THE MATHEMATICS AND OPTIMIZATION OF OSRS COMBAT

Palfore

14th July, 2020

Contents

Introduction

Oldschool Runescape is a MMORPG which features combat. In combat, players are awarded experience for damage dealt to their opponents (often called monsters). After a certain amount of experience is gained the player may level up, improving the abilities in combat. The mechanics of combat involve accuracy, maximum damage, attack speed, among other details. There have been several attempts to quantify the experience rates according to a player's, and their opponent's levels and equipment. Although they are quite accurate, there are additional corrections that can be applied. Additionally extensions to more complex fighting scenarios has not been performed. We will be specifically considering applications to the Nightmare Zone, a common method of training combat. This game is played with long term investments and considerations in mind, as many achievements are achieved over periods of months or even years. In this sense, even minor optimizations can result in real-world savings for the player. Furthermore, the codebase that accompanies this article can be found at: https://github.com/Palfore/OSRS-Combat. This allows for more advanced use of the derived formulas (such as in optimization problems).

Part I

Modeling Combat

Chapter 1

Basics

The official Wiki page provides a very good review of maximum hit mechanics. For completion we will provide a brief summary. For some reason, it does not contain information about accuracy calculations. These are can be found (unofficially) from osrsbox (which references DPS calculator by Bitterkoekje, the forum post of which has been archived), MachOSRS (reddit), and [deleted] (reddit). Following this, we will take care to accurately determine the number of hits required to kill an opponent, followed by experience rate calculations. Related calculations have been performed by Nukelawe, which discusses the impact of overkill on expected damage, which is often neglected in damage calculations. However we will justify that the application of this formula is not properly accounted for and we will instead provide an more accurate method.

1.1 Maximum Hit

There are three combat styles in the game: melee, ranged, and magic. The former two are quite similar, while the latter is slightly more involved. In general, there are 5 bonuses that a player can obtain to boost their maximum hit. These are bonuses from: potions, prayer, other, style, and special attack. These will be referred to as: B^{pot}, B^{pray}, B^{other}, B^{style}, B^{SA}, respectively. For non-magic styles, an effective damage level is defined as

$$L^{\text{eff}} \equiv \left\lfloor \left(L + B^{\text{pot}}\right) B^{\text{pray}} B^{\text{other}} \right\rfloor + B^{\text{style}}, \tag{1.1}$$

where L is either the ranged level or strength level. The base damage, D^{base} is then given by,

$$D^{\text{base}} = C_0 + C_1 S^{\text{eff}} + C_2 E^{\text{str}} + C_3 E^{\text{str}} S^{\text{eff}}, \tag{1.2}$$

$$\text{where } \{C\} = \left\{1.3, \frac{1}{10}, \frac{1}{80}, \frac{1}{640}\right\}, \tag{1.3}$$

and E^{str} is the ranged or melee strength bonus of the worn equipment. The max hit, M is then given by,

$$\boxed{M = \left\lfloor D^{\text{base}} B^{\text{SA}} \right\rfloor}, \tag{1.4}$$

where $B^{\text{SA}} = 1$ if no special attack is being used.

For melee, the `Keras` and `Saradomin Sword` are exceptions to this (see the wiki). For ranged, it should be noted that `bolt` effects only have probabilities of activating (and

therefore increasing M). For future considerations, this could be handled by defining an effective or average max hit: $\langle M \rangle = PM$, where P is the probability of activation. Additionally, some sources state that some equipment which would be categorized under "other" bonuses such as the `Slayer helmet` actually use $\lfloor D^{\text{base}} B^{\text{SA}} B^{\text{other}} \rfloor$ and so act as a multiplier instead of its expected placement. This is not stated on the wiki, but appears to be correct.

Magic

The base damage for a magic spell is fixed and only multiplicative bonuses are obtainable. A single bonus yields a max hit of,

$$M = D^{\text{base}} E^{\text{str}}, \tag{1.5}$$

where now E^{str} is the multiplicative magic damage bonus (think "magic strength"). In general, these effects stack, but intermediate flooring calculations can impact the max hit (check the wiki for ordering). `Charge`, `Magic dart`, `Salamanders`, and `Trident of the seas/swamp` have alternate calculations.

1.2 Accuracy

A will denote the player's probability of a successful hit (called accuracy). This attribute depends on both the player and the opponents stats. For all three combat styles, the attacker rolls an attack role, while the defender rolls a defensive roll. First effective levels are calculated,

$$L_A^{\text{eff}} \equiv \lfloor \left(L + B^{\text{pot}}\right) B^{\text{pray}} B^{\text{other}} \rfloor + B^{\text{style}} + 8, \tag{1.6}$$

where the subscript on L_A is used to denote the difference between the effective damage levels discussed above, and these effective accuracy levels. Then the maximum roll is given by,

$$R_{\text{max}} = \left\lfloor L_A^{\text{eff}} (E + 64) \right\rfloor, \tag{1.7}$$

where E is the equipment bonus for the respective attack or defense style. Now the accuracy is given according to,

$$\boxed{a = \begin{cases} 1 - \frac{1}{2} \frac{D_{\text{max}}+2}{A_{\text{max}}+1} & A_{\text{max}} \geq D_{\text{max}} \\ \frac{A_{\text{max}}}{2D_{\text{max}}+2} & \text{else,} \end{cases}} \tag{1.8}$$

where A_{max} and D_{max} represent the maximum attack and defense rolls (R_{max}) respectively. This is plotted in Fig. 1.1. No information suggests magic accuracy is any different. No other work / official statements show how special attacks factor into this. It is a fair assumption that $a^{\text{eff}} = aB^{\text{SA}}$ for special attacks that increase accuracy (since no flooring needs to be done for accuracy).

