**Chapter 7 – Fight Part 2**

With the main damage that that the hero takes being complete we have to process the status effects of the hero and then conclude with player inputs. Make sure you have completed part 1 of the Fight() function as the creation of status effects is explained in depth. Without delay, we shall go straight into the second phase.

Processing of Status Effects

The section is going to focus on working on the ‘heroStatus’ list. We will check the list in during every round of the fight and update effects by decreasing their turn timer by 1 and adding the effect they produce.

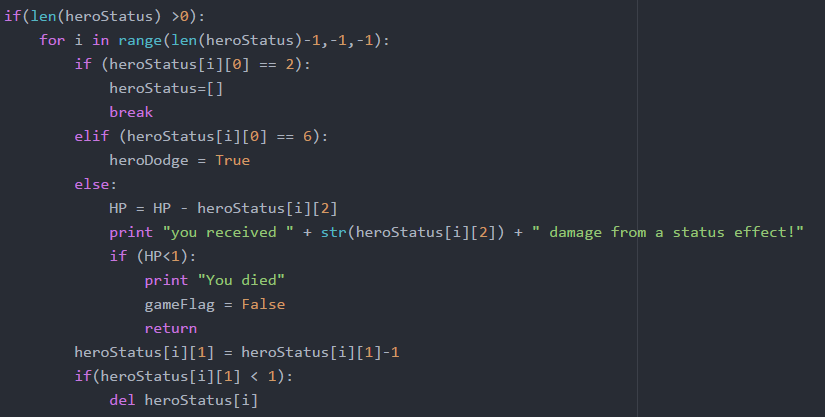
Currently ‘heroStatus’ is a list containing lists of size 3. Each sublist has the information: effect ID, effect duration, effect damage. The first step to updating all our status effects is to fonding out how many sublists are in ‘heroStatus’. Python has a function called len() which does just that. So if 3 effects are active then len(heroStatus) will give 3. Once we find the length, we want to iterate backwards through the effects, starting with the most recent first(reasons why will be explaned further on). To do this simply create a for loop with the starting number being the highest index(length minus 1) and the lowest number being -1 and our step interval also being -1. If everything is set up correctly, you should get this:

Screen Clipping

Now we should think about the possible status effects we have and there are three categories they can be split into: effects that do damage to the player, effect that removes all effects from the player and an effect that gives them dodge status.

It is wise to resolve status effects in this order: cleanse->dodge->damage. The reason we do this is so that we don’t want a player to use a cleansing potion but they die to a damage effect because we resolved it first. This is also precisely why we work through the list backwards so that the most recent effect is resolved first.

The first effect to resolve has ID value 2 and in our ‘heroStatus’ list, the ID is the first element of the sublist. Therefore we would check the value heroStatus[i][0] (where i is our iterator). If it is a 2, we replace the status list with a new empty list. If it is a 6 we set the hero dodge status to true. Anything else would be a damage effect so we have to do something a little extra. We must subtract the damge from the health but also print a message if they die from HP<1. At this point we follow stand procedures to exit the Fight() function by making our ‘gameFlag’ false and then using return.



We then added a small section at the end where every time we iterate through an effect, we decrease the duration number by 1 and then if the duration number is 0, we delete the status effect. That concludes the processing of status effects. We must remember to use str() to convert the number to text.

Processing of inputs

This section is considerably longer that the last but if we are smart about it, we can reuse a lot of the structure once we have created it. This part can be split into the hero attacking, the hero defending or the hero using an item. Before we do any of this we will first display the HP and simply call our PrintInventory() function.

