Week15

a. Explain how inheritance helps us avoid code duplication.

继承允许我们将多个类之间的公共功能放入一个父类(基类)中，然后从该父类创建子类。这就像一种模板，避免了我们被迫多次编写完全相同的代码(代码重复)，让子类可以直接继承这些公共代码，而无需重复编写。

b. Why does it make sense to use a Player class object wherever an Entity class object is used?

因为 `Player` 是 `Entity` 的子类，这意味着 `Player` 实际上同时拥有 `Player` 和 `Entity` 两种类型。`Player` 保证拥有 `Entity` 所拥有的一切(是一个超集)，甚至可能包含更多内容，因此在任何需要 `Entity` 对象的地方使用 `Player` 对象是合乎逻辑的。

c. What is the purpose of using virtual functions?

虚函数的主要目的是允许我们在子类中重写(Override)方法。通过引入动态分发(Dynamic Dispatch，通常通过虚函数表 V-table 实现)，虚函数让 C++ 能够在运行时意识到传入的基类指针实际上指向的是一个派生类对象，从而调用正确的重写函数。

d. How to use the override keyword and what are the associated benefits?

用法：在派生类中重写函数的声明后添加 override 关键字。

好处：

1. **可读性**：它让代码更具可读性，明确表示这是一个被重写的函数。
2. **防止错误**：它有助于避免 bug。例如，如果你拼写错了函数名，或者试图重写一个在基类中未标记为 `virtual` 的函数，编译器会报错。

e. Why can't you instantiate an object of an interface class containing pure virtual functions?

你无法实例化这样的类，因为纯虚函数(定义为 `= 0`)在基类中没有具体的实现(即没有函数体)。由于该类包含未实现的方法，它只是一个模板，因此无法创建该类的具体实例。

f. When can you instantiate an object of a class derived from an interface class with pure virtual functions?

只有当派生类(或层级结构中更上层的类)实际实现了所有的纯虚函数时，你才能实例化该对象。如果这些函数没有被实现，你就无法实例化该派生类。

g. Describe the visibility of private members.

`private` 成员只能被定义它们的那个类(以及友元)内部访问，类可以对其进行读写。子类(派生类)或类外部的代码(如 `main` 函数)都无法访问私有成员。

h. Describe the visibility of protected members.

`protected` 成员比 `private` 更可见，但比 `public` 低。它们可以被定义它们的类以及所有继承层次中的子类访问。但是，它们对于类外部的代码(如 `main` 函数)仍然是不可见的。

i. Explain the subtle difference between a virtual member function and a virtual destructor.

对于普通的虚函数，子类的实现会“重写”基类的函数，运行时会映射到具体的那个函数。而对于虚析构函数，并不是单纯的替换，而是确保“两者”都被调用：它会先调用派生类的析构函数，然后沿着继承层次向上调用基类的析构函数。

j. When is there a need to declare a destructor of a class as virtual?

当你允许一个类被继承(子类化)，并且可能通过基类指针来处理派生类对象(例如使用 `new Derived()` 赋值给 `Base*` 然后 `delete Base*`)时，必须将析构函数声明为 `virtual`。如果不这样做，删除对象时只会调用基类的析构函数，而不会调用派生类的析构函数，从而导致内存泄漏。