Week 9 – Session 2: Inheritance and composition

In laboratory session one earlier in the week you implemented the `Item`, `Weapon` and `Armour` class. Now we will integrate this with the `GameCharacter` class. If you look at figure 1 you will see there is composition between the `GameCharacter` and the item. Each game character has a weapon and some form of armour.

These are implemented as objects. So, a `GameCharacter` is composed of two additional objects (a weapon and an armour object).

Challenge 1: Implement the `GameCharacter` base class

Examine the class hierarchy below in figure 1.

Task 1 Declare the `GameCharacter` class - duration 15 minutes

Using the solution that you created in the previous class create a `GameCharacter` class. You can quickly do this by Right clicking on the project and selecting Add -> Class. VStudio will create the header file (.h) and the definition file (.cpp) for you.

Once you have added the `GameCharacter.h` and `GameCharacter.cpp` files, complete the declaration of the `GameCharacter` class in the header file as shown in the class diagram below.

Since we have lots of character class to derive, you can use the skeleton code available in my repository (https://github.com/eiramlan/RPG). This should significantly reduce your typing time.

Task 2 `GameCharacter` class constructors - duration 10 minutes

Implement the `GameCharacter` class constructors in the `GameCharacter.cpp`.

Remember when you create an object you should strive to ensure they all objects have default values of some sort. In the default constructor give the game character a name and some default health, weight limit, food and state values. This applies to your `Weapon` and `Armour` default constructors. We will get to that in the other challenges.

The constructors for the `GameCharacter` should also display a message saying:

```
A Game Character called <CharacterName> has been born
```

Task 3 Implement the Getters and setters - duration 15 minutes

Implement the `GameCharacter` class getters and setters in the `GameCharacter.cpp`.