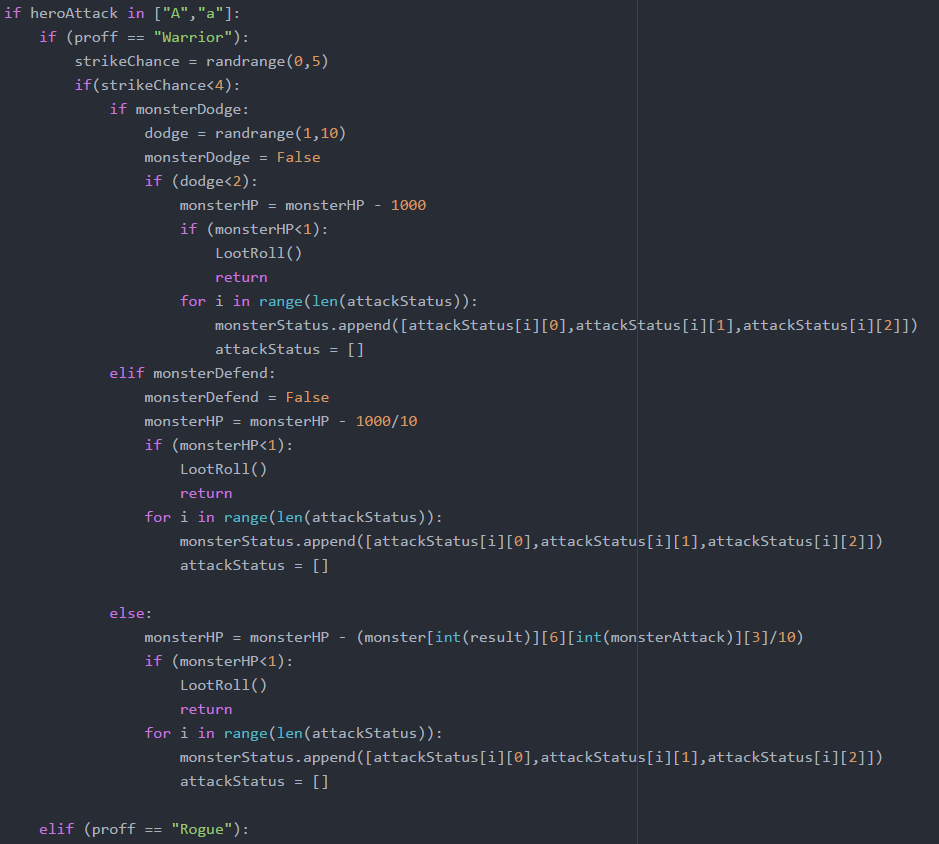
Screen Clipping

Next we have to implement the attack part of the battle. At this point you should have an idea on what types of attacks the different classes will have. In my example our warrior will have a high damage, single strike attack with good chance of hitting. Our rogue on the other hand will have a multiple strike attack(10) which individually do less damage and have a higher chance of missing. This means the warrior has good damage when landing a strike but has the possibility of dealing no damage at all where the rogue can afford to miss but will output lower total damage.

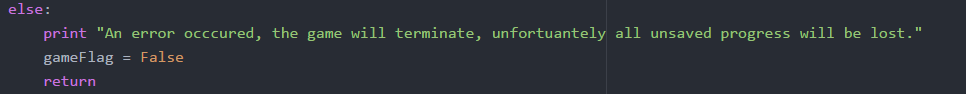
Screen ClippingBefore we can process inputs we need to know what the user wants to do and therefore we need a raw\_input() to find out:

This is the first part where we reuse old code. The major difference here will be we will have an ‘attackStatus’ this time for any effects that the player adds to the attack. We will then take items in the ‘attackStatus’ and add it to the ‘monsterStatus’. Another modification we will make is that when the monster HP reaches 0, we can call the LootRoll() function and then use return to exit. So if we take everything mentioned into account we will get the following:

(Note: the ‘attackStatus’ and ‘monsterStatus’ will become more clear after we cover using items)