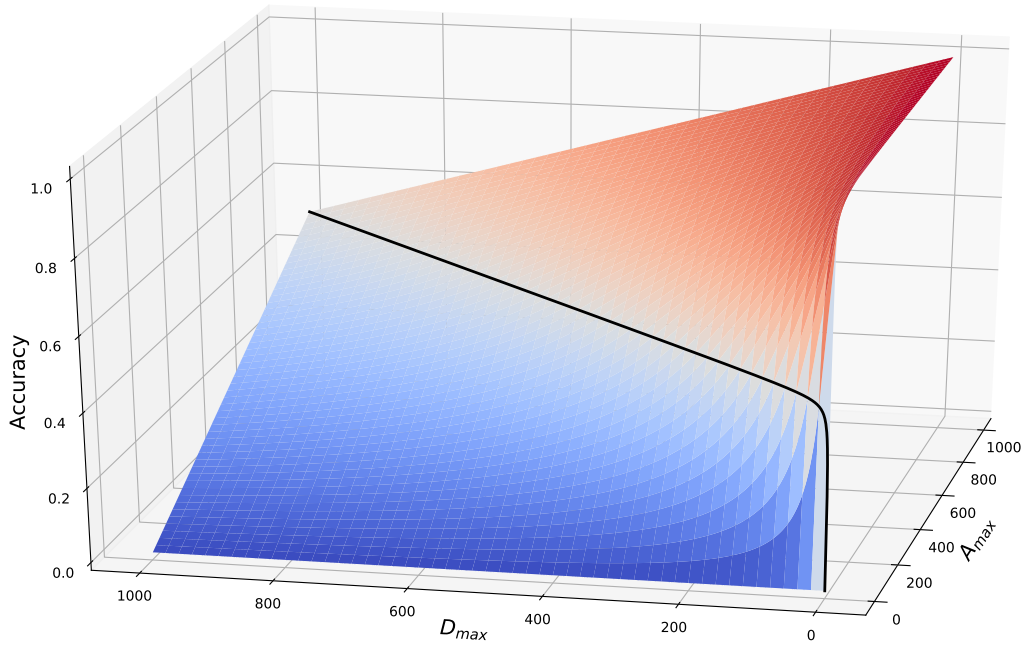


Figure 1.1: Player accuracy as a function of the player's maximum accuracy roll and opponent's maximum defense roll. The black line is the piecewise boundary.

1.3 Experience and Damage Per Second

In the next chapter, we will look at modeling the number of turns, n to kill an opponent. The time to kill is simply $T_A n/a$ where T_A is the time between attacks. Experience is calculated as a multiple, e of the damage done. So the experience per second is given by,

$$\boxed{E = \frac{eah_0}{T_A n}} \tag{1.9}$$

noting that n is also a function of h_0, so E does *not* increase linearly with h_0. For most opponents, $e = 4$. However there are exceptions.

Chapter 2

Models

There are several models that can used to model combat. They vary in how they handle overkill and they will be presented roughly in order of increasing accuracy. These models operate on the following assumptions:

1. A successful attack is uniformly distributed between $[0, M]$ (noting there are $M+1$ integers in this range).
2. An attack is successful with the probability $a \in [0, 1]$, which is the attacker's accuracy.
3. The defender starts at a health h_0.
4. Attacks occur every T_A seconds.
5. No special attacks, or weapon switches are considered.
6. The first attack occurs at time, $t = 0$ or attack number $n = 0$ depending on context.
7. If health regeneration is considered, it occurs every T_r, and heals one health. The first regeneration will occur as a uniform random variable at $t \sim U[0, T_r]$.

There are two relevant quantities that will be calculated for each model: time to health, and health after time.

There are some basic results we can already state:

1. The number of attack attempts is simply a times the number of successful attacks.
2. The time taken for n attacks is simply nT_A.

Only the last model will discuss health regeneration, and it will take a non-regeneration model as input so it applies generally. Finally, a comparison between the different models will be performed.

2.1 Crude Model

2.1.1 Health after n attacks

This model does not consider overkill and as a result is very straight forward. The average damage per successful attack is,

$$\langle D \rangle = \frac{1}{M+1} \sum_{i=0}^{M} i \tag{2.1}$$

$$= \frac{1}{\cancel{M+1}} \frac{M\cancel{(M+1)}}{2} \tag{2.2}$$

$$= \frac{M}{2}. \tag{2.3}$$

After each attack, the health will lower by this amount, recursively this allows us to state:

$$h_{n+1} = h_n - \langle D \rangle, \;\; h_0 \equiv \text{Initial Health}. \tag{2.4}$$

In this case the health after n successful attacks is:

$$h_{n+1} = h_n - \frac{M}{2} \tag{2.5}$$

$$\boxed{h_n = h_0 - n\frac{M}{2}}. \tag{2.6}$$

2.1.2 Attacks until h health

This equation can be inverted to give the number of turns to a given health,

$$\boxed{n = (h_0 - h_n)\frac{2}{M}}. \tag{2.7}$$

Both of these can be calculated in $\mathcal{O}(1)$ time.

2.2 Average

2.2.1 Average Damage

Nukelawe presents a derivation that gives the average damage per hit over the opponent's life to be. This means a contribution from the overkill region and a contribution from the normal region. This averaging acts as an approximation since adds each hit in the overkill region, however in reality only a couple hits during a fight would occur here. With this, the average damage on a hit is given by:

$$\langle D \rangle_{h_0}^{M} = \frac{y(y+1)}{h_0(M+1)} \left(\frac{1}{2}(M + h_0 + 1) - \frac{1}{3}(2y+1) \right), \tag{2.8}$$

where $y = \min(h_0, M)$. Since this is an average over the length of a fight, it means that we cannot determine the health after a given number of terms for this model.

2.2.2 Attacks to kill

Since we know the average damage over the whole fight, we can calculate the number of attacks to kill simply by,

$$n = \frac{h_0}{\langle D \rangle} \tag{2.9}$$

$$= \frac{h_0^2(M+1)}{y(y+1)} \frac{6}{(3M + 3h_0 + 3 - 4y - 2))} \tag{2.10}$$

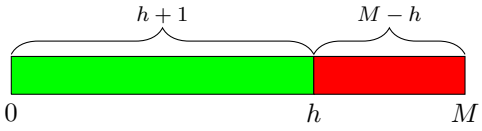
$$\boxed{n = \frac{6h_0^2(M+1)}{y(y+1)\,(3M + 3h_0 - 4y + 1))}} \tag{2.11}$$

This can be calculated in $\mathcal{O}(1)$ time.

2.3 Recursive Model

2.3.1 Expected Damage per Hit

The Crude model made the assumption that the player can always hit up to their max hit. This is not the case if the opponent has less health, h than the maximum hit.



Considering this, we could hit every integer below h each with a probability of $1/(M+1)$, and we could hit exactly h (since we're considering hits capped by h) with a probability of $(M-h)/(M+1)$. Averaging the expectations gives,

$$\langle D \rangle_{h<M} = \frac{1}{M+1} \sum_{i=0}^{h} i + \frac{M-h}{M+1} h \tag{2.12}$$

$$= \frac{1}{M+1} \frac{h(h+1)}{2} + \frac{M-h}{M+1} h \tag{2.13}$$

$$= \frac{h}{2} \frac{2M - h + 1}{M+1} \tag{2.14}$$

$$= \frac{h}{2} \left(1 + \frac{M-h}{M+1} \right) \tag{2.15}$$

$$= \frac{h}{2} \left(2 - \frac{h+1}{M+1} \right) \tag{2.16}$$

Overall then, our expected damage on a successful hit is:

$$\boxed{\langle D \rangle = \frac{1}{2} \begin{cases} M & \text{if } h \geq M \\ h \left(2 - \frac{h+1}{M+1} \right) & \text{if } h < M \end{cases}} \tag{2.17}$$

This is plotted in Fig. 2.1. The Nukelawe model averages over these contributions for the *overall* (i.e. not piecewise) expected hit, assuming each starting health is equally

