Questions

A. **Easy Questions** (pick 2)

1. Explain the concept of “Encapsulation”. Show a short code snippet to illustrate why “Encapsulation” is useful and what might happen if we do not use it.

封裝性就類似資料夾，在OOP C++裡面其實就是class的概念。有一些function只有特定variable會用到，就把這些variable和function包裝在一起，外人如果沒有存取這個資料夾的權限(或只有部分權限private、friend、protected…)，就不會輕易把東西外放給外人知道(data hiding)，這也強化了一些可能不小心打錯function名稱之類的保護。而最大的好處就是依照不同的資料夾的內容之間的關聯性，可以分門別類。Ex: 會計部門、金融部門、銷售部門，雖然都是管錢，但有不同的功用。

簡單的例子就是ChiBullet、MissileBullet、FireBullet、LaserBullet彼此都是不同資料夾，所以我如果想要對ChiBullet做處理，我根本不用擔心會不小心弄錯處理對象成MissileBullet，我名稱就統一叫做Bullet。

那如果不用Encapsulation的話，各個Bullet打結在一起，不僅頭昏眼花(因為每個bullet都要有不同名稱，如果用Encapsulation的話就因為資料夾不同，所以撞名也沒差)，也難寫code，更難debug。

Image-class:

```
class Image : public IObject {
public:
    // Smart pointer to bitmap.
    std::shared_ptr<ALLEGRO_BITMAP> bmp;
    /// <summary>
    /// Construct a image object.
    /// </summary>
    /// <param name="img">The image path in 'resources/images/' </param>
    /// <param name="x">X-coordinate </param>
    /// <param name="y">Y-coordinate </param>
    /// <param name="w">Width of the image, 0 indicates original size.</param>
    /// <param name="h">Height of the image, 0 indicates original size.</param>
    /// <param name="anchorX">The centerX of the object. (0, 0) means top-left, while (1, 0) means top-right.</param>
    /// <param name="anchorY">The centerY of the object. (0, 1) means bottom-left, while (1, 1) means bottom-right.</param>
    explicit Image(std::string img, float x, float y, float w = 0, float h = 0, float anchorX = 0, float anchorY = 0);
    /// <summary>
    /// Draw the loaded image.
    /// </summary>
    void Draw() const override;
    /// <summary>
    /// Return bitmap width.
    /// </summary>
    /// <returns>Width of the original or scaled bitmap.</returns>
    int GetBitmapWidth() const;
    /// <summary>
    /// Return bitmap height.
    /// </summary>
    /// <returns>Height of the original or scaled bitmap.</returns>
    int GetBitmapHeight() const;
};
```

IObject-class:

```
class IObject {
    friend class Group;
protected:
    // The iterator of objects linked-list when added to scene.
    // Can make removing objects faster.
    // Reference: Iterator, which is also a Design Pattern when implementing.
    std::list<std::pair<bool, IObject*>>::iterator objectIterator{};
    /// <summary>
    /// The interface cannot be instantiated directly, must be inherited.
    /// </summary>
    explicit IObject() = default;
    /// <summary>
    /// <summary>
    /// Construct an IObject.
    /// </summary>
    /// <param name="x">X-coordinate.</param>
    /// <param name="y">Y-coordinate.</param>
    /// <param name="w">Width of the image. 0 indicates original size.</param>
    /// <param name="h">Height of the image. 0 indicates original size.</param>
    /// <param name="anchorX">The centerX of the object. (0, 0) means top-left, while (1, 0) means top-right.</param>
    /// <param name="anchorY">The centerY of the object. (0, 1) means bottom-left, while (1, 1) means bottom-right.</param>
    explicit IObject(float x, float y, float w = 0, float h = 0, float anchorX = 0, float anchorY = 0);
public:
    // Determines whether this object should be drawn and updated.
    bool Visible = true;
    // The object's position, center depends on Anchor.
    Point Position;
    // The object's size.
    Point Size;
    // The center of the object. (0, 0) means top-left, (1, 1) means bottom-right.
    Point Anchor;
    /// <summary>
    /// The default virtual destructor to support polymorphism destruction.
    /// </summary>
    virtual ~IObject() = default;
    /// <summary>
    /// Copy constructor does not add a new instance to scene, they are still the same object.
    /// </summary>
    IObject(const IObject& other) = default;
    /// <summary>
    /// Copy assignment operator does not add a new instance to scene, they are still the same object.
    /// </summary>
    IObject& operator=(IObject const&) = default;
    /// <summary>
    /// Retrieve the object iterator for later insertion.
```

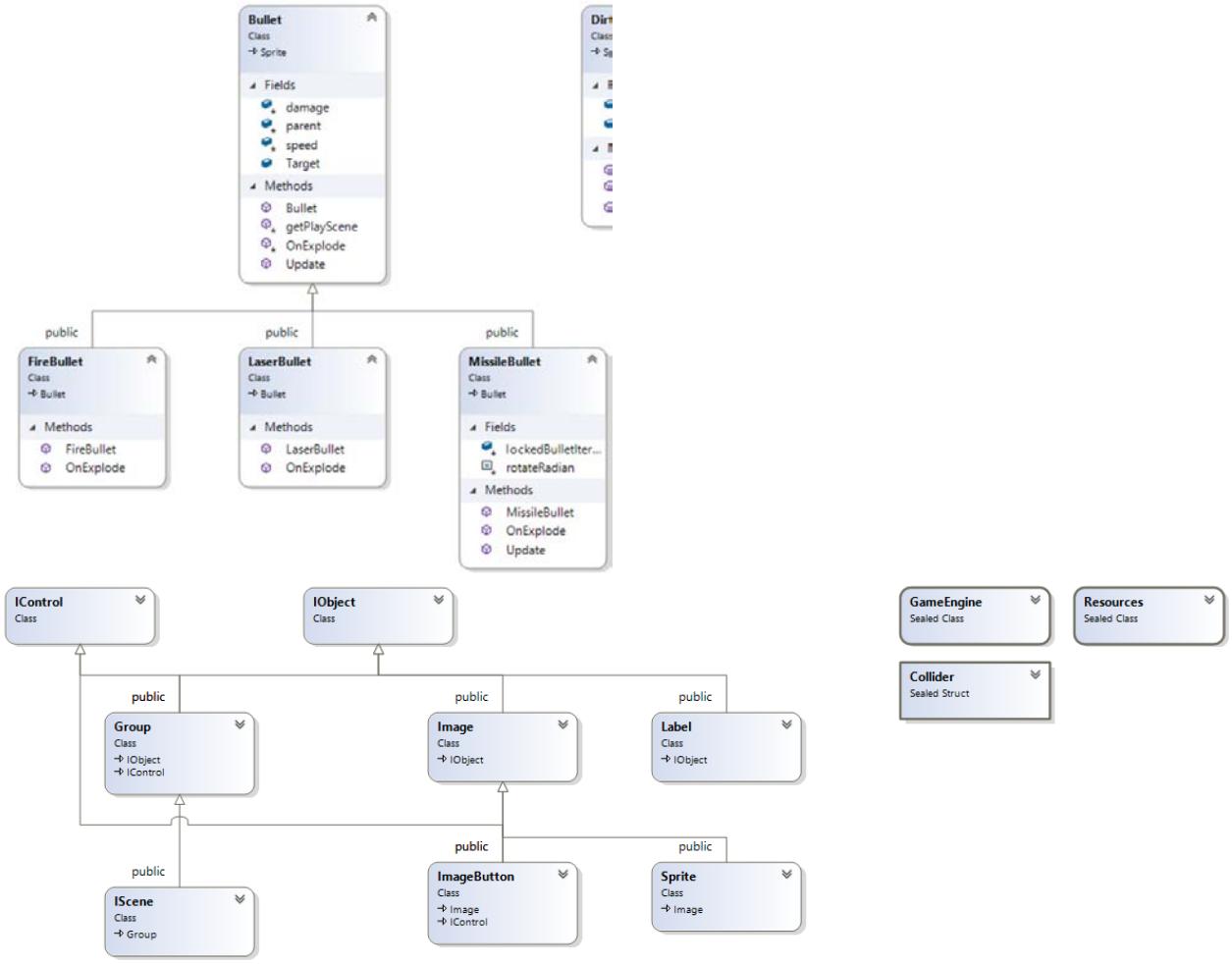
```
Image::Image(std::string img, float x, float y, float w, float h, float anchorX, float anchorY) :
    IObject(x:x, y:y, w, h, anchorX:anchorX, anchorY:anchorY) {
    if (Size.x == 0 && Size.y == 0) {
        bmp = Resources::GetInstance().GetBitmap(img);
        Size.x = GetBitmapWidth();
        Size.y = GetBitmapHeight();
    }
    else if (Size.x == 0) {
        bmp = Resources::GetInstance().GetBitmap(img);
        Size.x = GetBitmapWidth() * Size.y / GetBitmapHeight();
    }
    else if (Size.y == 0) {
        bmp = Resources::GetInstance().GetBitmap(img);
        Size.y = GetBitmapHeight() * Size.x / GetBitmapWidth();
    }
    else /* Size.x != 0 && Size.y != 0 */ {
        bmp = Resources::GetInstance().GetBitmap(img, width:Size.x, height:Size.y);
    }
}
void Image::Draw() const {
    al_draw_scaled_bitmap(bmp.get(), sx:0, sy:0, sw:GetBitmapWidth(), sh:GetBitmapHeight(),
        dx:Position.x - Anchor.x * GetBitmapWidth(), dy:Position.y - Anchor.y * GetBitmapHeight(),
        dw:Size.x, dh:Size.y, flags:0);
}
int Image::GetBitmapWidth() const {
    return al_get_bitmap_width(bmp.get());
}
int Image::GetBitmapHeight() const {
    return al_get_bitmap_height(bmp.get());
}
namespace Engine
```

2. Explain the concept of “Inheritance”. Show a short code snippet to illustrate why “Inheritance” is useful and what might happen if we do not use it.

