Crafted Weapon Powder Mechanics

ferricles / hppeng

27 March 2021

1 Introduction

Powder application mechanics on weapons in Wynncraft have been unclear ever since the 5-element powder system was conceived. Even with tooltips for damage conversions displaying on powder items, the real algorithm that runs when applying powders to weapons is slightly more complicated. The algorithm for applying powders to Crafted weapons, however, is more complex. It is a mix of the normal powder application algorithm and the one for applying powders as ingredients on Crafted Weapons.

For those who happen to know how both work, the algorithm for powder applications on Crafted Weapons can be expressed as:

```
(1) Apply all applied powders as ingredients.

(2) Apply all applied powders as powders on normal weapons.
```

This paper is meant to help those without knowledge of either understand the mechanics for powder application on crafted weapons (from here on out referred to as "powdering crafted weapons"). To do so, we must first understand two sub-algorithms: (1) powder application on non-crafted weapons and (2) powder application as ingredients for crafted weapons.

2 Convention

Powders have a "conversion factor" and a "constant factor." For example: The Earth V powder shown below has a conversion factor of 38% and a constant factor of 15-18 (minimum 15, maximum 18).

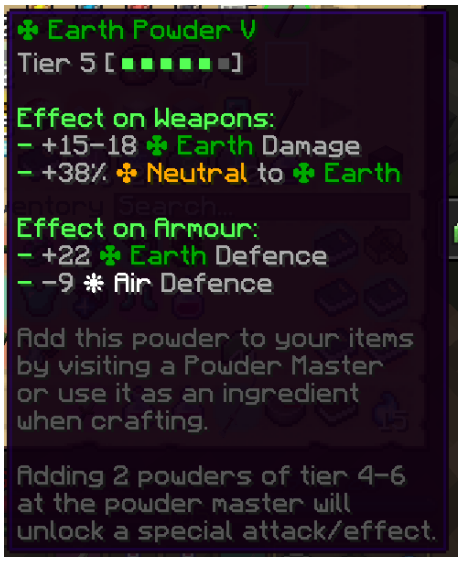

Figure 1: The Earth V powder.

Weapons have multiple different "base damages." There is base damage unpowdered, base damage powdered, base damage for spells, etc. In this paper, we will need to use unpowdered base damage and powdered base damage. We will explore how powders change unpowdered base damage into powdered base damage in full detail.

In this paper, `:=` will denote assignment of the value/expression on the right hand side to the (typically a variable) left hand side.

3 Powders on Non-Crafted Weapons

On non-crafted weapons, powders apply sequentially according to their conversion and then constant factors. The algorithm is as follows.

Let `[N1, N2]` be the unpowdered weapon's neutral damage range, `[n1, n2]` be the current neutral damage range, and `[x1, x2]` be the current elemental damage of the powder's element.

For each powder in the powder array:

```
(1) [low, high] := min(remaining neutral dmg, conversion factor * [N1, N2])
(2) [n1, n2] := floor([n1, n2] - [low, high])
(3) [x1, x2] := floor([x1, x2] + [low, high])
(4) [x1, x2] := [x1, x2] + [powder min, powder max]
```

Note: If the conversion `[low, high]` is greater than the remaining neutral damage, then it is lowered to the remaining neutral damage.

4 Powders as Ingredients

On crafted weapons, powders applied as ingredients apply sequentially using a different algorithm. The order is the English-reading order: top left, top right, middle left, middle right, bottom left, bottom right.

There are two crucial mechanics to understand about crafted weapons. The first is that their neutral damage range **always** follows a 10% variation from a baseline roll after attack speed and material tiers are taken into account. For example, if this post-processing baseline roll is 100 and powders were not used as ingredients, then the neutral damage on the weapon would be [90, 110] ([floor(100*0.9) = 90, floor(100*1.1) = 110]). We will cover the second important mechanic after seeing the algorithm, which is as follows.

Let `n` be the current neutral damage base, `x` be the current elemental damage base of the powder's element, and `min, max` be the minimum and maximum values of the powder's constant factor.

For each powder in the powder array p:

```
(1) conversion = floor(conversion factor * n)
(2) n := n - conversion
(3) x := x + conversion
(4) x := x + floor((min + max)/2)
```

The second important mechanic is that powders applied in the recipe apply as ingredients AFTER powders applied while powdering the crafted item. For example, suppose we had a weapon with a recipe of 1 tier 3 fire (F3) powder and 1 tier 2 water (W2) powder included in the crafting recipe. With the finished crafted weapon, we applied 1 tier 6 earth (E6) and 1 tier 5 thunder (T5) powder on. The powder array for this weapon would be [E6, T5, F3, W2] because the post-crafting powders apply as ingredients before the recipe powders.

The powder as ingredient mechanic works by modifying the base values of damages. Notice how it is theoretically impossible to reach 0 neutral damage when using powders as ingredients, as conversions apply

to the remaining neutral damage instead of the unpowdered neutral damage.