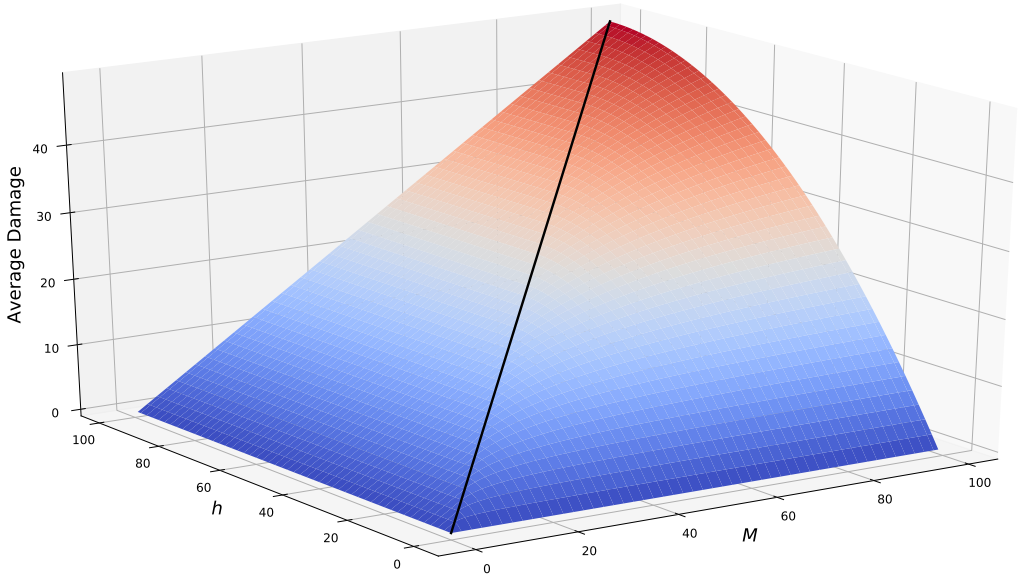


Figure 2.1: The average damage of an attack with a maximum hit of M on an opponent with h health. The black line is the piecewise boundary.

likely. This isn't totally accurate, regardless, it is good to check that our equations agree under this assumption:

$$\langle D \rangle_{\text{overall}} = \langle \langle D \rangle_{h<M} + \langle D \rangle_{h \geq M} \rangle \tag{2.18}$$

$$= \frac{1}{h_0} \left(\sum_{h<M} \langle D \rangle_{h<M} + \sum_{h \geq M} \langle D \rangle_{h \geq M} \rangle \right) \tag{2.19}$$

$$= \frac{1}{h_0} \left(\sum_{n=M+1}^{h_0} \frac{M}{2} + \sum_{n=1}^{y} \frac{n}{2} \left(2 - \frac{n+1}{M+1} \right) \right) \tag{2.20}$$

$$= \frac{y(y+1)}{h_0(M+1)} \left(\frac{1}{2}(M + h_0 + 1) - \frac{1}{3}(2y+1) \right), \tag{2.21}$$

where $y = \min(M, h_0)$. The last step is actually quite tricky and involved to show. For that reason it can be found in Appendix A. This agrees with Nukelawe's findings.

2.3.2 Health after n attacks

Despite the above being an "average", it is not the average damage expected per hit over the life time of an opponent since certain healths are much more likely to appear than others. As an example, suppose there is an opponent with 100 health fighting an attacker with a max hit of 30. On the first hit, the most likely health is 85. This means that unlike the above calculation, each hit is *not* equally likely. Particularly, this over estimates the contribution in the overkill region, since (depending on the circumstances) let's say 1 or 2 hits is expected per kill. Those 1 or 2 hits are more likely to occur at specific h values, but the average sums across all h equally. In the non-overkill region, this has no impact due to it being constant. The proper treatment is to use the piecewise

damage expectation in the recursive definition:

$$h_{n+1} = h_n - \frac{1}{2}\begin{cases} M & \text{if } h_n \geq M \\ h_n\left(2 - \frac{h_n+1}{M+1}\right) & \text{if } h_n < M. \end{cases} \tag{2.22}$$

This however is too cumbersome to handle at once. Since the health is a decreasing monotonic function, we can consider it in two parts. While h is above or equal to M the solution to the above is the same as the Crude model's,

$$h_n = h_0 - n\frac{M}{2}. \tag{2.23}$$

The solution to the other is a bit more complex. After a certain number of iterations the health will drop below M and the second case above will kick in. We'll say this occurs after L iterations (noting that this *average* quantity can be a non-integer),

$$M > h_0 - L\frac{M}{2} \tag{2.24}$$

$$\frac{2}{M}(h_0 - M) < L \tag{2.25}$$

$$\implies L = 2\left(\frac{h_0}{M} - 1\right). \tag{2.26}$$

Now the expected health that the second condition starts at is given by,

$$\langle h_L \rangle = h_0 - 2\left(\frac{h_0}{M} - 1\right)\frac{M}{2} \tag{2.27}$$

$$= M. \tag{2.28}$$

Thus the second case is expected to starts on iteration L with an initial health of M. For simplicity, we will define $m \equiv n - L$. Returning to our recursive function,

$$h_{m+1} = h_m - \frac{h_m}{2}\left(2 - \frac{h_m + 1}{M+1}\right) \tag{2.29}$$

$$= h_m - h_m + \frac{h_m}{2}\frac{h_m + 1}{M+1} \tag{2.30}$$

$$= \frac{h_m^2 + h_m}{2(M+1)} \tag{2.31}$$

$$h_{m+1} = \gamma(h_m^2 + h_m), \tag{2.32}$$

where $\gamma \equiv 1/2(M+1)$. This type of recurrence relation is called a Quadratic Map, and unfortunately it has no closed form solution in general. Some work is shown in Appendix B to attempt to simplify it, but no final form was realized. For the sake of completeness, we will call the solution to Eq. 2.32 $f(m; M, f_0)$. At this point we're left with,

$$h(n; h_0, M) = \begin{cases} h_0 - \frac{1}{2}nM & \text{else if } n \leq L \\ f(n - L; M, M) & \text{otherwise.} \end{cases} \tag{2.33}$$

In general, we are more interested in the number of iterations that is required to kill an opponent which is the inverse of this function, $n = h^{-1}(h; h_0, M)$. However, since the recurrence relation cannot be solved analytically, we cannot obtain a general expression for this. We could numerically compute this result. However remember that we are

dealing with *expectation values* or averages. This means it is totally possible for the inverted function to say "The opponents health will be 18 in 4.5 iterations, on average". This is a problem because our recursive equation can only increment the iteration by one (and we can only start at f_0)! In the next section, we will look at an approximation that will allow us to handle non-integer expectations.

