(tileSet,Key)：設定參數並傳給 Player 的 Render() 做處理。

1. 我們不是使用 SDL 中的 SDL_keysym 功能，而是採用，SDL_GetKeyboardState 否則持續按著按鍵系統會無法正確偵測。

```
if (gCurrentKeyStates[SDL_SCANCODE_SPACE]) {
    clock_t now = clock();
    double elapsed_ms = double(now - SpaceTime) * 1000.0 / CLOCKS_PER_SEC;
    if (elapsed_ms >= Bullet_Shoot_Interval) {
        if(Bullet_Cnt >= 1){
            if (Left)
                Bullet.push_back(new Player_Bullets(mBox.x - BULLET_WIDTH,
                    mBox.y + Player_HEIGHT/2 - BULLET_HEIGHT/2, -12));
            if (Right)
                Bullet.push_back(new Player_Bullets(mBox.x + Player_WIDTH,
                    mBox.y + Player_HEIGHT/2 - BULLET_HEIGHT/2, 12));
            Bullet_Total++;
            SpaceTime = now;
            Bullet_Cnt--;
            Mix_PlayChannel( -1, gShoot, 0 );
        }
    }
}
```

2. 使用設定的跑步與跳躍參數控制 Player 的動作，並且讓其能夠滿足物理規則。

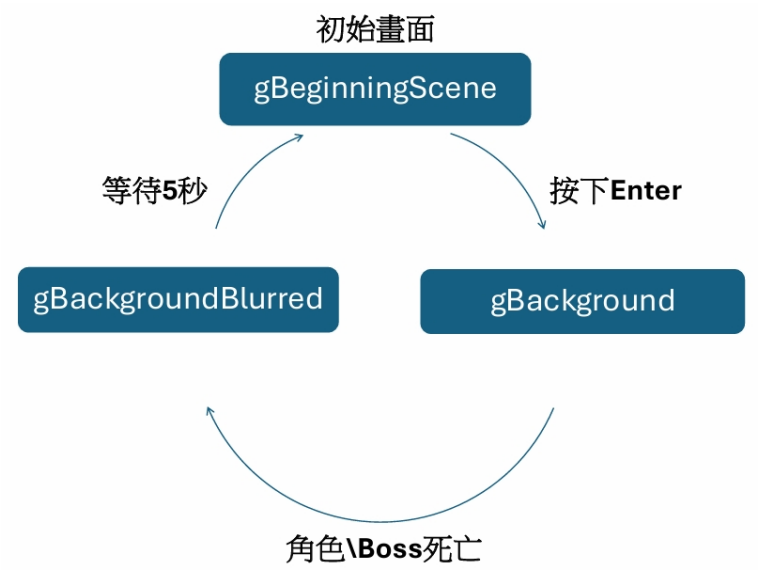
(a) 在地面時才能夠加速，在空中時則否。

(b) 在空中時可以轉換方向移動。

雖然 (b) 不符合物理，但可增加遊玩性。

2.4.3 畫面設計

1. 背景
   設定三個 LTexture 關係如下



2. 角色
   Player 的圖片由陣列儲存 (main.cpp line 570-669)。因此在角色渲染時只要 handleEvent 傳值給 Key(main.cpp line 1056)，FullRender 再讀 Key 之值即可 (main.cpp line 1053)。也就是 gPlayerTexture = gKeyPressSurfaces[Key];。

3. 子彈
   判斷方向與角度進行渲染 (main.cpp line 188 196 227)。

2.4.4 聲音

1. gMusic：背景音樂，利用可播放循環的函數 Mix_PlayMusic() 並使用 Mix_VolumeMusic() 調整聲音大小。Mix_HaltMusic() 終止音樂。

2. gChest, gShoot: 開寶箱、射子彈聲音，為了可以同時支持播放多個音效，利用 Mix_PlayChannel() 實作。