After this algorithm runs, the crafted weapon will have base damages of [n, e, t, w, f, a]. The actual damage ranges that display will be of the form:

```
(1) Neutral := [floor(0.9n), floor(1.1n)]
(2) Earth := [floor(0.9e), floor(1.1e)]
(3) Thunder := [floor(0.9t), floor(1.1t)]
(4) Water := [floor(0.9w), floor(1.1w)]
(5) Fire := [floor(0.9f), floor(1.1f)]
(6) Air := [floor(0.9a), floor(1.1a)]
```

5 Powdering Crafted Weapons

As mentioned earlier, the basic algorithm for powdering crafted weapons is:

```
(1) Apply all applied powders as ingredients.
(2) Apply all recipe powders as ingredients.
(3) Use base values to get a full damage range.
(4) Apply all applied powders as powders on normal weapons.
```

In more detail, the algorithm is:

- For each powder in the powder array:

```
(1) conversion := floor(conversion factor * n)
(2) n := n - conversion
(3) x := x + conversion
(4) x := x + floor((min + max)/2)
```

- For each damage type (n, e, t, w, f, a):

```
(1) Neutral := [floor(0.9n), floor(1.1n)]
(2) Earth := [floor(0.9e), floor(1.1e)]
(3) Thunder := [floor(0.9t), floor(1.1t)]
(4) Water := [floor(0.9w), floor(1.1w)]
(5) Fire := [floor(0.9f), floor(1.1f)]
(6) Air := [floor(0.9a), floor(1.1a)]
```

- Damages are now in format [n1, n2], [e1, e2], etc. We can proceed to applying each powder applied as a powder on a non-crafted weapon.

- For each powder in the powder array:

```
(1) [low, high] := min([n1, n2], conversion factor * [N1, N2])
(2) [n1, n2] := floor([n1, n2] - [low, high])
(3) [x1, x2] := floor([x1, x2] + [low, high])
(4) [x1, x2] := [x1, x2] + [powder min, powder max]
```

In the second stage of the algorithm, ONLY the post-crafting powders will apply as normal powders. The recipe powders ONLY apply as ingredients (as they should).

6 Worked Example

The algorithm is pretty confusing to get through without a full example, so we will examine a fresh crafted weapon to that end. We start with this crafted weapon:

Figure 2: Combusting Dirk of the Defender

For the purposes of this paper, we will not go through how these base damages were calculated. What we need to know is that the base neutral damage is 434 before ingredient powders were applied. For the curious, 1 tier 3 fire powder was applied as an ingredient during crafting for a level 100-102 Weaponsmithing recipe. Only tier 1 materials were used.

Again, our unpowdered base neutral damage (n) is 434. We will first apply 1 tier 3 thunder (T3) powder, 1 tier 3 fire (F3) powder, 1 tier 3 thunder (T3) powder, and 1 tier 3 fire (F3) powder, in that order specifically.

The T3 powder has a conversion factor of 13% and a constant factor of [2,18]. The F3 powder has a conversion factor of 19% and a constant factor of [6,10].

Starting with the first sub-algorithm (powder as ingredient):

1. Apply powders as ingredients. Because one F3 powder was used in the recipe, the powder array for this step is [T3 F3 T3 F3]

   - Thunder 3 (I)

     ```
     (1) n = 434, t = 0, f = 0
     (2) floor(0.13 * 434) = 56
     (3) n := n - 56 = 378
     (4) t := t + 56 = 56
     (5) t := t + floor((2+18)/2) = 66
     ```

   - Fire 3 (I)

     ```
     (1) n = 378, t = 66, f = 0
     (2) floor(0.19 * 378) = 71
     (3) n := n - 71 = 307
     (4) f := n + 71 = 71
     (5) f := f + floor((6+10)/2) = 79
     ```

   - Thunder 3 (II)

     ```
     (1) n = 307, t = 66, f = 79
     (2) floor(0.13 * 307) = 39
     (3) n := n - 39 = 268
     ```

```
(4) t := t + 39 = 105
(5) t := t + floor((2+18)/2) = 115
```

- Fire 3 (II)

```
(1) n = 268, t = 115, f = 79
(2) floor(0.19 * 268) = 50
(3) n := n - 50 = 218
(4) f := f + 50 = 129
(5) f := f + floor((6+10)/2) = 137
```

2. That's the first round of powder application. We then convert base damage to a range. We grab the bases from the last round of calculation: n = 218, t = 115, f = 113, e = w = a = 0.

```
(1) [n1, n2] := [floor(0.9n), floor(1.1n)] = [196, 239]
(2) [t1, t2] := [floor(0.9t), floor(1.1t)] = [103, 126]
(3) [f1, f2] := [floor(0.9f), floor(1.1f)] = [123, 150]
```

3. Apply powders as if they were applying on a non-crafted weapon. The F3 powder used during crafting doesn't apply, so our array is [T3 F3 T3].

- Thunder 3 (I)

```
(1) .13 * [196, 239] = [25.48, 31.07]
(2) n := floor([196, 239] - [25.48, 31.07]) = [170, 207]
(3) t := floor([103, 126] + [25.48, 31.07]) = [128, 157]
(4) t := [128, 157] + [2, 18] = [130, 175]
```

- Fire 3

```
(1) floor(.19 * [196, 239]) = [37.24, 45.41]
(2) n := floor([170, 207] - [37.24, 45.41]) = [132, 161]
(3) f := [123, 150] + [37, 45] = [160, 195]
(4) f := [160, 195] + [6, 10] = [166, 205]
```

- Thunder 3 (II)

```
(1) floor(.13 * [196, 239]) = [25.48, 31.07]
(2) n := floor([132, 161] - [25.48, 31.07]) = [106, 129]
(3) t := floor([130, 175] + [25.48, 31.07]) = [155, 206]
(4) t := [155, 206] + [2, 18] = [157, 224]
```

4. Our final damages are:

```
(1) Neutral = [106, 129]
(2) Fire = [166, 205]
(3) Thunder = [157, 224]
```

Here are the actual results:

Figure 3: Powdered Combusting Dirk of the Defender

The results match the calculations!

7 Conclusion

I hope you learned something from this paper.

Credits:

- hppeng/Deco113 - brainstorming the original idea for how powders might apply on Crafted weapons.
- nbcss - providing the algorithms for damage calculation when crafting powdered weapons.
- nbcss - providing and reverse-engineering the powder as ingredient algorithm.
- ferricles - testing and confirming the final algorithms for powders on normal weapons and powders (not as ingredients) on crafted weapons.