Approximate Solution

Despite not being able to find a closed-form solution for Eq. 2.32, we can look for approximate solutions. Although it's rare, there are a few quadratic recurrence relations that **do** have solutions. There are two relevant cases, ones where a constant term is introduced, or ones where the linear term is removed. Since increasing n (and therefore iterations) results in smaller and smaller h_n, it suggests that adding constant terms will heavily skew the results in this region. Removing the linear term is still not ideal since its contribution increases relative to the quadratic term, in the low h_n limit (when this recursive case is required). However this should be less significant and we'll find that it does pretty well! If this justification does not satisfy you, please check out Appendix C. With this approximation, our recursive relation becomes,

$$h_{m+1} = \gamma h_m^2. \tag{2.34}$$

Starting with $h_0 = M$, and looking at a few terms reveals,

$$\begin{aligned}
h_1 &= \gamma^1 M^2 && (2.35)\\
h_2 &= \gamma^1 h_1^2 && (2.36)\\
&= \gamma^1\gamma^2 M^{(2\cdot 2)} && (2.37)\\
h_3 &= \gamma^1 h_2^2 && (2.38)\\
&= \gamma^1\gamma^2\gamma^4 M^{(2\cdot 2\cdot 2)} && (2.39)\\
\implies h_m &= \gamma^{\sum_{i=0}^{m-1} 2^i} M^{2^m} && (2.40)\\
&= \gamma^{2^m-1} M^{2^m} && (2.41)\\
&= \frac{1}{\gamma}(\gamma M)^{2^m} && (2.42)\\
&\boxed{h_m = \frac{1}{\gamma}\left(\frac{1}{2} - \gamma\right)^{2^m}} && (2.43)
\end{aligned}$$

This equation is not only a closed form expression, but it can also be inverted!

$$\log_2(\log_{\frac{1}{2}-\gamma}(\gamma h_m)) = m. \tag{2.44}$$

This equation asymptotically reaches 0, but the opponent dying occurs when h_m drops below 1 yielding an average kill on iteration,

$$\log_2(\log_{\frac{1}{2}-\gamma}(\gamma)) = m. \tag{2.45}$$

How can we make use of this approximation to improve our more accurate calculation? Remember that we are trying to solve the issue of non-integer iterations. Instead of beginning on iteration 0 for the iterative procedure, we can start m at any real number in (0, 1), and iterate upwards on that. For example to get the health at iteration 4.87, we could use our analytic approximation to calculate 0.87 (which uses *one* approximation), then iterate 4 times using the correct equation to get to the final result. So

this defines our new best approximation,

$$\boxed{\begin{aligned} h(n;h_0,M) &= \begin{cases} h_0 - \frac{1}{2}nM & \text{if } n \leq L \\ \frac{1}{\gamma}\left(\frac{1}{2}-\gamma\right)^{2^{n-L}} & \text{if } n-L<1 \\ f\left(n-L;M,\frac{1}{\gamma}\left(\frac{1}{2}-\gamma\right)^{2^{n-L}}\right) & \text{otherwise} \end{cases} \\ \text{where, } \gamma &= \frac{1}{2(M+1)} \text{ and } L = 2\left(\frac{h_0}{M}-1\right). \end{aligned}} \tag{2.46}$$

2.3.3 Attacks until h health

To solve for when the opponents health drops below one, $n = h^{-1}(1)$, we must use a numerical root finding algorithm to solve for the zero of $(1 - h(n))$. It is possible that $h^{-1}(1/2)$ is more representative of death than $h^{-1}(1)$. However, empirically, using 1 is in better agreement with simulation.

The time complexity can be approximated using Eq. 2.44:

$$\begin{aligned} \mathcal{O} &= \log_2(\log_{\frac{1}{2}-\gamma}(\gamma h_m)) && (2.47)\\ &\propto \log\left(\frac{\log(\gamma h_m)}{\log(\frac{1}{2}-\gamma)}\right) && (2.48)\\ &\approx \log\left(\frac{\log(\frac{1}{\gamma h_m})}{\log(2)}\right) && (2.49)\\ &= \log\log\left(\frac{1}{\gamma h_m}\right) - \log\log(2) && (2.50)\\ &\sim \log\log\left(\frac{1}{\gamma h_m}\right) && (2.51)\\ &\approx \log\log\left(\frac{2m}{h_m}\right). && (2.52) \end{aligned}$$

Here, $\propto$ is used to signal a multiplicative factor was ignored, and $\sim$ is used to signal an additive factor was ignored. In the first line the change of base formula was used, and the factor $log(2)$ was ignored. In the second line, $1/2 - \gamma \approx 1/2$ was used. This leaves us with $\mathcal{O}(\log\log(2m/h))$ evaluations required, which is practically constant. However, the number of evaluations for the numerical inversion depends on the implementation. In practice, at most 30 evaluations are required.

2.4 Approximately Analytic Recursion

2.4.1 Health after n attacks

We can simplify the Recursive model by extending the approximation to all $n > L$:

$$\boxed{h(n;h_0,M) = \begin{cases} h_0 - \frac{1}{2}nM & \text{if } n \leq L \\ \frac{1}{\gamma}\left(\frac{1}{2}-\gamma\right)^{2^{n-L}} & \text{otherwise.} \end{cases}} \tag{2.53}$$

2.4.2 Attacks until h health

This is useful because it can be inverted analytically:

$$\boxed{n(h;h_0,M)=\begin{cases}(h_0-h)\frac{2}{M} & \text{if } h\geq M\\ \log_2(\log_{\frac{1}{2}-\gamma}(\gamma h)) & \text{otherwise}\end{cases}}. \tag{2.54}$$

These can be evaluated in $\mathcal{O}(1)$ time.

2.5 Health Regeneration

This is a method that considers health regeneration as piecewise averages. We start with one of our models which can give the health of the opponent after n total attacks, $h(n;a,M,h_0)$ (without regeneration). Based on the weapon attack interval, T_A, we can convert to seconds through $t=nT_A$. This leaves us with $h(t/T_A;a,M,h_0)$. These equations cannot solve health regeneration since the health no longer decreases monotonically, and as a result the cases cannot be separated. Let's roll with this restriction and solve the problem within a window where the opponent does not heal. First we will make the assumption that the first regeneration occurs at $t=T_R$, which is the time between regenerations. We will later remove this restriction by averaging the first regeneration over $t\in[1,T_R]$. To reduce clutter, let's remove some explicit parameters and redefine h as $h(t,h_0)$. The health just before the opponent heals is

$$h_1=h(T_R,h_0), \tag{2.55}$$

which is the health after T_R seconds starting from some initial health h_0. Immediately after, the opponent will heal by $+1$ health,

$$h_1=h(T_R,h_0)+1. \tag{2.56}$$

Iterating again:

$$\begin{aligned}h_2&=h(T_R,h_1)+1 &(2.57)\\ &=h(T_R,h(T_R,h_0)+1)+1. &(2.58)\end{aligned}$$

Letting, $f(x)=h(T_R,x)$ to, again, reduce clutter and iterating again yields,

$$\begin{aligned}h_3&=f(h_2)+1 &(2.59)\\ &=f(f(h_1)+1)+1 &(2.60)\\ &=f(f(f(h_0)+1)+1)+1. &(2.61)\end{aligned}$$

In general this gives us an *iterated function*,

$$h_m=g^{\circ m}(h_0), \tag{2.62}$$

where m is the number of regeneration periods, and $g(x)\equiv f(x)+1$. As an example of the function composition, for $m=2$ the above expands to:

$$\begin{aligned}g^{\circ 2}(x)&=(g\circ g)(x)=g(g(x)) &(2.63)\\ &=f(f(x)+1)+1. &(2.64)\end{aligned}$$

This is great but this process doesn't continue indefinitely since the opponent will eventually die. There is a health, h^* after which, the opponent will not be able to heal.