Inheritance 繼承(**Is-a**)。科學都是站在巨人的肩膀上，程式也是。牛頓的F=ma和Ek=0.5mV^2在古典力學也許夠用，但是如果研究越做越多，東西越來越龐大，就不再負荷得了，所以愛因斯坦的E=mc^2就繼承了所有古典力學的所有特性，並且把一些古典力學不能解釋或運作計算的東西給支撐起來。

繼承在C++裡面是種累積越來越多的概念(不會有現實中敗家子通通弄不見的情況)。

舉例來說，MissileBullet是種Bullet，所以他要繼承Bullet的特性，要知道敵人在哪，如果撞到敵人，他應該符合實體的物理規則消失。　而Bullet又繼承了Sprite的特性，因為Sprite是有速度、有角度、會移動的圖ge，Image就要有pointer來儲存本地的圖檔在哪。而Image又繼承IObject，因為只要是物件(不管是文字、圖片、….)，就要時時更新。



```cpp
MissileBullet::MissileBullet(Engine::Point position, Engine::Point forwardDirection, float rotation, Turret* parent) :
    Bullet(img:"play/bullet-3.png", speed:100, damage:4, position, forwardDirection, rotation:rotation + ALLEGRO_PI / 2, parent, type:MISSILE) {
}
void MissileBullet::Update(float deltaTime) {
    if (!Target) {
        float minDistance = INFINITY;
        Enemy* enemy = nullptr;
        for (auto& it:IObject*& : getPlayScene()->EnemyGroup->GetObjects()) {
            Enemy* e = dynamic_cast<Enemy*>(it);
            float distance = (e->Position - Position).Magnitude();
            if (distance < minDistance) {
                minDistance = distance;
                enemy = e;
            }
        }
        if (!enemy) {
            Bullet::Update(deltaTime);
            return;
        }
        Target = enemy;//Target 是Enemy*。
        Target->lockedBullets.push_back(_Val:this);//被這個"missile"bullet新鎖定的enemy，也記錄一下自己被這顆bullet鎖定了
        lockedBulletIterator = std::prev(_First:Target->lockedBullets.end());
        //因為Target->lockedBullets是std::list<Bullets*>所以Target->lockedBullets.end()是Bullets*，故lockedBulletIterator也是Bullets*
        //std::prev可參考https://blog.csdn.net/u013630349/article/details/47105319
        //std::prev(..., a)相當於std::advance(..., -a)。advance是往前走a個iterator(往前是指類似於往陣列idx大的方向)，那當然如果是負數的話，就類似於向量方向。
        //所以std::prev(Target->lockedBullets.end());相當於找出Tagrget->lockedBullets.end()!!!注意!!!end()是指整個有效範圍「後」一個iterator!!!
        //因此才需要std::prev，往前一格才能回到有效位置。
        //lockedBulletIterator是個在MissileBullet.hpp裡面的std::list<Bullet*>::iterator lockedBulletIterator，記錄著每個已經鎖定好enemy目標的bullet。
    }
    Engine::Point originVelocity = Velocity.Normalize();
    Engine::Point targetVelocity = (Target->Position - Position).Normalize();
    float maxRotateRadian = rotateRadian * deltaTime;
    float cosTheta = originVelocity.Dot(targetVelocity);
    // Might have floating-point precision error.
    if (cosTheta > 1) cosTheta = 1;
    else if (cosTheta < -1) cosTheta = -1;
    float radian = acos(cosTheta);
    if (abs(radian) <= maxRotateRadian)
        Velocity = targetVelocity;
    else
        Velocity = ((abs(radian) - maxRotateRadian) * originVelocity + maxRotateRadian * targetVelocity) / radian;
```

```cpp
PlayScene* Bullet::getPlayScene() {
    return dynamic_cast<PlayScene*>(Engine::GameEngine::GetInstance().GetActiveScene());
}
void Bullet::OnExplode(Enemy* enemy) {
}
Bullet::Bullet(std::string img, float speed, float damage, Engine::Point position, Engine::Point forwardDirection, float rotation, Turret* parent, kind_of_bulle
    Sprite(img, x:position.x, y:position.y), speed(speed), damage(damage), parent(parent) {
    Velocity = forwardDirection.Normalize() * speed;
    Rotation = rotation;
    CollisionRadius = 4;
    this->type = type;
}
void Bullet::Update(float deltaTime) {
    Sprite::Update(deltaTime);
    PlayScene* scene = getPlayScene();
    // Can be improved by Spatial Hash, Quad Tree, ...
    // However simply loop through all enemies is enough for this program.
    for (auto& it:IObject*& : scene->EnemyGroup->GetObjects()) {
        Enemy* enemy = dynamic_cast<Enemy*>(it);
        if (!enemy->Visible)
            continue;
        if (Engine::Collider::IsCircleOverlap(c1:Position, r1:CollisionRadius, c2:enemy->Position, r2:enemy->CollisionRadius)) {
            OnExplode(enemy);

            if(enemy->CollisionRadius==15){//15是TakoEnemy的CollisionRadius
                this->Velocity.x = -this->Velocity.x;
                this->Velocity.y = -this->Velocity.y;
                this->Rotation += ALLEGRO_PI;
                enemy->Hit(damage:damage/5);//如果是Tako，就扣正常血量的五分之一，所以
                return;
            }


            enemy->Hit(damage);//如果不是Tako，就扣正常血量
            if (type != CHI)getPlayScene()->BulletGroup->RemoveObject(objectIterator);
            return;
        }
    }
    // Check if out of boundary.
    if (!Engine::Collider::IsRectOverlap(rect1p1:Position - Size / 2, rect1p2:Position + Size / 2, rect2p1:Engine::Point(x:0, y:0), rect2p2:PlayScene::GetClientSize()))
        getPlayScene()->BulletGroup->RemoveObject(objectIterator);
```

```cpp
namespace Engine {
    Sprite::Sprite(std::string img, float x, float y, float w, float h, float anchorX, float anchorY,
        float rotation, float vx, float vy, unsigned char r, unsigned char g, unsigned char b, unsigned char a) :
        Image(img, x:x, y:y, w, h, anchorX:anchorX, anchorY:anchorY), Rotation(rotation), Velocity(Point(x:vx, y:vy)), Tint(al_map_rgba(r, g, b, a)) {
    }
    void Sprite::Draw() const {
        al_draw_tinted_scaled_rotated_bitmap(bmp.get(), Tint, cx:Anchor.x * GetBitmapWidth(), cy:Anchor.y * GetBitmapHeight(),
            dx:Position.x, dy:Position.y, xscale:Size.x / GetBitmapWidth(), yscale:Size.y / GetBitmapHeight(), angle:Rotation, flags:0);
    }
    void Sprite::Update(float deltaTime) {
        Position.x += Velocity.x * deltaTime;
        Position.y += Velocity.y * deltaTime;
    }
}namespace Engine;
```

```cpp
namespace Engine {
    Image::Image(std::string img, float x, float y, float w, float h, float anchorX, float anchorY) :
        IObject(x:x, y:y, w, h, anchorX:anchorX, anchorY:anchorY) {
        if (Size.x == 0 && Size.y == 0) {
            bmp = Resources::GetInstance().GetBitmap(img);
            Size.x = GetBitmapWidth();
            Size.y = GetBitmapHeight();
        }
        else if (Size.x == 0) {
            bmp = Resources::GetInstance().GetBitmap(img);
            Size.x = GetBitmapWidth() * Size.y / GetBitmapHeight();
        }
        else if (Size.y == 0) {
            bmp = Resources::GetInstance().GetBitmap(img);
            Size.y = GetBitmapHeight() * Size.x / GetBitmapWidth();
        }
        else /* Size.x != 0 && Size.y != 0 */ {
            bmp = Resources::GetInstance().GetBitmap(img, width:Size.x, height:Size.y);
        }
    }
    void Image::Draw() const {
        al_draw_scaled_bitmap(bmp.get(), sx:0, sy:0, sw:GetBitmapWidth(), sh:GetBitmapHeight(),
            dx:Position.x - Anchor.x * GetBitmapWidth(), dy:Position.y - Anchor.y * GetBitmapHeight(),
            dw:Size.x, dh:Size.y, flags:0);
    }
    int Image::GetBitmapWidth() const {
        return al_get_bitmap_width(bmp.get());
    }
    int Image::GetBitmapHeight() const {
        return al_get_bitmap_height(bmp.get());
    }
}namespace Engine

```

```cpp
#include "Point.hpp"
#include "IObject.hpp"

namespace Engine {
    IObject::IObject(float x, float y, float w, float h, float anchorX, float anchorY) :
        Position(Point(x, y)), Size(Point(x:w, y:h)), Anchor(Point(anchorX, anchorY)) {}
    std::list<std::pair<bool, IObject*>>::iterator IObject::GetObjectIterator() const {
        return objectIterator;
    }
    void IObject::Draw() const {}
    void IObject::Update(float deltaTime) {}
}namespace Engine
```

那如果不用繼承的話，想想看上面的MissileBullet，不就MissileBullet這一個Class就要把Bullet、Sprite、Image、IObject的code全部寫進同一個Class裡面。這樣就會很亂，容易多考慮或少考慮某些東西。而且如果要創造一種新的Bullet，真的會哭死。

3. Explain the concept of “Polymorphism”. Show a short code snippet to illustrate why “Polymorphism” is useful and what might happen if we do not use it.

Polymorphism多形。就是多種樣態。

光是種波動，他有波動、能量的特性。而光也有粒子性，所以是光的波粒二象性。同一個東西有很多不同的面相。就像我，在清大是學生，在交大是敵校的學生，在宿舍是齋民、在家裡是兒子、也是哥哥，同時也是堂哥堂弟表哥表弟，對畢業的學校來說是校友。同一個東西，在面對不同人或物的時候，就會表現不同的特性。

所以C++借用這種概念，雖然感覺不太一樣。

同樣都是Turret，因為他們各自有不同的特性(發射不同類型的飛彈)，所以各自繼承了Turret變成ChiTurret、MissileTurret、……。可是這些Turret都有共通點，就是他們都會發射子彈，都有個CreateBullet的function。所以我可以想像成，Turret是個共通的東西，當我是MissileTurret的時候，就展現MissileTurret的特性，發射MissileBullet。當我是ChiTurret的時候，就展現ChiTurret的特性，發射ChiBullet。這種Polymorphism給了我們很大的彈性，可以見招拆招，見鬼滅鬼，來一個打一個，反正我可以變形，所以可以用同一個variable去整合很多個不同種的東西。

多型(Polymorphism)則包含多載(Overloading)(參數不同)和複寫(Overriding)(子類別的function覆蓋父類別的)。

4. What is the difference between Procedural Programming and Object-Oriented Programming? Name at least three advantages of OOP.