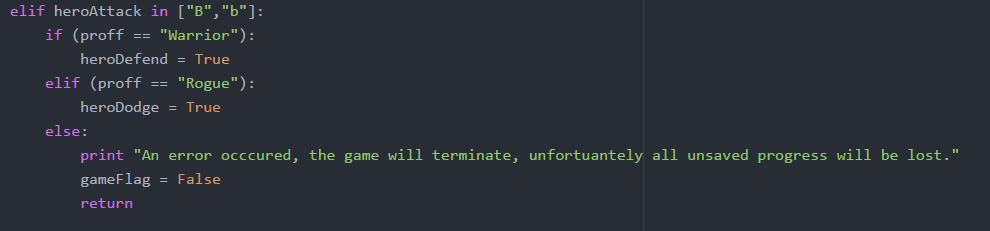


Everything above should be familiar appart from the resolution of the status effects but don’t worry, we will explain that later on this chapter. Rogue will follow a similar structure with the only difference being a for loop that loops 10 times and the damage value of the rogue being less. Therefore it won’t be included in this document again but you can look at the example files if you do want to have a look at it.

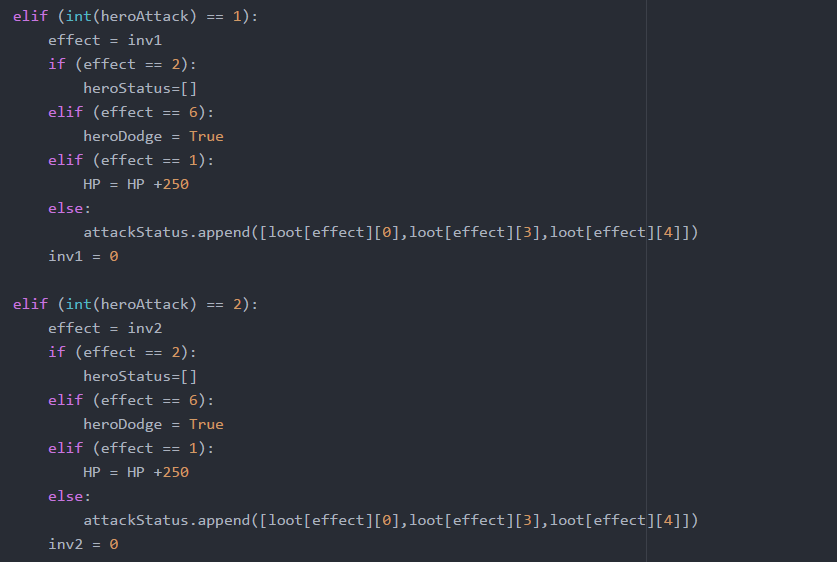


We also added an extra error message at the end of this just in case there is a problem with the roles but theoretically this should never appear if every role is covered properly. This is here for emergencies as we want the player to know if something does go wrong.

The next input will be defence and for our two classes we made them have different effects. If you are a warrior you get a defend option which means you block a percentage amount but if you are a rogue you get a dodge option which can avoid all damage but can also fail.



We are so close to completing our Fight() function and completing the whole game but there is still one final thing to process – items.

If our user wants to use an item from the inventory, the must input a number from 0 to 9. We are going to just show one inventory slot in action as they will also be the same and you only have to change variable when copying it 9 times. This is where we will also explain how the ‘attackStatus’ will be used. This is how the first two inventory slots look like:

Brief explanation: Item 2 cleanses all effects, item 6 adds dodge effect, item 1 adds HP to the character. All other items are offensive stats and we need to put them into the attack status. We simply add the item ID, damage and duration.

The effect is only applied once the player attacks, hence why we had the following line earlier:

Screen Clipping

Once the attack is made, we remove the effect and our final step is to resolve the monster statuses. Luckily it follows the same format as ‘heroStatus’ but with changes so that the outcomes are sensible:



That is the end of our fight function and we are now 1 chapter away from having a complete game. We have to bring our game together to produce a main screen where the player can choose to save or load a game as well as automating the room generation.

EXTENSION

Have you considered having spells in your game? These spells would using a separate resource such as mana or energy. Different classes can have different spells such as a Mage class that can heal themselves which is perhaps stronger than a potion. Another alternative is a fireball like spell which adds a damage over time debuff. The possibilities are endless but can be tricky to implement.

Also if you have the time, you can include ASCII art for these spells too!