Starting at a health of zero, we can determine how much health the player had T_R seconds ago,

$$h^* = h(-T_R, h_0 = 0) \tag{2.65}$$

Let's summarize. The opponent starts at h_0 health. They will not heal after h^*. The time taken to die is equal to the number of healing cycles until they drop below h^*. Plus the time it takes to get from that remaining health (not necessarily h^*) to 0. Pseudocode for this is given below, where `health_after` and `time_to_kill` are both non-regen functions. Parameters like accuracy and max hit are omitted.

```
def time_to_kill_regen(h_0):
  h = h_0
  t = 0
  h* = health_after(-T_R, 0)
  while h ≥ h*:
    h = health_after(T_R, h) + 1
    t += T_R
  return t + time_to_kill(h)

```

Now consider the initial condition. We assumed that the first regeneration occurred at T_R. However, it is a discrete (due to game ticks) uniform random variable distributed between $t_r \sim U[1, T_R]$. If a tick occurs every τ seconds, then there are T_R/τ game ticks before a heal.

$$h_1 = h(t_r, h_0) \tag{2.66}$$
$$h_2 = h(T_R, h(t_r, h_0) + 1) \tag{2.67}$$
$$h_3 = h(T_R, h(T_R, h(t_r, h_0) + 1) + 1) \tag{2.68}$$
$$h_n = (h \circ (x+1))^{\circ n-1}(T_R, h(t_r, h_0)) \tag{2.69}$$
$$\langle h_n \rangle = \frac{\tau}{T_R} \sum_{i=1}^{T_R/\tau} (h \circ (x+1))^{\circ n-1}(T_R, h(i\tau, h_0)) \tag{2.70}$$

Now this has to be numerically inverted to give the time to kill.

2.6 Comparisons

We will only be comparing "turns to *kill*" since "turns to a given health" is only possible with some models. In addition, since the health regeneration case adds additional complexities for expectedly minor improvements, it also won't be considered here.

The combat is actually simple to simulate, however because it is a stochastic process that converges slowly getting high precision results is difficult this way. Regardless it can be done over the course of hours or days depending on the desired precision level. Our models can then be compared to these numerically simulated fights as is done in Fig. (2.3). Regardless of the model, the turns to kill grows quickly when the max hit is low compared to the initial health of the opponent as shown in Fig. 2.2. By visual inspection, the time appears to scale roughly linearly with the health of the opponent, and exponentially with decreasing max hits.

Looking closer at the graphs, the deviations between different models becomes more apparent. We can investigate this further by comparing the *error* between the models' predictions and the simulation. Figure 2.3 shows a wide variation in performance across these models. Most notably, the crude model performs the worst overall. Both the

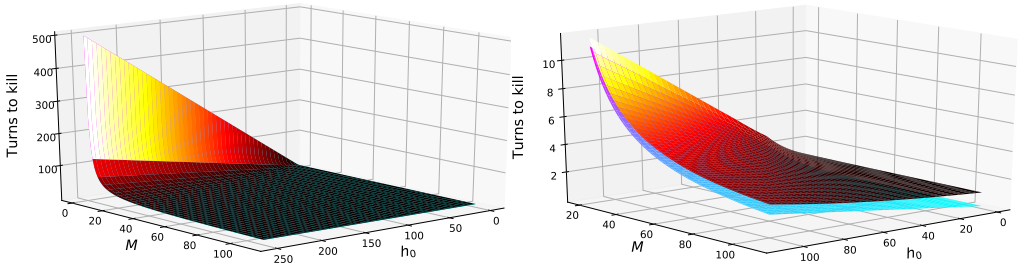


Figure 2.2: The number of successful turns to kill an opponent that has h_0 health, given that the attacker's max hit is M. The MarkovChain (hot colors) and Crude (cool colors) methods are shown. However, due to the scale of the left plot they are indistinguishable. The right plot shows a smaller region which allows for a comparison.

average and recursive methods have high error peaks, but otherwise do quite well. The Markov Chain model performs essentially perfectly, with it's approximation not falling far behind. Some statistics about the models are shown in Table 2.1. Notably, the Markov Chain model has a significantly lower average error, standard deviation, and maximum error than all other models.

Model	Average	Standard Deviation	Maximum	Minimum
Crude	18.14	18.54	98.20	1.75e-04
Average	10.29	17.92	73.00	8.85e-06
MarkovChain	0.06	0.06	0.62	7.35e-07
MarkovChainApprox	1.56	4.82	32.73	7.35e-07

Table 2.1: Error statistics for various models.

Finally, one more interesting comparison to make is seeing which models perform best in which regions. Figure 2.4 shows a heatmap depicting which models perform the best in which regions. There are several interesting trends that emerge (although their explanations are unknown but likely arbitrary). The comparison between all the models are done in three stages: first the crude, average, and recursive model are compared, then the Markov Chain Approximation is added, and finally all 5 models are compared. This is done since the Markov Chain models dominate, which would prevent us some seeing how the simpler models relate. The Crude model only makes an appearance when $M = 1$ since overkill plays no role here and all models agree. The arbitrary ordering of evaluations places the Crude model first (but all models are equally valid here). In the first plot, we see that Average dominates the majority of the domain. However this is slightly mis-leading as the improvement over the Recursive model here is small, whereas in the region where the Recursive model dominates the improvement is substantial. Perhaps a better visualization would interpolate the colors depending on relative improvement. Regardless, in the next plot the Markov Chain Approximation quickly dominates essentially the whole domain. Finally, the last plot shows a battle between both Markov Chain models. However, there are a few details to note here. First there is a "cut-out" region at low max hits for which the Markov Chain model could not be evaluated due to numerical overflow in its current implementation. Second, it is strange that the approximation performs similarly well throughout and shares roughly equal domain (although this may be due to simply being such similar estimates). What makes this even more strange though is that a pattern emerges, in which the approximation seems to dominate at *angles* in the domain i.e. lines with slopes that are multiples of h_0/M. There is no current explanation for this.