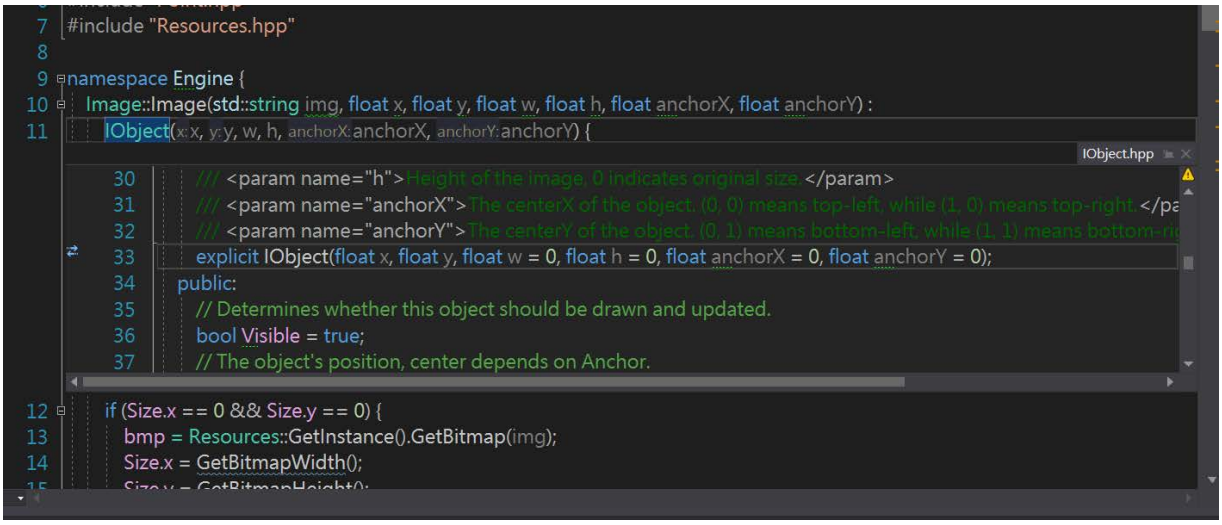
就是比較C和C++。三大不同就是Encapsulation、Inheritance、Polymorphism。

優點: 把程式結構化(分門別類)、增加可讀性和擴展性，增強彈性，並且強化各個variable的安全性。

5. In what condition should we call the base class function from a derived class function? In what condition can we omit the call? Elaborate them on the constructor, destructor, and other general overridden virtual functions.

只要是繼承下來的class，在繼承者(derived class)的constructor裡面，就應該呼叫到被繼承者(base class)的constructor。至於何時可以忽略這個call的話，只要是鑽石型的繼承，應該呼叫的是上上層的constructor，之後再來呼叫上層的兩個或以上個的constructor，只不過Compiler會自動幫我們忽略。

Constructor 需要、Deconstructor 需要、overridden virtual functions 不需要



6. How does it feel to add a new object (e.g. Scene, Button, Tower, Enemy, …) in OOP style instead of Procedural Programming style? In your opinion, which coding style is better?

我是覺得以新增物件的角度來看待的話，是簡單一些，因為每個secene-object/Button-object…所做的事情雖然個別不同，但是大致上差不多，所以比較好維護，而且以資料夾的型式呈現的話比較容易debug。而如果是Procedural Programming，上面2.的MissileBullet的例子，就算連擁有OOP的Encapsulation但無其他像是Inheritance的東西，要維護都已經非常不容易了，更何況是連Class都沒有的Procedural Programming。以長遠眼光來講，以OOP的方式來寫code比較好。但OOP的缺點就是門檻很高，想要踏進不太容易，因為有太多太多的規則需要遵守。雖然這些規則都有一定的道理存在，但如果不知道的話很容易誤入陷阱。以Constructor為例，光是Constructor就有分Default、Copy、Move……所以雖然現在被OOP搞得很痛苦，不過如果上手之後應該可以倒吃甘蔗。

7. What are the advantages of performing a reverse BFS, starting from our base instead of starting from each enemy? What will happen if we replace BFS with DFS? Can pathfinding be achieved through multiple forward BFSs? (Multiple BFSs starting from each enemy)

反向的BFS的優點在於只需要做一次即可，因為終點的數值是最小，所以敵人只要根據目前所在tile的數值往小的地方走即可。當然，Multiple BFSs也可以達到敵人往終點走的目的，不過因為每個敵人所在的位置不同，所以要根據每個敵人的位置為每個敵人量身定做一個數值表格，這樣有點浪費時間和RAM資源，而且只要多新增一組數據(此處指涉value table)，就會多很多很多很多變數，這樣要debug或維護程式也相應地會複雜很多。

至於BFS和DFS的比較，BFS肯定是比DFS好。DFS和BFS都具有完全性，但明顯BFS是最佳解而DFS不是。主要的原因是，BFS是一次性地比較此node的周圍(Breadth First)，而且唯一確定以後就完全不會再更動。DFS是一次性地把一個路徑探到最底(Depth First)，可是也許有兩個node都有相接，有多餘一條的路可以通，這樣的話也許這一條全部探完之後，也的確有解了。但下一條探完之後，發現這個解比上一次得到的解更好，就會做修改。所以DFS具有完全性，但每次探到最底之後雖然可以確定一個解，但不是唯一確定最佳解。所以會造成DFS相比BFS耗時多很多。

```cpp
std::vector<std::vector<int>> PlayScene::CalculateBFSDistance() {
    bool visited[MapHeight][MapWidth]; memset(visited, _Val: false, _Size: sizeof(visited));

    // Reverse BFS to find path.
    std::vector<std::vector<int>> map(_Count: MapHeight, _Val: std::vector<int>(std::vector<int>(_Count: MapWidth, _Val: -1)));//初始值全都是-1。
    std::queue<Engine::Point> que;
    // Push end point.
    // BFS from end point.
    if (mapState[MapHeight - 1][MapWidth - 1] != TILE_DIRT)
        return map;
    que.push(_Val: Engine::Point(x: MapWidth - 1, y: MapHeight - 1));
    map[MapHeight - 1][MapWidth - 1] = 0;
    visited[MapHeight - 1][MapWidth - 1] = true;
    while (!que.empty()) {
        Engine::Point p = que.front();
        que.pop();

        //OK
        // TODO 3 (1/1): Implement a BFS starting from the most right-bottom block in the map.
        //               For each step you should assign the corresponding distance to the most right-bottom block.
        //               mapState[y][x] is TILE_DIRT if it is empty.
        int dis_of_this_tile = INT_MAX;
        //找尋左上右下的不等於-1的最小值，之後再+1即可
        //開始分別依序處理左、上、右、下
        //左
        if (0 <= p.x - 1 && p.x - 1 < MapWidth && 0 <= p.y && p.y < MapHeight) {//確定是在有效的map矩形範圍
            Engine::Point left(x: p.x - 1, p.y);
            if (map[left.y][left.x] == -1 && mapState[left.y][left.x] == TILE_DIRT) {//如果現在這個tile還沒被計算過距離(因為==-1)，而且這個tile確實是可以放武器的
                if (!visited[(int)left.y][(int)left.x]) {//再確認這個tile確實沒有在que裡面
                    que.push(left); visited[(int)left.y][(int)left.x] = true;
                }
            }
            else if (map[left.y][left.x] != -1 && map[left.y][left.x] < dis_of_this_tile)dis_of_this_tile = map[left.y][left.x];
        }
        //上
        if (0 <= p.x  && p.x < MapWidth && 0 <= p.y - 1 && p.y - 1 < MapHeight) {
            Engine::Point up(p.x, y: p.y - 1);
            if (map[up.y][up.x] == -1 && mapState[up.y][up.x] == TILE_DIRT) {
                if (!visited[(int)up.y][(int)up.x]) {
```

8. The header of the slider declares three Engine::Image objects. How do you utilize them to implement the slider? If you do not use them to implement the slider, how can you make use of them by reusing the methods in the
Engine::Image class?

三個Engine::Image objects分別是Bar、End1、End2。

因為Image這個Class繼承了IObject這個專門管控可以Draw出東西來的Class，所以我只要調用Bar(Image-class，同時有了繼承的is-a的特性: is an IObject)的Draw function，即可把Bar的圖片畫出來。End1和End2同理。

```
Slider::Slider(float x, float y, float w, float h) :
    ImageButton(img:"stage-select/slider.png", imgIn:"stage-select/slider-blue.png", x:x, y:y),
    Bar(img:"stage-select/bar.png", x:x, y:y, w, h),
    End1(img:"stage-select/end.png", x:x, y:y + h / 2, w:0, h:0, anchorX:0.5, anchorY:0.5),
    End2(img:"stage-select/end.png", x:x + w, y:y + h / 2, w:0, h:0, anchorX:0.5, anchorY:0.5){
    Position.x += w;
    Position.y += h / 2;
    Anchor = Engine::Point(x:0.5, y:0.5);
}
//OK
// TODO 4 (0/6): Finish the 6 functions below and ensure it can control both BGM and SFX volume.
//          The slider shouldn't be dragged outside the bar, and you should also keep the mouse-in / mouse-out effect.
void Slider::Draw() const {
    //OK
    // TODO 4 (1/6): Draw all components.
    Bar.Draw();
    End1.Draw();//畫左側兩個
    End2.Draw();//畫右側兩個
    ImageButton::Draw();
}
void Slider::SetOnValueChangedCallback(std::function<void(float value)> onValueChangedCallback) {
    //OK
    // TODO 4 (2/6): Set the function pointer. Can imitate ImageButton's 'SetOnClickCallback'.
    OnValueChangedCallback = onValueChangedCallback;
}
void Slider::SetValue(float value) {
    //OK
    // TODO 4 (3/6): Call 'OnValueChangedCallback' when value changed. Can imitate ImageButton's 'OnClickCallback'.
    //          Also move the slider along the bar, to the corresponding position.
    if (value > Max || value < Min)return;
    if(this->value != value && Down){
        OnValueChangedCallback(value);
        Position.x = End1.Position.x + value * Bar.GetBitmapWidth();
    }

}
void Slider::OnMouseDown(int button, int mx, int my) {
    //OK
    // TODO 4 (4/6): Set 'Down' to true if mouse is in the slider.
    if (Position.x - GetBitmapWidth() / 2 <= mx && mx <= Position.x + GetBitmapWidth() / 2 && Position.y - GetBitmapHeight() / 2 <= my && my <= Position.y + GetBitmapHeight() / 2) {
        Down = true;
    }
}
void Slider::OnMouseUp(int button, int mx, int my) {
    //OK
    // TODO 4 (5/6): Set 'Down' to false.
    Down = false;
}
void Slider::OnMouseMove(int mx, int my) {
    //OK
```

9. The game crashes when the player wins. What tools and techniques did you try to find the bug? Why does the bug crash the game? What have you learned through this debugging experience?

我是用VS。基本上助教教的所有debug的技巧我上學期都摸得熟門熟練了(因為自己寫的code的bug實在有夠多)。

(1) VS的自動排版：平常很常用到，不過這次沒用到，因為助教的coding style滿整潔的

(2) Auto Completion、Show Docs：平常常用，這次project也常用到，不過如果是指抓這個bug的話沒用到。

(3) Code Navigation：抓這個bug的主要方法。

(4) Break Points、Call Stacks：抓這個bug的主要方法。

基本上這次抓bug的方式是Code Navigation和Break Points和Call Stacks，而這也是我最常用的方法。Break Points可以逐次看目前的變數的狀態是什麼，然後我就發現delete 那邊，這個變數早就被delete過了，為什麼還要delete一次? 所以就發現是重複delete的問題。

這次學到的經驗是，原來VS本來就已經debug很強大了，還有更強大的東西叫做

ReSharper，是VS的外掛插件，一年的租金是一萬台幣，還好學生免費。

10. What is the hidden cheat code sequence in the game that can increase your money and nuke the entire map? How do you find it? How does the template code detect the entered cheat code sequence?

