A ) Easy Question

|  |  |
| --- | --- |
| 1 | 把功能與資料包裝起來成為一個物件，並定義物件的介面。讓外界的程式，透過物件所提供的介面，來與這個物件溝通。外界的使用者，無須知道，物件內部如何執行他的作業。只需要知道，如何使用物件的介面，來完成自己的工作即可。所以，設計物件的開發者，日後要更新物件的程式 時，只要維持物件的介面不變，便可以確保，使用這物件的外部程式，不 需要改寫，一樣可以執行。而外界程式的開發者，只要遵循不直接存取物件內部的功能及資料，僅透過介面來使用物件的規則，便可以確保程式執行無誤。 |
| 2 | 繼承(Inheritance)，就是子類別藉由繼承父類別的方式，取得父類別 所定義的屬性及方法。通常，父類別會定義通用的屬性及方法，而子類別 則延伸父類別定義特定的屬性及方法，以解決特定的問題。 |
| 3 | 簡單來說就是相同名稱的方法(Method)，多個相同名稱的方法，傳入不同 的參數，會執行不同的敘述。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 | 在procedural programming中，我們是寫入各種指令或function去讓電腦一步一步地做。OOP則是將各種數據或行為包裝成object。程式易維護、程式的重複使用、程式易擴展 |
| 5 | Constructor 需要、Deconstructor 需要、overridden virtual Function 一般而言不需要。 |
| 6 | 快速、方便、省時、直覺；OOP |
| 7 | 中斷點、呼叫堆疊。原場景 Group 裡的東西提早 delete 掉。導致後續要用 Group 裡 delete 時重複 delete 導致錯誤。 |
| 8 | Astronomia.ogg；抬棺黑人又浮現等你了。 |
| 9 | 按下視窗右上 x 。game.Start => startEventLoop() => while(!done) 負責檢查若按下則跳出結束。 |

B ) Medium Question

|  |  |
| --- | --- |
| 1 | 把 map 裡每列全數讀入 mapData，同時檢查設定的地圖規格 mapData.size()) / MapWidth != 0 以及 mapData.size()) != MapWidth \* MapHeight，接著再 mapState[i][j]= mapData[i \* MapWidth + j]並設定每一格的物件 Engine::Image。  逐行讀入敵人類型、間隔、數量，同時按照數量重複做 enemyWaveData.emplace\_back(type, wait)建立敵人 list 。PlayScene::Update 裡的 for (int i = 0; i < SpeedMult; i++) 會 將 ticks 加上 deltaTime，如果 ticks 小於敵人的間隔時間則跳過，反之則生成敵人，並將 ticks 減掉敵人的間隔時間，同時把該敵人個體從 enemyWaveData 刪除。 |
| 2 | PlayScene::Update 裡利用 for-loop 以 SpeedMult 次數作條件，進行更新 ;  可以放turret ; 因為SpeedMult==0的時候只是讓enemy和射子彈動作暫停，但Update還是會繼續，所以還是可以放置turret。 |
| 3 | 各 object 的管理及更新較為複雜且麻煩，類似上學期每個 object 都要各自處理，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。 |
| 4 | 因為當呼叫changescene，會把原本scene中的所有東西都delete掉。如果這時候再去對原本的scene做update，就會因為找不到被delete掉的東西而出錯 |
| 5 | Deltatime 為更新動作間時間差，當作 function 的 parameter 可利用繼承特性將所有繼承而來的物件運算做更新，可直接執行而不必每個都還要算時間差；將 threshold 移除可能導致因電腦本身執行效能不佳，導致 Deltatime 過長，以致後面的運算出來的值過大，也可避免 bullet- through-paper problem。 |
| 6 | 理論上可以(?)；優點: 直覺、不易出錯。缺點:應該沒有吧。 |
| 7 | List 的大小是動態的，可隨不同狀況下做大小變動。直接用變數儲存的話沒有彈性，不同場景的所需的 object 數目可能有所不同。 |
| 8 | 看情況。優點:父類別跟子類別各自的 member function 跟 data 明確， 不容易混淆誤用，且 image 是很多其他 class 的 base class，缺點:繼承多一層。 |
| 9 | 不是；Win 畫面終止時會呼叫 Terminate function 利用 base class(WinScene =>IScene=>Group )寫好的 Clear() 逐一 delete 釋放掉那些東西。那就在用完的時候把他delete掉。 |

C ) Hard Question

|  |  |
| --- | --- |
| 1 | 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；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 |
| 2 | 對；因為pointer可以是nullptr，在function中接收參數時就可以選擇要不要接收物件，讓程式彈性更大。   1. 使用某些C的函式庫需要接收pointer。 2. 可以不用先知道要宣告的size 3. Function pointer |
| 3 | 做的就是類似Garbage Collection的功能，但這邊的作法並沒有那麼複雜，只有在場景切換時將GameEngine中未使用的resources 釋放。  好處是可以避免記憶體浪費，壞處是有可能會刪掉之後需要使用的物件。 |
| 4 | 將大部分的動作切細都輸出 LOG，了解執行的順序或執行到哪裡時發生錯誤，記錄使用者的行為。 |
| 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。 |
| 6 | Std::bind配合std::function可以方便儲存以及呼叫function；std::bind 回傳一個function object；Std::function可以儲存任意種函式，但function pointer只能儲存固定型態的function |
| 7 | Factory method；Template method； |

* Raw Pointers & Smart Pointers
  + Raw Pointer：原本寫C用的 \*ptr都是raw pointers
  + Smart Pointers：用c++11之後有的unique\_ptr，shared\_ptr，weak\_ptr