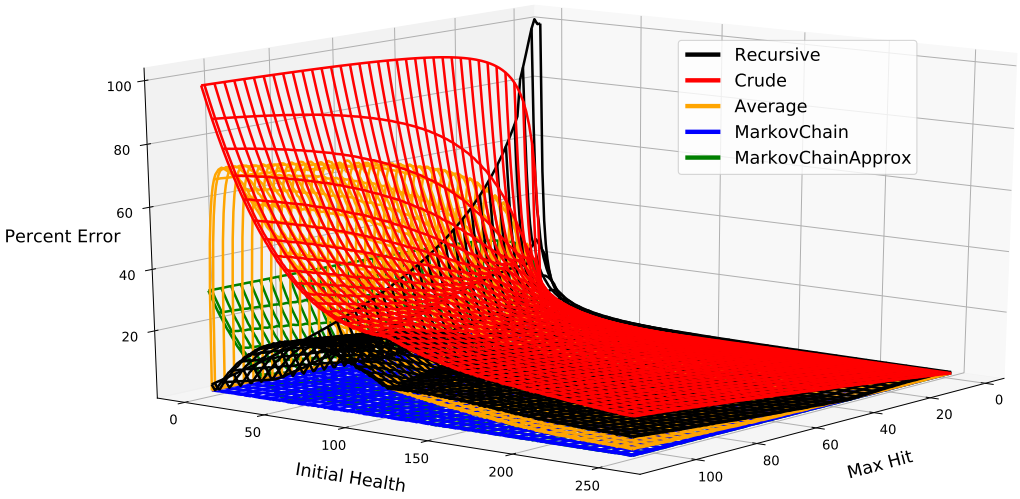


Figure 2.3: A comparison between different models. The percent relative error with respect to the simulations are shown.

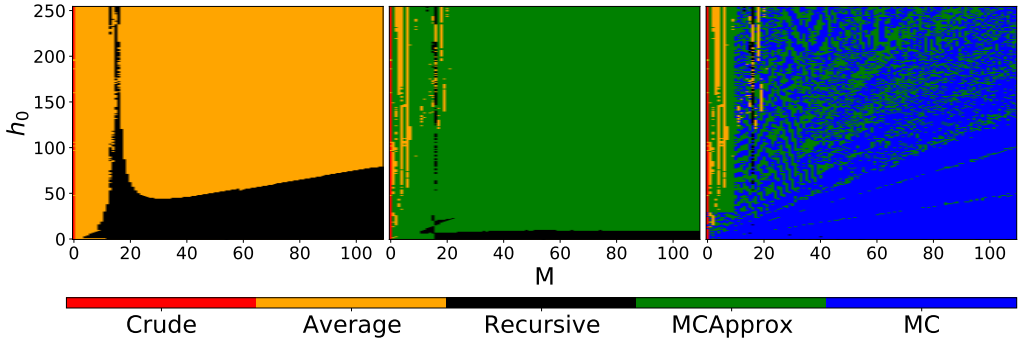


Figure 2.4: The model which agrees best with simulation, for different initial healths, and maximum hits. The final plot shows a comparison which includes all models. Since the MarkovChain (MC) models have very low errors, they occlude the other details. So, the other plots only include a subset of models. The leftmost only shows Crude, Average, and Recursive. The middle plot additionally includes MarkovChainApprox (MCApprox). Several interesting trends are discussed in the main body.

Part II

Applications

Chapter 3

Nightmare Zone

We can now attempt to extend this into an useful application. For practical purposes, the Nightmare Zone (NMZ) is a combat training ground minigame that offers a low intensity, but efficient method of training combat skills. The player enters an arena with several bosses they have previously fought in the main game. At any given time, four (randomly chosen - with replacement) are in combat with the player. Before beginning, the player can decide a minimum of five bosses they would like to re-fight.

This can be seen as a first a check, and second an optimization problem (to maximize experience). At this point we have calculated the experience per hour in terms of: accuracy, max hit, initial opponent health, and attack speed. The accuracy and max hit are involved yet straightforward calculations, while the other parameters are generally in a database. We have only made two assumptions so far, no health regeneration, and constant combat, along with 1 approximation in the turns to kill formula. 1 health is regenerated every minute the opponent is under their maximum health. [Pretty sure I've seen exceptions] Thus, kills under on minute shouldn't be affected by this, while kills on the order of a few minutes only have a very small influence (since the duration of the fight typically implies high health). The lower the initial health of the opponent, the lower the impact of health regeneration.

3.1 Experimental Considerations

The health regeneration has a small effect of increasing damage per second by extending the non-overkill region and increasing the damage cap in the overkill region). Calculations are performed assuming continuous combat but due to random spawning, they may line up in a way that delays combat. Additionally, larger opponents would increase the probability of delay. Maintaining the player's own health requires some sort of consumption which may introduce delays. Finally, the bosses may spawn in unequal proportions (on average this should wash away, but may either increase or decrease the experience rates). These effects are summarized in Table. 3.1.

3.1.1 Validation

To increase the probability that the player will remain in combat, only opponents who occupy a single tile will be used (that way more can stack up on the player, and they are more likely to retaliate against someone). For the sake of accuracy and reducing human error, no prayers, or special attack will be used. Additionally, tank gear, and absorption potions will be used to minimize the influence of sustaining the player's health. Runelite's built-in experience rate tracker will be used to collect data.

Figure 3.1: A player (bottom-right) fighting against four bosses in the NMZ. Three of the bosses at this time happened to be the same, however each boss is expected to occur with equal probability.

Property	Regeneration	Intermittent Combat	Sustainability	Equal Counts
Impact	Higher	Lower	Lower	Random
Multiplicative	No	Yes	Yes	Yes

Method	Overkill Consideration	Experience Prediction
Crude	Over Estimate	Loose Lower Bound
Recursive	Approximate	Upper Bound
Simulation	Approximate	Upper Bound
Bitter	Over Estimate	Loose Lower Bound

Table 3.1: The top table shows the effect on experience rates when certain factors are properly considered. Multiplicative factors means that that the ratio between two different rates are independent of these quantities, non-multiplicative factors appear inside the particular method which in general would not cancel. The bottom table relates how overkill is considered to the over/under prediction of experience rates. The properties with the largest influence decrease the experience rate. So the methods which well-approximate overkill are upper bounds. Those that over estimate it are lose lower bounds since otherwise they would be upper bounds so this additional decrease may or may not put it below the true expectation.

Figure 3.2: The main setup used for validation. Due to the nature of testing, some health, strength, and combat levels were gained, but were properly taken into account.

Time-Dependent Boosts

There are several time-dependent boosts in the game, most notably potions. Additionally, Dharok is a set of equipment which provides increased damage based on the health missing from the player. Since health recovers at a rate of one health per minute, and the boosted level from potions decreases one level per minute, these introduce time-dependent experience rates. Thankfully, the treatment is relatively simple. Let $a = a(t)$, and $M = M(t)$. At each boosted state, we can calculate the expected experience rate, and average over the boosted states:

$$\boxed{\langle E \rangle = \frac{1}{N} \sum_{i=0}^{N} E(a(t_i), M(t_i))} \tag{3.1}$$

For example, if the player re-drinks a potion after $N = 10$ levels have dropped, then the above determines the average experience. Things are a little bit more complex when both potions and Dharok are used since the timings might not line up, but for large times the detail should get averaged out.