上上下下左右左右baa enter或者上上下下左右左右ba shift enter。

我就trace code的時候發現????? `EffectGroup->AddNewObject(new Plane());money += 10000;` 是啥???我怎麼都沒印象有Plane這種東西，所以就去把它上面的code給看了一下，發現需要輸入一串密語即可開啟遊戲特別寫出來的外掛，就查了一下code裡面的所有編號，去Allegro官網查這些數字所代表的鍵盤key是什麼。
至於怎麼監測到cheat code嘛...就是不管使用者輸入什麼，只要鍵盤一按下去，就把這個鍵紀錄起來。Code裡面有監測目前容納的連續鍵入的上限數量，當keyCode.size()>code.size()的時候就把目前keyCode所記錄的最早鍵入的鍵給刪掉。然後就依序判斷這個keyCode的sequence的每個元素是否吻合code的每個元素。

```
void PlayScene::OnKeyDown(int keyCode) {
    IScene::OnKeyDown(keyCode);
    if (keyCode == ALLEGRO_KEY_TAB) {
        DebugMode = !DebugMode;
    }
    else {
        keyStrokes.push_back(keyCode);
        if (keyStrokes.size() > code.size())
            keyStrokes.pop_front();
        if (keyCode == ALLEGRO_KEY_ENTER && keyStrokes.size() == code.size()) {
            auto it _List_iterator<_List_val<_List_simple_types<int>>> = keyStrokes.begin();
            for (int c : code) {
                if (!((*it == c) ||
                    (c == ALLEGRO_KEYMOD_SHIFT &&
                    (*it == ALLEGRO_KEY_LSHIFT || *it == ALLEGRO_KEY_RSHIFT))))
                    return;
                ++it;
            }
            EffectGroup->AddNewObject(new Plane());
            money += 10000;
        }
    }
const std::vector<int> PlayScene::code = {ALLEGRO_KEY_UP, ALLEGRO_KEY_UP, ALLEGRO_KEY_DOWN, ALLEGRO_KEY_DOWN,
                                          ALLEGRO_KEY_LEFT, ALLEGRO_KEY_RIGHT, ALLEGRO_KEY_LEFT, ALLEGRO_KEY_RIGHT,
                                          ALLEGRO_KEY_B, ALLEGRO_KEY_A, ALLEGRO_KEYMOD_SHIFT, ALLEGRO_KEY_ENTER };
```

11. When there are a certain number of enemies in the “danger zone”, the speed multiplier is forced to be lower than one and an intense BGM starts playing. What is the filename of the BGM? What will happen to the intense BGM if the player gets through the current crisis but immediately encountered another crisis?

File Name: `astronomia.ogg`

當get through 前面一個crisis，astronomia.ogg的播放就會被掐斷。如果immediately encounter另一個crisis，astronomia.ogg會再從頭播放一次。

```
95  else if (deathCountDown != -1)
96      SpeedMult = 1;
97  // Calculate danger zone.
98  std::vector<float> reachEndTimes;
99  for (auto& it:IObject*& : EnemyGroup->GetObjects()) {
100     reachEndTimes.push_back(dynamic_cast<Enemy*>(it)->reachEndTime);
101 }
102 // Can use Heap / Priority-Queue instead. But since we won't have too many enemies, sorting is fast enough.
103 std::sort(_First:reachEndTimes.begin(), _Last:reachEndTimes.end());
104 float newDeathCountDown = -1;
105 int danger = lives;
106 for (auto& it:float& : reachEndTimes) {
107     if (it <= DangerTime) {
108         danger--;
109         if (danger <= 0) {
110             // Death Countdown
111             float pos = DangerTime - it;
112             if (it > deathCountDown) {
113                 // Restart Death Count Down BGM.
114                 AudioHelper::StopSample(deathBGMInstance);
115                 if (SpeedMult != 0)
116                     deathBGMInstance = AudioHelper::PlaySample(audio:"astronomia.ogg", loop:false, AudioHelper::BGMVolume, position:pos);
117             }
118             float alpha = pos / DangerTime;
119             alpha = std::max(_Left:0, _Right:std::min(_Left:255, _Right:static_cast<int>(alpha * alpha * 255)));
120             dangerIndicator->Tint = al_map_rgba(r:255, g:255, b:255, a:alpha);
121             newDeathCountDown = it;
122             break;
123         }
```

12. When will the function main return the exit value zero?

`GameEngine.cpp` 裡的 `while(!done)`裡有 `event loop`，如果玩家按下視窗關閉按鈕的話會把 `done` 設為 `true`，這樣就跳出 `while` 迴圈，紀錄 `LOG` 為 `Terminating…`，然後進行 `activeScene->Terminate`，然後 `destroy()`，之後回到 `main` 進行 `return 0`。

```
80  void GameEngine::startEventLoop() {
81      bool done = false;
82      ALLEGRO_EVENT event;
83      int redraws = 0;
84      auto timestamp:time_point<steady_clock> = std::chrono::steady_clock::now();
85      while (!done) {
86          al_wait_for_event(event_queue, &event);
87          switch (event.type) {
88          case ALLEGRO_EVENT_DISPLAY_CLOSE:
89              // Event for clicking the window close button.
90              LOG(type:VERBOSE) << "Window close button clicked";
91              done = true;
92              break;
93          case ALLEGRO_EVENT_TIMER:
94              // Event for redrawing the display.
95              if (event.timer.source == update_timer)
96                  // The redraw timer has ticked.
97                  redraws++;
98              break;
99          case ALLEGRO_EVENT_KEY_DOWN:
100             // Event for keyboard key down.
101             LOG(type:VERBOSE) << "Key with keycode " << event.keyboard.keycode << " down";
102             activeScene->OnKeyDown(event.keyboard.keycode);
103             break;
```

B. Medium Questions (2%, TA will randomly pick 2 questions below)

1. When the value of the Slider changes, it fires a callback. Which code file contains the code of the callback function? How does that function change the BGM / SFX audio volumes?

參考https://reurl.cc/3Dm0G9。當value改變時，會call這個function:
`OnValueChangedCallback(value);`。但如果去`Slider.hpp`看的話，`OnValueChangedCallback`不是一個`function`，而是一個`private data member`，長成這副德行:

```
std::function<void(float value)> OnValueChangedCallback;。
```

根據上面那個網頁的例子，其實`OnValueChangedCallback`這個東西是一個類似function pointer的東西，在StageSelectScene.cpp裡面有用`sliderBGM->SetOnValueChangedCallback(std::bind(&StageSelectScene::BGMSlideOnValueChanged, this, std::placeholders::_1));`來把這個function跟`BGMSlideOnValueChanged`綁在一起。

同樣的，裡面也有`sliderSFX->SetOnValueChangedCallback(std::bind(&StageSelectScene::SFXSlideOnValueChanged, this, std::placeholders::_1));`。

所以當執行`OnValueChangedCallback(value);`的時候，會呼叫StageSelectScene.cpp的

```
void StageSelectScene::BGMSlideOnValueChanged(float value) {
    AudioHelper::ChangeSampleVolume(bgmInstance, value);
    AudioHelper::BGMVolume = value;
}
```

之後`ChangeSampleVolume(bgmInstance, value);`再利用allegro提供的`interface: al_set_sample_instance_gain(sample_instance.get(), volume)`來修改數值。

2. In stage 2, we can build a maze through turrets, forcing enemies to take more time before reaching our base. However, there are some limits on where the player can build the turret. What are the limits and how are the limits implemented?

在PlayScene.cpp裡的`void PlayScene::OnMouseUp(int button, int mx, int my)`函式，有使用到函式`// Check if valid. if (!CheckSpaceValid(x, y))`，
其中裡面有先把想放的目標tile設為TILE_OCCUPIED，之後又重新`std::vector<std::vector<int>> map = CalculateBFSDistance();`把「假定已經把`turret`放進去目標`tile`」的數字表格算好。如果敵人想要攻擊的目標完全被包圍堵住的話，所有外圍的`tile`的數字都會被擋住，所以數字都是最原始`memset`成的`-1`，因此在那之後的`if (map[0][0] == -1)return false;`一定會執行。而之後的一連串`if`是在判斷敵人的位置，因為敵人所在之處也不能直接建砲塔。

```
450 bool PlayScene::CheckSpaceValid(int x, int y) {
451     if (x < 0 || x >= MapWidth || y < 0 || y >= MapHeight)
452         return false;
453     auto map00 :PlayScene::TileType = mapState[y][x];
454     mapState[y][x] = TILE_OCCUPIED;
455     std::vector<std::vector<int>> map = CalculateBFSDistance();
456     mapState[y][x] = map00;
457     if (map[0][0] == -1)
458         return false;
459     for (auto& it :IObject*& : EnemyGroup->GetObjects()) {
460         Engine::Point pnt;
461         pnt.x = floor(_Xx: it->Position.x / BlockSize);
462         pnt.y = floor(_Xx: it->Position.y / BlockSize);
463         if (pnt.x < 0) pnt.x = 0;
464         if (pnt.x >= MapWidth) pnt.x = MapWidth - 1;
465         if (pnt.y < 0) pnt.y = 0;
466         if (pnt.y >= MapHeight) pnt.y = MapHeight - 1;
467         if (map[pnt.y][pnt.x] == -1)
468             return false;
469     }
470     // All enemy have path to exit.
471     mapState[y][x] = TILE_OCCUPIED;
472     mapDistance = map;
473     for (auto& it :IObject*& : EnemyGroup->GetObjects())
474         dynamic_cast<Enemy*>(it)->UpdatePath(mapDistance);
475     return true;
476 }
```