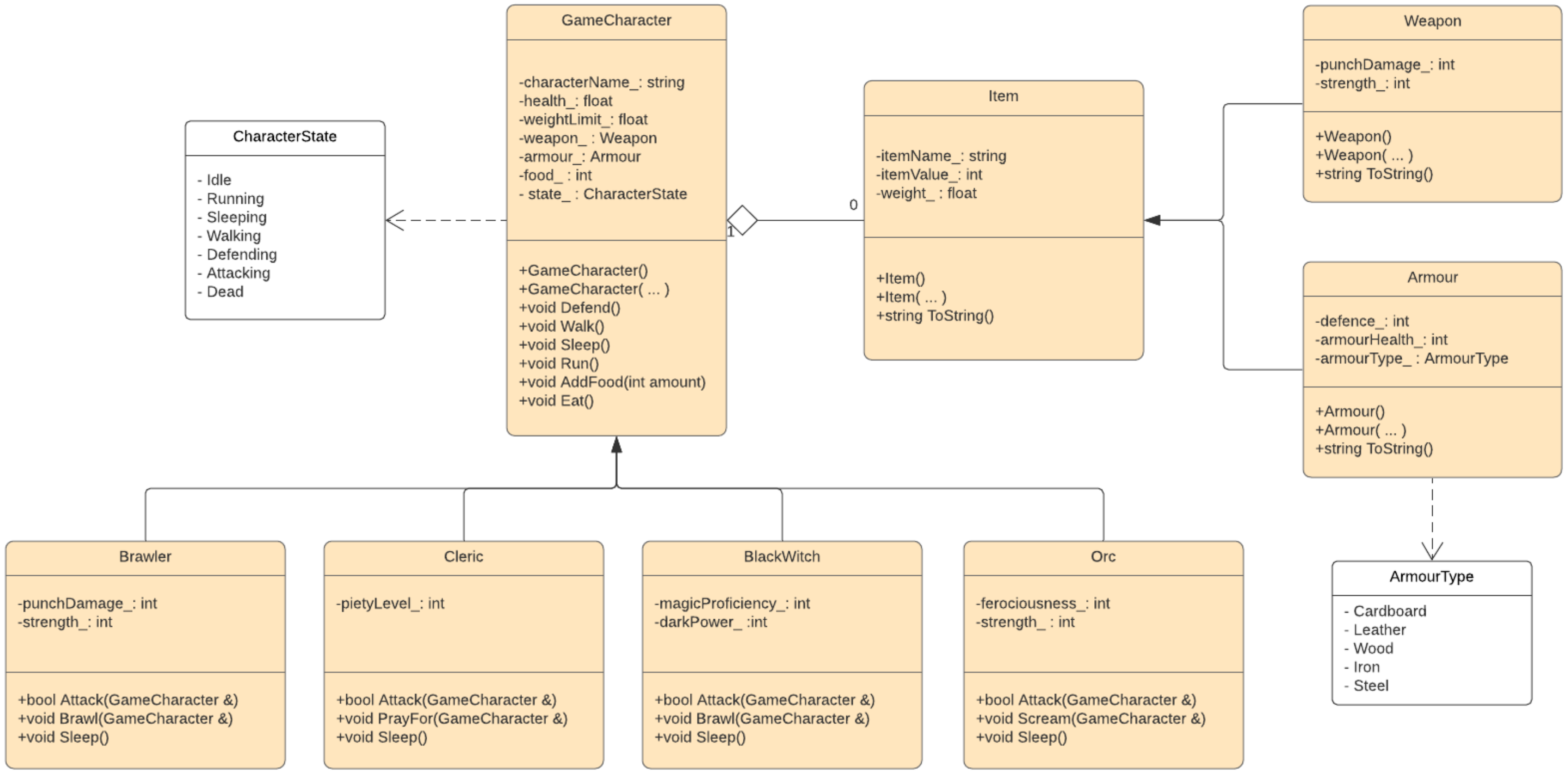

Figure 1: UML class diagram to complete

Task 4 Implement remaining function- duration 40 minutes

The `GameCharacter` class has a number of additional functions you need to implement. These are very simple, and the desired functionality is listed below:

- **`AddFood()`** – Add the amount of food to the characters total
- **`Eat()`** – Consume four units of food – If you have that many. For each unit of food consumed add 0.25 units to health. The value of food_ should not be < 0.
- **`Attack()`** – The function should display a message saying:

```
<CharacterName> is attacking <OtherCharacter> with a <WeaponName>
```

The function should also change the character's state to the Attacking state
For some reason, we want to return the Boolean value true.

`Defend()` – The function should display a message saying:

```
<CharacterName> is defending themselves with a <ArmourName> made
from <ArmourType>
```

`Walk()`,`Run()` and **`Sleep()`** – Print a message saying the

```
<CharacterName> is <state>
```

and change the state of the character to sleeping, walking or running accordingly.

Challenge 2: Implement a derived character class

Now that you have a base `GameCharacter` class implemented you can use this foundation to create several custom game character classes that extends the behaviour of the base class.

Task 1 Declare a derived `GameCharacter` class - duration 10 minutes

Select one of the derived character classes from figure 1 and add a corresponding .cpp and .h file.

Once you have added the .h and.cpp files, complete the declaration of the for the specified class in the header file as shown in the class diagram above. Feel free to add new attributes if you wish.

Task 2 Derived class constructors - duration 10 minutes

Implement the class constructors in the .cpp file for your custom character.

Remember when you create an object you should strive to ensure they all objects have default values of some sort.

Task 3 Implement the Getters and setters - duration 15 minutes

Implement the class getters and setters in the .cpp.

Task 4 Implement remaining function- duration 20 minutes

The derived character class has a number of additional functions you need to implement. These are very simple, and the desired functionality is listed below:

For all

`Attack``()` – Should print a message "`Class Type <CharacterName> is attacking <CharacterName> with a <WeaponName>`", set the status to Attacking, and return `true`

`Sleep()` – Add 15% health to the character. Print out the message "`<CharacterName> is sleeping`"

`Brawler::Brawl()` – Should print a message "`Brawler  <CharacterName> is brawling <CharacterName>`"

`Cleric::PrayFor()` – Should print a message "`Cleric <CharacterName> is praying for <CharacterName>`"

`BlackWitch::Bewitch()` – Should print a message "`Black witch <CharacterName> is attempting to bewitch <CharacterName>`"

`Orc::Scream``()` – Should print a message "`Orc <CharacterName> is screaming at <CharacterName>`"

Additional challenge: Build an RPG battle system

The classes you have implemented here form the basis of a potential RPG game. If you refer to the main RPG function, the skeleton to play the game has been initialised. You should extend this by adding logic to the classes for the attack and defend functions. These additions will make the game more realistic.

Most of the times, you just need to change the base class functions, since your derived might be calling those functions as well. There are also specific functions allocated for your derived character, so changes must be made only in the character classes.

Character Attack logic

The character attack logic should work as follows:

- The character cannot attack another character if:
    - They do not have a weapon equipped (exception for brawler)
        - Brawl function called if no weapon equipped
    - Their health is <= 20;
    - The character being attacked is dead
- If the hit strength of the weapon used by the attacking character is less than the defence value of the armour of the defending character, then there is only a 20% chance of the attack being successful. Otherwise there is a 60% chance of a successful attack.
- If the defending character is not wearing any armour, then the chance of a successful attack increases to 80%.
- An unsuccessful attack will result in no damage on the character being attacked. However, the health of the weapon used by the attacking character will be reduced by a random value between 10% and 20% if the defender is wearing armour of any type.
- Weapons with health <=0 should be removed from the inventory.
- If the attack is deemed **successful** it will inflict damage on the defending character as follows:
    - – 10% health to the character being attacked if their state is Defending
    - – 100% health to the character being attacked if their state is Sleeping
    - No effect if the character being attacked state is Dead
    - – 20% health to the character being attacked for all other states.

    - The health of the defending characters armour will also be reduced by 10%
    - Armour with health <=0 should be removed from the inventory.

Feel free to improve the game with your own logic. For instance, special functions (`Brawl()`, `PrayFor()`, `Bewitch()` and `Scream()`) can also inflict damage to the other characters. You can implement the logic yourself.

You can also design the dice rolling to affect the other elements of the game play. Feel free to share the link of your project in Discord. I would love to check your repository and see what you have managed to produce.