Comparisons

Settings	Crude	Recursive	Simulation	Bitt-Nuke	GamePlay
Easy No Potions $M \in [26]$ 8.36h	66.9 +15.19%	59.1 +1.83%	58.5 +0.78%	56.7 −2.42%	58.1±0.6 [1.0%]
Hard No Potions $M \in [27]$ 8.19h	68.7 +6.52%	65.6 +1.71%	65.5 +1.49%	65.0 +0.75%	64.5±0.4 [0.7%]

Easy Potions $M \in [27..32]$ 6.84h	79.8 +18.16%	68.6 +1.62%	68.3 +1.21%	65.7 −2.69%	67.5±1.0 [1.5%]
Hard Potions $M \in [27..32]$ 6.81h	79.6 +6.25%	75.3 +0.55%	75.2 +0.45%	74.7 −0.23%	74.9±0.9 [1.2%]

Table 3.2: Experience per hour predictions from different methods. Potions and Easy or Hard bosses which correspond to the boss difficulty (which only increases health and offensive capability) are varied. Each method reports two values, the xp/h [in kilo-xp] and the signed percent difference from the game play. Green coloring indicates there is agreement within error, red is used otherwise. The bracketed quantities are the relative error of the reported rate. Max hits are given as a range when potions are used. Data collection periods are also given in hours.

Since the largest (still small) source of errors in all models (constant combat, and health sustaining (absorptions potions)) negatively impact experience rates, we expect the models to be tight upper bounds on the actual gameplay experience. However, it's possible that the bosses don't spawn in equal amounts, which may bias the results in either direction (Runelite doesn't properly record kill counts to check this). For no potions, only the max hit matters, but for potions, the strength level also determines what the boost range. To prevent needing to account for this, when potions are used, runs are only carried out for a single strength level. For no potions, as long as the max hit is unchanged any strength level can be used.

Bitt-Nuke over-considers low end which lowers estimate, but it can't be known if this lowering beats the unaccounted for non-constant combat. Which may explain why it was occasionally under-estimated results, while the Recursive method was always expected to be a upper-bound. This means that with additional corrections, the Recursive model will perform better than Bitt-Nuke, although as reported this seems effect isn't strong.

Chapter 4

Optimal Equipment

Given the ability to model combat effectively, a natural question arises: What is a best equipment for a player to use against an opponent. Our objective to maximize is the experience rate, given most completely by Eq. (3.1). There are a lot of parameters underneath this equation: the defensive characteristics of the opponent pool (levels, bonuses), the offensive characteristics of the player, and the player's boosts (prayer, potions). However the majority of this has already been modeled and discussed in the previous sections. The experience rate can be calculated for any fixed set of equipment, so how can we use this to find the most efficient setup.

There are 11 equipment slots, and each slot has on the order of 10-100 items (there is a large variance). A brute force search, would require between 10^{11} and $100^9 = 10^{22}$ sets to be considered. Thus our problem boils down to a search space reduction problem. Intuitively, we know this problem can be greatly simplified. For a player with maxed levels, there is no reason to compare a `Ghrazi rapier` to a `Bronze dagger`. But what exactly allows to say that?

Let's assume the player would like to train attack. Each melee weapon in the game has several `attack types` which can be at least one of `slash, stab, crush`. Equipment affects each type differently so it makes sense to compartmentalize this problem into fixing both the training skill, and the attack type. This would allow us to iterate over the valid `attack types` for each weapon and pick whichever gives the highest experience rate. At this point the question becomes "For a given player training *attack* using a *Ghrazi rapier* with *stab*, what is the best equipment for this?".

With this simplification, the number of parameters that we have to consider for each piece of equipment is significantly reduced. However only the `attack type` bonus, the `strength bonus`, and if we are comparing weapons the `attack speed` would matter. Then, for each slot we can take the subset of the equipment that is strictly worse than everything else i.e. we cannot know that (stab = 5, strength = 5) is better than (stab = 10, strength = 1) (since the way accuracy and max hit influence xp is complex and relies on a given model), but we can know that (stab = 1, strength = 1) is worse than both and so we shouldn't consider it.

This typically reduces the subsets to between 1 and 5 items per slot. We also have to iterate over the `attack types` so for melee there are typically between $3 \cdot 1^{11} = 3$ to $3 \cdot 5^{11} \approx 3 \cdot 10^6$ possible optimal equipment sets. Many slots work out to only having a single item, and the typical number of sets is about 100-250. This means that everything single other combination is worse than something in the sets, which is a really powerful statement. Especially considering that this doesn't rely on any model!

One final consider is the influence of set effects, most equipment with set effects are not optimal without their effects and are therefore ruled out of the above process.

However, running the same algorithm but fixing the slots associated with the equipment set resolves this. The amount of sets to consider increases relatively quickly (and approximately linearly) with the number of equipment sets considered. In practice, 6 additional sets typically introduces 400 additional sets to consider.

Ultimately what we end up with is a computational efficient method for determining the *mathematically* optimal set of equipment for a player to fight an arbitrary group of opponents.

Chapter 5

Optimal Training Order

We will try to find the most efficient ordering to level up melee combat skills to maximize efficiency. For simplicity, we will fix equipment, and NMZ opponents. Let's start at $(\text{attack}, \text{strength}) = (a, s)$ and end at (A, S). Defense is not considered since it does not increase experience rates (apart from less interruption required from sustaining health). The first question is should we obtain $(a+1, s)$ or $(a, s+1)$. This question can be asked recursively, producing a tree-like structure of possible training schemes as depicted in Fig. 5.1. Each downward path is a possible way to get to the target levels.

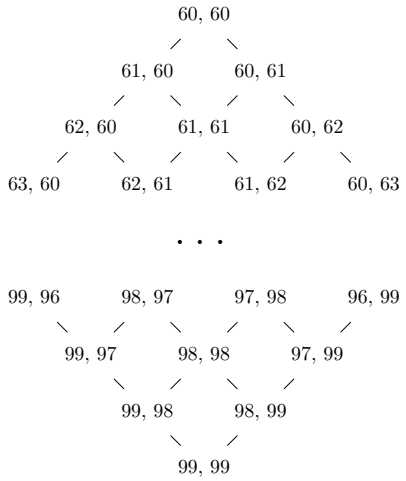


Figure 5.1: The possible ways to train from 60 attack and strength to 99 attack and strength. Branching to the left represents training an attack level, and to the right is strength. At a point, 99 is reached along the edge and the tree cannot grow out anymore and eventually collapses to the desired levels.