3. How does the game parse the file containing the map and the file containing the enemy sequence? How are the enemies generated according to the timer ticks?

```cpp
void PlayScene::ReadMap() {
    std::string filename = std::string("resources/map") + std::to_string(MapId) + ".txt";
    // Read map file.
    char c;
    std::vector<bool> mapData;
    std::ifstream fin(filename);
    while (fin >> c) {
        switch (c) {
        case '0': mapData.push_back(false); break;
        case '1': mapData.push_back(true); break;
        case '\n':
        case '\r':
            if (static_cast<int>(mapData.size()) / MapWidth != 0)
                throw std::ios_base::failure("Map data is corrupted.");
            break;
        default: throw std::ios_base::failure("Map data is corrupted.");
        }
    }
    fin.close();
    // Validate map data.
    if (static_cast<int>(mapData.size()) != MapWidth * MapHeight)
        throw std::ios_base::failure("Map data is corrupted.");
    // Store map in 2d array.
    mapState = std::vector<std::vector<TileType>>(MapHeight, std::vector<TileType>(MapWidth));
    for (int i = 0; i < MapHeight; i++) {
        for (int j = 0; j < MapWidth; j++) {
            const int num = mapData[i * MapWidth + j];
            mapState[i][j] = num ? TILE_FLOOR : TILE_DIRT;
            if (num)
                TileMapGroup->AddNewObject(new Engine::Image("play/floor.png", j * BlockSize, i * BlockSize, BlockSize, BlockSize));
            else
                TileMapGroup->AddNewObject(new Engine::Image("play/dirt.png", j * BlockSize, i * BlockSize, BlockSize, BlockSize));
        }
    }
}
```

floor是藍色，dirt是土色。注意mapData只有0和1，共兩種選項

```cpp
void PlayScene::ReadEnemyWave() {
    std::string filename = std::string("resources/enemy") + std::to_string(MapId) + ".tx
    // Read enemy file.
    float type, wait, repeat;
    enemyWaveData.clear();//std::list<std::pair<int, float>> enemyWaveData;
    std::ifstream fin(filename);
    while (fin >> type && fin >> wait && fin >> repeat) {
        for (int i = 0; i < repeat; i++)
            enemyWaveData.emplace_back(type, wait);//類似push_back
    }
    fin.close();
}
```

```cpp
auto current=pair<int, float> = enemyWaveData.front();
if (ticks < current.second)
  continue;
ticks -= current.second;
/*ticks是從開啟playscene就開始數時間，但是一過這個敵人所需要的時間，就把ticks扣掉，之後繼續數
*，達到可以的時間就又出產敵人，然扣又扣掉ticks，如此反反覆覆*/
enemyWaveData.pop_front();
const Engine::Point SpawnCoordinate = Engine::Point(x SpawnGridPoint.x * BlockSize + BlockSize / 2, y SpawnGridPoint.y * BlockSize + BlockSize / 2);
Enemy* enemy;
switch (current.first) {
case 1:
  EnemyGroup->AddNewObject(obj enemy = new SoldierEnemy(SpawnCoordinate.x, SpawnCoordinate.y));
  break;
case 2:
  EnemyGroup->AddNewObject(obj enemy = new PlaneEnemy(SpawnCoordinate.x, SpawnCoordinate.y));
  break;
```

4. What did the template do to achieve different speed multipliers? When the speed multiplier is zero, can the player build new turrets? Why or why not?

```
for (int i = 0; i < SpeedMult; i++) {
    IScene::Update(deltaTime);
    // Check if we should create new enemy.
    ticks += deltaTime;
    if (enemyWaveData.empty()) {
        if (EnemyGroup->GetObjects().empty()) {
            // Free resources.
            /*delete TileMapGroup;
            delete GroundEffectGroup;
            delete DebugIndicatorGroup;
            delete TowerGroup;
            delete EnemyGroup;
            delete BulletGroup;
            delete EffectGroup;
            delete UIGroup;*/
            //delete imgTarget;
            // Win.
            Engine::GameEngine::GetInstance().ChangeScene(name: "win");
        }
        continue;
    }
    auto current:pair<int, float> = enemyWaveData.front();
    if (ticks < current.second)
```

透過for迴圈來加快數ticks的速度和更新畫面。而因為sample不在for迴圈裡面更新，所以音樂並不會加快速度。如果暫停，代表SpeedMult是0，所以不會進入for迴圈，所以關於子彈以及turret砲管方向、爆炸髒汙等等畫面不會進行更新，也不會計算ticks，所以不可能有新的敵人。!!但是!!如同Medium Question的第二題，可不可以新建turret是交給這個函式做決定

```
bool PlayScene::CheckSpaceValid(int x, int y) {
    if (x < 0 || x >= MapWidth || y < 0 || y >= MapHeight)
        return false;
    auto map00:PlayScene::TileType = mapState[y][x];
    mapState[y][x] = TILE_OCCUPIED;
    std::vector<std::vector<int>> map = CalculateBFSDistance(
    mapState[y][x] = map00;
    if (map[0][0] == -1)
        return false;
    for (auto& it:IObject*& : EnemyGroup->GetObjects()) {
        Engine::Point pnt;
        pnt.x = floor(_Xx:it->Position.x / BlockSize);
        pnt.y = floor(_Xx:it->Position.y / BlockSize);
        if (pnt.x < 0) pnt.x = 0;
        if (pnt.x >= MapWidth) pnt.x = MapWidth - 1;
        if (pnt.y < 0) pnt.y = 0;
        if (pnt.y >= MapHeight) pnt.y = MapHeight - 1;
        if (map[pnt.y][pnt.x] == -1)
            return false;
    }
    // All enemy have path to exit.
    mapState[y][x] = TILE_OCCUPIED;
    mapDistance = map;
    for (auto& it:IObject*& : EnemyGroup->GetObjects())
        dynamic_cast<Enemy*>(it)->UpdatePath(mapDistance);
    return true;
```

而這個函式並沒有因為SpeedMult的改變而受到任何影響，所以完全是可以新建砲塔的，只要沒有把錢敗光光而且地圖上還有空位而且這個空位確實是可放的的話。

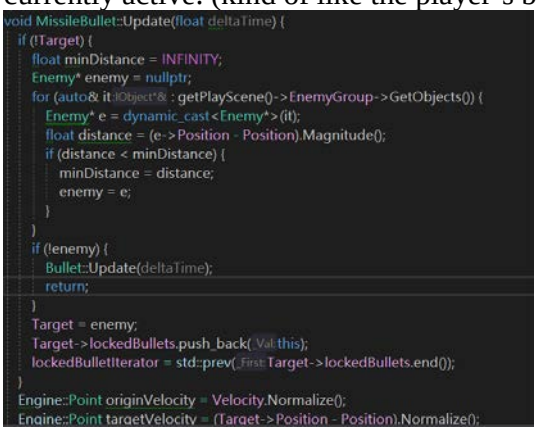
5. How does the homing missile of the Missile Turret follow the enemy? Explain the math behind the homing missile.

```
Engine::Point originVelocity = Velocity.Normalize();
Engine::Point targetVelocity = (Target->Position - Position).Normalize();
float maxRotateRadian = rotateRadian * deltaTime;
float cosTheta = originVelocity.Dot(targetVelocity);
// Might have floating-point precision error.
if (cosTheta > 1) cosTheta = 1;
else if (cosTheta < -1) cosTheta = -1;
float radian = acos(cosTheta);
if (abs(radian) <= maxRotateRadian)
    Velocity = targetVelocity;
else
    Velocity = ((abs(radian) - maxRotateRadian) * originVelocity + maxRotateRadian * targetVelocity) / radian;
Velocity = speed * Velocity.Normalize();
Rotation = atan2(_Yx:Velocity.y, _Xx:Velocity.x) + ALLEGRO_PI / 2;
Bullet::Update(deltaTime);
```

先把原速度取單位向量，同時也取目標所造成的位移的單位向量，透過數學的內積公式(a, b) dot (x, y) = |(a, b)|x|(x, y)|x cosΘ。因為(a, b)和(x, y)都已經是單位向量，所以算出來的值就直接是cosΘ。再來就判斷這個目標要轉的角度和最大能轉的角度是否衝突。如果OK，就速度轉v，否則速度進行對應的計算。之後夾角一併調整。之後轉彎角度設為 v向量的極座標角度 +pi/2

6. The homing missile of the Missile Turret automatically selects a target enemy to follow. What happens when the target enemy dies or the missile explodes? Does the template perform extra

bookkeeping code to deal with this situation? Explain why iterators are used in such a way. If we reuse the enemy instance without deleting it upon dying, can we remove the bookkeeping code? Note that we can reuse the instance by simply adding a flag, indicating whether the instance is currently active. (kind of like the player's bullet in last semester's final project hackathon)



```
void MissileBullet::Update(float deltaTime) {
  if (!Target) {
    float minDistance = INFINITY;
    Enemy* enemy = nullptr;
    for (auto& it:IObject*& : getPlayScene()->EnemyGroup->GetObjects()) {
      Enemy* e = dynamic_cast<Enemy*>(it);
      float distance = (e->Position - Position).Magnitude();
      if (distance < minDistance) {
        minDistance = distance;
        enemy = e;
      }
    }
    if (!enemy) {
      Bullet::Update(deltaTime);
      return;
    }
    Target = enemy;
    Target->lockedBullets.push_back(_Val:this);
    lockedBulletIterator = std::prev(_First:Target->lockedBullets.end());
  }
  Engine::Point originVelocity = Velocity.Normalize();
  Engine::Point targetVelocity = (Target->Position - Position).Normalize();
```

Target enemy dies情況: 每個missile在每次敵人死亡之後，都會重新計算目標，而目標要找誰? 就是找距離我這個missile最近(不是離missileTurret最近!!而是離missile最近!!)的敵人，所以就用range-based的for-loop把所有敵人都算過一次距離，取最小的(亦即最接近的)敵人作為攻擊目標。如果場上一個敵人都沒有，就會進入圖中的if(!enemy)部分。既然沒有敵人，那就不需要特定目標，所以根本不用改變速度，就按照目前的切線速度繼續給他飛下去，等到飛出螢幕之外，會自動被回收記憶體。

Missile explodes情況: 就跟其他bullet一樣的方式，弄一些dirt的視覺和聲音特效，以及回收記憶體。

Does the template perform extra bookkeeping code to deal with this situation? Ans: 有，用Iterator，而使用Iterator是只有MissileBullet才有的，其他諸如FireBullet、ChiBullet都是沒有的。

```
void MissileBullet::Update(float deltaTime) {
  if (!Target) {
    float minDistance = INFINITY;
    Enemy* enemy = nullptr;
    for (auto& it:IObject*& : getPlayScene()->EnemyGroup->GetObjects()) {
      Enemy* e = dynamic_cast<Enemy*>(it);
      float distance = (e->Position - Position).Magnitude();
      if (distance < minDistance) {
        minDistance = distance;
        enemy = e;
      }
    }
    if (!enemy) {
      Bullet::Update(deltaTime);
      return;
    }
    Target = enemy;//Target 是Enemy*。
    Target->lockedBullets.push_back(_Val:this);//被這個"missile"bullet新鎖定的enemy，也記錄一下自己被這顆bullet鎖定了
    lockedBulletIterator = std::prev(_First:Target->lockedBullets.end());
    //因為Target->lockedBullets是std::list<Bullets*>所以Target->lockedBullets.end()是Bullets*，故lockedBulletIterator也是Bullets*
    //std::prev可參考https://blog.csdn.net/u013630349/article/details/47105319
    //std::prev(..., a)相當於std::advance(..., -a)。advance是往前走a個iterator(往前是指類似於往陣列idx大的方向)，那當然如果是負數的話，就類似於向量方向
    //所以std::prev(Target->lockedBullets.end());相當於找出Tagrget->lockedBullets.end()!!!注意!!!end()是指整個有效範圍「後」一個iterator!!!
    //因此才需要std::prev，往前一格才能回到有效位置。
    //lockedBulletIterator是個在MissileBullet.hpp裡面的std::list<Bullet*>::iterator lockedBulletIterator，記錄著每個已經鎖定好enemy目標的bullet。
```

之後，如果這個missileBullet已經二維碰撞成功後，就進行OnExplode，第一行就先把這顆missileBullet從`std::list<Bullet*>::iterator lockedBulletIterator`移除，因為這顆子彈都要被銷毀了，當然他這顆子彈是沒有目標enemy的。

```
void MissileBullet::OnExplode(Enemy* enemy) {
    Target->lockedBullets.erase(lockedBulletIterator);
    std::random_device dev;
    std::mt19937 rng(_X0: dev());
    std::uniform_int_distribution<std::mt19937::result_type> dist(_Min0: 4, _Max0: 10);
    getPlayScene()->GroundEffectGroup->AddNewObject(new DirtyEffect(img: "play/dirty-3.png", timeSpan: dist([&] rng), enemy->Position.x, enemy->Position.y));
}
```

7. In the Play scene, multiple Engine::Groups are used for storing different objects. What will happen if we remove the groups and add all new objects directly into the scene?

要寫資料更新與維護、記憶體回收的code的時候你的腦袋會燒壞，因為根本搞不清楚我處理過誰了，誰還沒處理過。如果把所有turret歸在同一類，所有bullet歸在同一類，所有tile歸在同一類，這樣要管理比較方便。

因此，利用物件導向建好可以作好部分共同的部分，只要activeScene->Update(deltaTime)便更新所有Group。

```
void GameEngine::update(float deltaTime) {
    if (!nextScene.empty()) {
        changeScene(nextScene);
        nextScene = "";
    }
    // Force lag to avoid bullet-through-paper issue.
    if (deltaTime >= deltaTimeThreshold)
        deltaTime = deltaTimeThreshold;
    activeScene->Update(deltaTime);
}
```

8. The public change scene function of Engine::GameEngine does not change the scene immediately but waits until the next update. What might happen if the scene is changed immediately when the public function is called?

```
147         // Redraw the scene.
148         if (redraws > 0 && al_is_event_queue_empty(event_queue)) {
149             if (redraws > 1)
150                 LOG(type: VERBOSE) << redraws - 1 << " frame(s) dropped";
151             // Calculate the timeElapsed and update the timestamp.
152             auto nextTimestamp: time_point<steady_clock> = std::chrono::steady_clock::now();  nextTimestamp = {_MyDur=4931426979527000 nanoseconds }
153             std::chrono::duration<float> timeElapsed = nextTimestamp - timestamp;  timeElapsed = 56.0011520 seconds
154             timestamp = nextTimestamp;
155             // Update and draw the next frame.
156             update(deltaTime: timeElapsed.count());
157             draw();
158             redraws = 0;
```

，之後:

```
void GameEngine::update(float deltaTime) {  deltaTime = 56.0011520
    if (!nextScene.empty()) {
        changeScene(nextScene);
        nextScene = "";  nextScene = "stage-select"
    }
```

，之後:

```
void GameEngine::changeScene(const std::string& name) {
    if (scenes.count(name) == 0)
        throw std::invalid_argument( Message: "Cannot change to a unknown scene.");
    // Terminate the old scene.
    activeScene->Terminate();
    activeScene = scenes[name];
    // Release unused resources.
    if (freeMemoryOnSceneChanged)
        Resources::GetInstance().ReleaseUnused();
    // Initialize the new scene.
    activeScene->Initialize();
    LOG( type: INFO) << "Changed to " << name << " scene";
}
```

所以，是得先把目前所在scene給Terminate(就是把所有imagebutton阿sprite阿、sampleAudio之類的東西清掉)再把新的scene做初始化，之後才能draw，否則如果反覆change scene，就會memory leak。

9. There is a deltaTime parameter in the update function, what are the benefits of using this parameter? What might happen if we remove the threshold of this parameter in Engine::GameEngine?

```
// Redraw the scene.
if (redraws > 0 && al_is_event_queue_empty(event_queue)) {
    if (redraws > 1)
        LOG(type:VERBOSE) << redraws - 1 << " frame(s) dropped";
    // Calculate the timeElapsed and update the timestamp.
    auto nextTimestamp:time_point<steady_clock> = std::chrono::steady_clock::now();
    std::chrono::duration<float> timeElapsed = nextTimestamp - timestamp;
    timestamp = nextTimestamp;
    // Update and draw the next frame.
    update(deltaTime:timeElapsed.count());
    draw();
    redraws = 0;
}
```

(如果要redraw，計算畫面和畫面間的時間差，之後進入update函式)

```
void GameEngine::update(float deltaTime) {
    if (!nextScene.empty()) {
        changeScene(nextScene);
        nextScene = "";
    }
    // Force lag to avoid bullet-through-paper issue.
    if (deltaTime >= deltaTimeThreshold)
        deltaTime = deltaTimeThreshold;
    activeScene->Update(deltaTime);
}
```

進入update函式，看要不要判斷nextscene，之後active->scene會:

```
void Group::Update(float deltaTime) {
    for (auto it:_List_iterator<...> = objects.begin(); it != objects.end();) {
        auto prevIt:_List_iterator<...> = it++;
        if (prevIt->second->Visible)
            prevIt->second->Update(deltaTime);
    }
}
```

，之後各個子彈再各自update(就是(x, y)座標變換)

```
void Sprite::Update(float deltaTime) {
    Position.x += Velocity.x * deltaTime;
    Position.y += Velocity.y * deltaTime;
}
namespace Engine;
```

其實我會覺得if(deltaTime >= deltaTimeThreshold)這個判斷式很冗，因為照理講完全不會用到。

deltaTime = a-b，而且a跟b都是自動從電腦獲取時間戳記，所以照理講deltaTime不可能很大，更不可能大到超過dealtaTimeThreshold的範圍，所以這個if判斷式永遠不會成立，除非我用IDE的中斷點功能把畫面中斷，這樣的話才有可能獲取到的兩個時間戳記的差，很大。但使用者不可能會用IDE的中斷點功能，所以我再想了一下，會不會是為了預防老舊的電腦(或更精確講，預防運算能力比較跟不上的電腦)，會有deltaTime很大的情況。

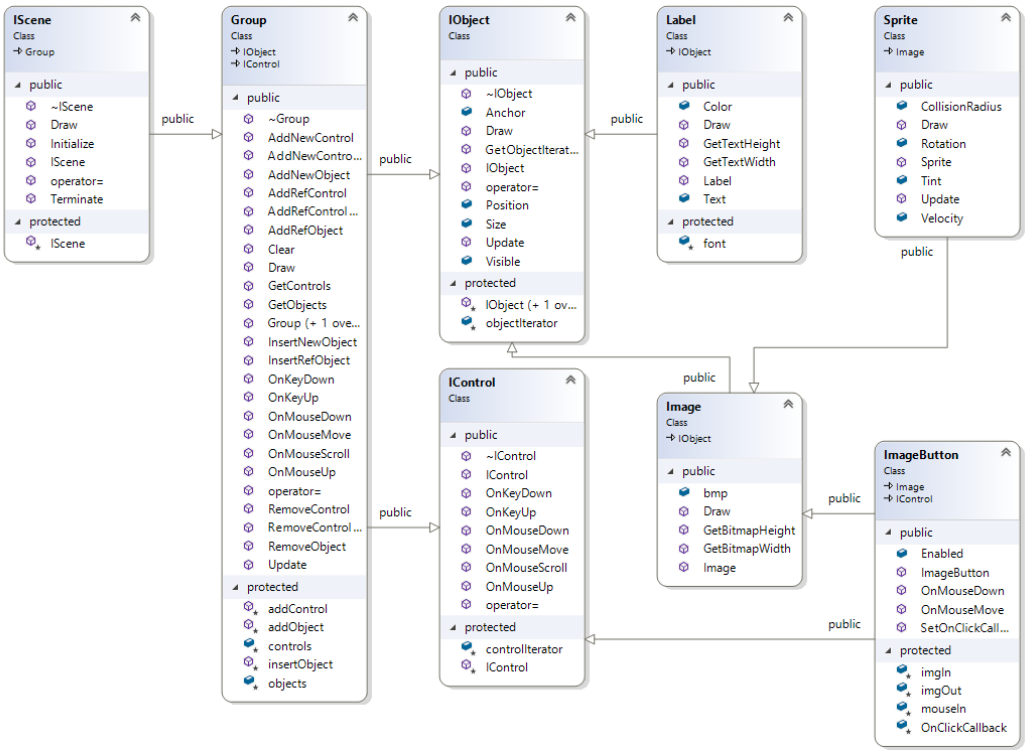
如果是運算吃力的電腦，比如說要從stage-select-scene切換到playscene，可是因為運算量太大，deltaTime太大，可能三分鐘之類的，這樣會導致終於成功切換場景成遊戲場景後，玩家馬上就輸了，要跳到lose secene(因為這三分鐘內完全沒有玩到，但是遊戲的運算仍然會持續進行)，玩家會覺得莫名其妙。

10. In Engine::Group, there are two linked-lists, storing objects and controls respectively. Can we simplify this into a single linked-list? What are the advantages and disadvantages of storing them in different linked-lists?

感覺 one linked-list 在這裡不好。因為 list 裡面的內容物是 pair，一種是<bool, IObject*>，一種是<bool, IControl*>，而 IObject 和 IControl 是兩個完全沒有任何關係的 class，所以如果要合併的話會有型別上的困擾。而且如果即便沒有型別上的困擾，這樣弄成單一一個 linked list 好像也沒有很好地發揮 OOP 這種權責區分、資料夾分工、各司其職的生產線的感覺。

```
class Group: public IObject, public IControl {
protected:
    // Stores all object pointers in the scene.
    // The first boolean indicates whether the scene should delete it.
    std::list<std::pair<bool, IObject*>> objects;
    // Stores all control pointers in the scene.
    // The first boolean indicates whether the scene should delete it.
    std::list<std::pair<bool, IControl*>> controls;
    // Note: Using linked-list and inline-new might cause some serious cache misses,
    //       however this implementation brings us more convenience than the impact
    //       on performance. Trying to deal with cache misses requires complicated code,
    //       so we'll just ignore it for simplicity.
    /// <summary>
    /// Add Object to scene.
    /// </summary>
    /// <param name="shouldDelete">Indicates whether the scene handle the Object deletion for you.</param>
    /// <param name="obj">The Object to add.</param>
    void addObject(bool shouldDelete, IObject* obj);
    /// <summary>
    /// Add Control to scene.
    /// </summary>
    /// <param name="shouldDelete">Indicates whether the scene handle the Control deletion for you.</param>
    /// <param name="ctrl">The Control to add.</param>
    void addControl(bool shouldDelete, IControl* ctrl);
```

```
void Group::addObject(bool shouldDelete, IObject* obj) {
    objects.emplace_back(shouldDelete, obj);
    obj->objectIterator = std::prev(_First:objects.end());
}
void Group::addControl(bool shouldDelete, IControl* ctrl) {
    controls.emplace_back(shouldDelete, ctrl);
    ctrl->controlIterator = std::prev(_First:controls.end());
}
```



11. Engine::Group is a class derived from Engine::IObject, while storing a linked-list of pointers to many Engine::IObjects. Why do we store the pointer to Engine::IObjects instead of directly storing the value of Engine::IObjects?

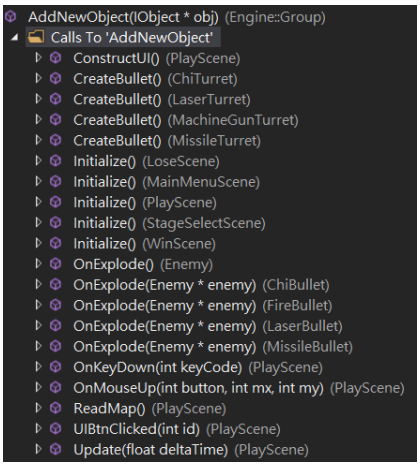
由上一題的第二張圖片的addObject函式

```
void Group::addObject(bool shouldDelete, IObject* obj) {
    objects.emplace_back(shouldDelete, obj);
    obj->objectIterator = std::prev(_First:objects.end());
}
```

為例，是誰呼叫了他? 循線找出:

```
void Group::AddNewObject(IObject* obj) {
    addObject(shouldDelete:true, obj);
}
```

。而AddNewObject所接收的argument是什麼?來自於誰?用View All Hierarchy去找，發現是:



。抓取其中的MachineGunTurret.cpp裡的CreateBullet函式為例，

```
void MachineGunTurret::CreateBullet() {
    Engine::Point diff = Engine::Point(cos(Rotation - ALLEGRO_PI / 2), sin(Rotation - ALLEGRO_PI / 2));
    float rotation = atan2(diff.y, diff.x);
    Engine::Point normalized = diff.Normalize();
    // Change bullet position to the front of the gun barrel.
    getPlayScene()->BulletGroup->AddNewObject(new FireBullet(Position + normalized * 36, diff, rotation,
    AudioHelper::PlayAudio("gun.wav");
}
```

是new FireBullet……，也就是，動態規劃。

所以整題的解答就是，因為動態規劃比較彈性，要占用記憶體或釋放記憶體都相對方便，也比較不會有這種是為在某個function裡面的local變數導致外面看不到的情況。

況且，不同場景的所需的object數目可能有所不同，所以可隨不同狀況下做大小變動才是最佳解。

12. Can we put all members in the Engine::Sprite class into Engine::Image class to combine them into a single class? What are the pros and cons of separating them?

答案在easy question的第一、二題。

13. In Win scene, the code allocates space by the new statement but does not delete them in the same file. Is this a memory leak issue? Why or why not? If the memory leakage issue exists, how can we fix it?
不是memory leak issue，因為這只不過是因為有繼承，所以交給上一層Class的function: `IScene::Terminate();`做處理罷了。

```
void Group::Clear() {
    for (auto& it : objects) {
        if (it.first) delete it.second;
    }
    objects.clear();
    for (auto& it : controls) {
        if (it.first) delete it.second;
    }
    controls.clear();
}
```

而Terminate();本身呼叫`Group::Clear()`，而`Clear()`裡面去針對每個現在仍然存在的`IObject`物件(`IObject`的定位是，只要是可以`Draw()`出來的東西，就是`IObeject`)進行`delete`，所以不會有`memory leak`的問題。

C. Hard Questions (1%, TA will randomly pick 1 question below)

1. What is Memory Leak? Can the invention of Smart Pointers reduce this problem? How are Smart Pointers implemented and what are the benefits of using them? Elaborate on the difference between std::unique_ptr, std::shared_ptr, and std::weak_ptr.

> 指標生命結束時，未釋放掉原先配置的記憶體，導致該記憶體空間後續無法被再次利用。離開使用 malloc 或 new 生出這個指標的東西的函式後，沒有任何指標有指到它，變成說後續想 free 或 delete 也做不到。
> 可以，Smart Pointers 在指標生命周期結束時，會自行釋放記憶體。
> `unique_ptr:`
> 確保被配置出來的記憶體空間只會被一個 `unique_ptr` 物件管理的 smart pointer；當指標消失時，就會自動釋放記憶體。
> `shared_ptr:`
> 可以有多個 `shared_ptr` 物件指向同一段記憶體空間。共用同一份資源，內部會記錄這份資源被使用的次數(reference

counter)，只要還有 shared_ptr 物件的存在、資源就不會釋放；只有當所有使用這份資源 shared_ptr 物件都消失的時候，資源才會被自動釋放。

weak_ptr:

基本上搭配 shared_ptr 使用，指標存活與否不影響記憶體是否被釋放。且不可透過 weak_ptr 做資料存取，僅是監控 shared_ptr 目前的狀況

C++標準庫智慧指標

使用這些智慧型指標做為封裝純舊 C++ 物件 (POCO) 指標的首要選擇。

- `unique_ptr`
  只允許一個基礎指標的擁有者。 用做 POCO 的預設選項，除非您確信自己需要 `shared_ptr`。 可以移至新擁有者，但不可複製或共用。 替換已被取代的 `auto_ptr`。 與 `boost::scoped_ptr` 比較。 `unique_ptr` 小而高效; 大小是一個指標,它支援 rvalue 引用,以便從C++標準庫集合快速插入和檢索。 標頭檔：`<memory>`。 有關詳細資訊,請參閱如何:創建和使用unique_ptr實例和unique_ptr類。

- `shared_ptr`
  參考計數的智慧型指標。 在您想要將原始指標指派給多個擁有者時使用，例如，當您從容器傳回指標的複本，但是想要保留原來的指標時。 原始指標只有在所有的 `shared_ptr` 擁有者都超出範圍或放棄擁有權之後，才會被刪除。 大小是兩個指標；一個針對物件，另一個則針對含有參考計數的共用控制區塊。 標頭檔：`<memory>`。 有關詳細資訊,請參閱如何:創建和使用shared_ptr實例和shared_ptr類。

- `weak_ptr`
  與 `shared_ptr` 一起使用的特殊案例智慧型指標。 `weak_ptr` 可以讓您存取由一個或多個 `shared_ptr` 執行個體擁有的物件，但是不會參與參考計數。 當您想要觀察物件，但不需要物件保持運作時，即可使用。 在某些要中斷 `shared_ptr` 執行個體之間循環參考的情況下為必要項。 標頭檔：`<memory>`。 有關詳細資訊,請參閱如何:創建和使用weak_ptr實例和weak_ptr類。

https://kheresy.wordpress.com/2012/03/03/c11_smartpointer_p1/

2. Many high-level languages forbid the usage of raw pointers. Does it mean that raw pointers are deprecated in this day and age? Come up with three situations when raw pointers are necessary.

對；C++20(不完全)跟C++23orC++26(完全)會移除raw pointer

when they compete with references

argument passing; but references can't be null, so are preferable.

? □ as class members to denote association rather than composition; usually preferable to references because the semantics of assignment are more straightforward and in addition an invariant set up by the constructors can ensure that they are not `0` for the lifetime of the object

? • as a handle to a (possibly polymorphic) object owned somewhere else; references can't be null so again they are preferable

? • `std::bind` uses a convention where arguments that are passed are copied into the resulting functor; however `std::bind(&T::some_member, this, ...)` only makes a copy of the pointer whereas `std::bind(&T::some_member, *this, ...)` copies the object; `std::bind(&T::some_member, std::ref(*this), ...)` is an alternative

whenthey do **not** compete with references

? • as iterators!

? • argument passing of **optional** parameters; here they compete with `boost::optional<T&>`

? • as a handle to a (possibly polymorphic) object owned somewhere else, when they can't be declared at the site of initialization; again, competing with `boost::optional<T&>`

3. Garbage Collection is a clever concept introduced in Lisp and become widely popular in higher-level programming languages such as Java, C#, Python. What does the code in Engine::Resources do? What are the benefits and drawbacks of automatically managing the resources?

```
void Resources::ReleaseUnused() {
    // A better way may be to count the memory usage and release unused resources
    // when the total resource memory exceeds a certain threshold. However, we'll
    // just keep it simple here and only release unused resources in GameEngine when
    // changing between scenes.
    for (auto it=_List_iterator<...> = bitmaps.begin(); it != bitmaps.end();) {
        if (it->second.use_count() == 1) {
            LOG(type:INFO) << "Destroyed Resource<image>: " << it->first;
            it = bitmaps.erase(it);//看起來是只要一erase，他的return值會是目前操作的it的隔壁(類似array的idx，往數字大的方向移動)
        } else ++it;
    }
    for (auto it=_List_iterator<...> = fonts.begin(); it != fonts.end();) {
        if (it->second.use_count() == 1) {
            LOG(type:INFO) << "Destroyed Resource<font>: " << it->first;
            it = fonts.erase(it);
        } else ++it;
    }
    for (auto it=_List_iterator<...> = sample_instance_pairs.begin(); it != sample_instance_pairs.end();) {
        if (it->second.first.use_count() == 1) {
            LOG(type:INFO) << "Destroyed<sample_instance>: " << it->first;
            it = sample_instance_pairs.erase(it);
        } else ++it;
    }
    // Stops playing samples whose instance isn't referenced.
    for (auto it=_List_iterator<...> = samples.begin(); it != samples.end();) {
        if (it->second.use_count() == 1) {
            LOG(INFO) << "Destroyed Resource<audio>: " << it->first;
            it = samples.erase(it);
        } else ++it;
    }
}
```

(ReleaseUnused只有在按X離開窗口，才會執行)

切換場景時將空閒不會用到的資源所占記憶體釋放，但未啟用，目前僅在按下x結束遊戲時才有。

```
185  void GameEngine::changeScene(const std::string& name) {
186      if (scenes.count(name) == 0)
187          throw std::invalid_argument(_Message:"Cannot change to a unknown scene.");
188      // Terminate the old scene.
189      activeScene->Terminate();
190      activeScene = scenes[name];
191      // Release unused resources.
192      if (freeMemoryOnSceneChanged)
193          Resources::GetInstance().ReleaseUnused();
194      // Initialize the new scene.
195      activeScene->Initialize();
196      LOG(type:INFO) << "Changed to " << name << " scene";
197  }
```

```
void GameEngine::Start(const std::string& firstSceneName, int fps, int screenW, int screenH,
    int reserveSamples, const char* title, const char* icon, bool freeMemoryOnSceneChanged, float deltaTimeThreshold) {
    LOG(type:INFO) << "Game Initializing...";
    // Update Allegro5 configs.
    this->fps = fps;
    this->screenW = screenW;
```

```
void Start(const std::string& firstSceneName, int fps = 60, int screenW = 800, int screenH = 600, int reserveSamples = 1000,
    const char* title = "Tower Defense (I2P(II)_2020 Mini Project 2))",
    const char* icon = "icon.png", bool freeMemoryOnSceneChanged = false,
    float deltaTimeThreshold = 0.05);
```

預設值是false

以GetBitmap為例:

```
std::shared_ptr<ALLEGRO_BITMAP> Resources::GetBitmap(std::string name) {
    if (bitmaps.count(name) != 0)
        return bitmaps[name];
    std::string bitmapPath = bitmapPathPrefix + name;
    ALLEGRO_BITMAP* bmp = al_load_bitmap(filename:bitmapPath.c_str());
    if (!bmp) throw Allegro5Exception(message:("failed to load image: " + bitmapPath).c_str());
    LOG(type:INFO) << "Loaded Resource<image>: " << bitmapPath;
    bitmaps[name] = std::shared_ptr<ALLEGRO_BITMAP>(bmp, al_destroy_bitmap);
    return bitmaps[name];
}
```

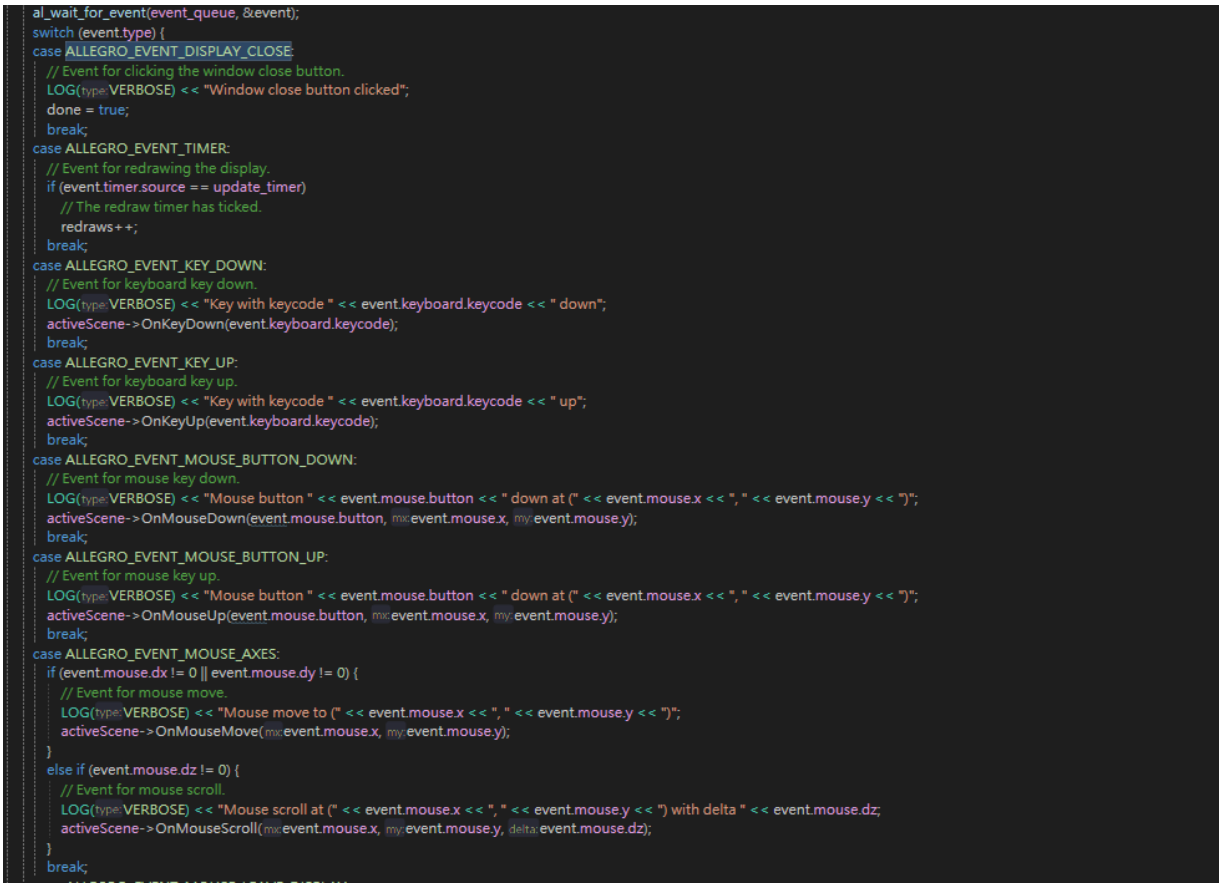
優點:簡單，方便。

缺點:GC時需所有記憶體跑遍，效率問題。

4. Crash Reports are designed to send logs with detailed information to the developer, allowing them to analyze and prevent the program from crashing again. If you want to implement such a system, how can Engine::LOG help?

```
bool LOG::Enabled = false;
bool LOG::LogVerbose = false;
const char* LOG::FilePath = "log.txt";

LOG::LOG(LogType type) {
    enabled = Enabled;
    this->type = type;
    if (canLog()) {
        ofs.open(FilePath, std::ofstream::app);//先把ofs設定為FilePath這個txt
        std::cout << '[' << getLabel(type) << "] ";//一個寫入黑底白字窗
        ofs << '[' << getLabel(type) << "] ";//一個寫入硬碟txt
    }
}

LOG::~LOG() {
    try {
        if (canLog()) {
            std::cout << std::endl;
            ofs << std::endl;
            ofs.close();
        }
    } catch (...) {}
}

bool LOG::canLog() const {
    return enabled && (type != VERBOSE || LogVerbose);
}

const char* LOG::getLabel(LogType type) {
```

```
        switch (type) {
        case VERBOSE: return "VERBOSE";
        case DEBUGGING:   return "DEBUGGING";
        case INFO:    return "INFO";
        case WARN:    return "WARN";
        case ERROR:   return "ERROR";
        }
        return "UNKNOWN";
    }
    void LOG::SetConfig(bool enabled, bool logVerbose, const char* filePath) {
        Enabled = enabled;
        LogVerbose = logVerbose;
        FilePath = filePath;
        // Clear log file content.
        std::ofstream ofs(FilePath, std::ofstream::out);
    }
```



```
al_wait_for_event(event_queue, &event);
switch (event.type) {
case ALLEGRO_EVENT_DISPLAY_CLOSE:
  // Event for clicking the window close button.
  LOG(type:VERBOSE) << "Window close button clicked";
  done = true;
  break;
case ALLEGRO_EVENT_TIMER:
  // Event for redrawing the display.
  if (event.timer.source == update_timer)
    // The redraw timer has ticked.
    redraws++;
  break;
case ALLEGRO_EVENT_KEY_DOWN:
  // Event for keyboard key down.
  LOG(type:VERBOSE) << "Key with keycode " << event.keyboard.keycode << " down";
  activeScene->OnKeyDown(event.keyboard.keycode);
  break;
case ALLEGRO_EVENT_KEY_UP:
  // Event for keyboard key up.
  LOG(type:VERBOSE) << "Key with keycode " << event.keyboard.keycode << " up";
  activeScene->OnKeyUp(event.keyboard.keycode);
  break;
case ALLEGRO_EVENT_MOUSE_BUTTON_DOWN:
  // Event for mouse key down.
  LOG(type:VERBOSE) << "Mouse button " << event.mouse.button << " down at (" << event.mouse.x << ", " << event.mouse.y << ")";
  activeScene->OnMouseDown(event.mouse.button, mx:event.mouse.x, my:event.mouse.y);
  break;
case ALLEGRO_EVENT_MOUSE_BUTTON_UP:
  // Event for mouse key up.
  LOG(type:VERBOSE) << "Mouse button " << event.mouse.button << " down at (" << event.mouse.x << ", " << event.mouse.y << ")";
  activeScene->OnMouseUp(event.mouse.button, mx:event.mouse.x, my:event.mouse.y);
  break;
case ALLEGRO_EVENT_MOUSE_AXES:
  if (event.mouse.dx != 0 || event.mouse.dy != 0) {
    // Event for mouse move.
    LOG(type:VERBOSE) << "Mouse move to (" << event.mouse.x << ", " << event.mouse.y << ")";
    activeScene->OnMouseMove(mx:event.mouse.x, my:event.mouse.y);
  }
  else if (event.mouse.dz != 0) {
    // Event for mouse scroll.
    LOG(type:VERBOSE) << "Mouse scroll at (" << event.mouse.x << ", " << event.mouse.y << ") with delta " << event.mouse.dz;
    activeScene->OnMouseScroll(mx:event.mouse.x, my:event.mouse.y, delta:event.mouse.dz);
  }
  break;
```

5. Try-Catch statements can intercept the error thrown by the inner code. Why are Try-Catches preferred instead of error codes in modern programming? List out five advantages of the Try-Catch statement.

5.

Code更加簡單、乾淨、不會錯過錯誤。較易除錯，傳統的error codes容易使程式雜亂無章，且部分狀況無法由error code處理。

Imagine that we did not have exceptions, how would you deal with an error detected in a constructor? Remember that constructors are often invoked to initialize/construct objects in variables:

1. vector<double> v(100000); *// needs to allocate memory*
2. ofstream os("myfile"); *// needs to open a file*

The `vector` or `ofstream` (output file stream) constructor could either set the variable into a “bad” state (as `ifstream` does by default) so that every subsequent operation fails. That’s not ideal. For example, in the case of `ofstream`, your output simply disappears if you forget to check that the open operation succeeded. For most classes that results are worse. At least, we would have to write:

```
vector<double> v(100000); // needs to allocate memory
if (v.bad()) { /* handle error */ } // vector doesn't actually have a bad(); it relies on exceptions
ofstream os("myfile"); // needs to open a file
if (os.bad()) { /* handle error */ }
```

That’s an extra test per object (to write, to remember or forget). This gets really messy for classes composed of several objects, especially if those sub-objects depend on each other

易於使用

可在例外發生時加入相對應的措施，使程式能正常結束

允許我們拋出例外

不會拖慢執行速度

增進程式的穩定性及效率

6. There are multiple std::binds and std::functions in the template code. What are them for? How do they work? Also, elaborate on the difference between a normal function pointer and a member function pointer.

7. Factory Method can be seen as a virtual constructor while Template is a feature for classes to deal with generic types. Which concept suits better to simplifying the instantiation of different turrets? Furthermore, which of them can achieve constructing and deconstructing each scene on scene change? (In the current template we keep a single instance of each scene for simplicity)