The optimization task then becomes how do we traverse from the top of this tree-like structure to the bottom most efficiently? We can assign a *cost* to each edge in this graph corresponding to the time it takes to level up. The experience required to level up to a given level, x is:

$$\xi(x) = \frac{\left\lfloor x - 1 + 300 \times 2^{\frac{(x-1)}{7}} \right\rfloor}{4} \tag{5.1}$$

The time to level attack or strength is given respectively by,

$$t^{\text{attack}} = \xi(a)/\langle E(a, s; \text{attack})\rangle \tag{5.2}$$

$$t^{\text{strength}} = \xi(s)/\langle E(a, s; \text{strength})\rangle, \tag{5.3}$$

where we explicitly show that the experience per hour, E depends on the attack style (due to the combat bonus). The above two equations give the cost for branching to the left, and right, respectively. There is a relatively simple pattern to generate the children of any row. Building the whole tree has a $\mathcal{O}((A-a)(S-s))$ time-complexity, making this a feasible operation. The sum of costs (times) along any path is the time it takes to complete the leveling. Minimizing this corresponds to the family of problems called shortest path problems. Our particular problem is solving the single-source directed graph with non-negative weightings. This can be solved using the Dijkstra Algorithm in quadratic time. A Python library Dijkstra implements this. So this problem is solved in two parts: 1) Construct graph, 2) Apply Dijkstra. A few solutions are shown in Fig. 5.2. There is occasional switching, but ultimately strength is maximized first, followed by attack.

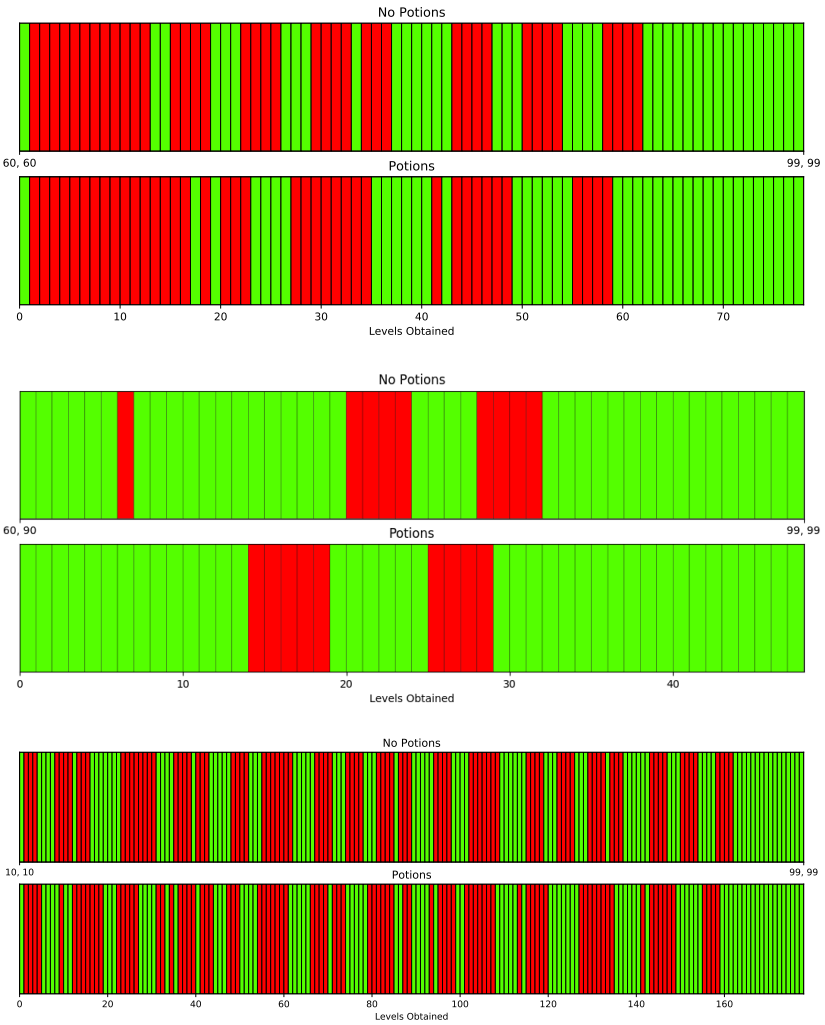


Figure 5.2: The most efficient training is obtained by following these training schemes. Starting on the left, the player should train attack (green) or strength (red) until they reach the right side (99 attack & strength). Three schemes are shown for different starting levels: (60, 60), (60, 90), and (10, 10). The latter is unrealistic since 60 attack is required to wield the dragon scimitar used in these calculations, but this shows the long term behavior. Additionally, for each scheme calculations using or not using potions is shown.

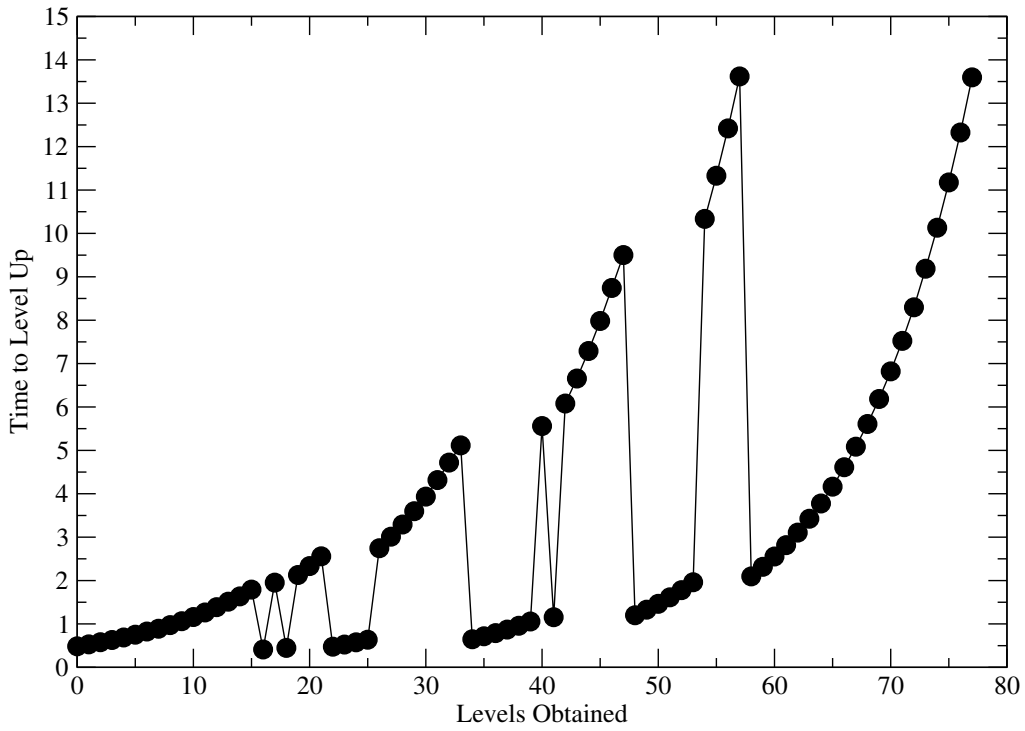


Figure 5.3: Following the optimal training scheme (using potions) starting from (60, 60), the training time for each level is plotted until the player arrives at (99, 99).

Taking one of these schemes, like $(60, 60) \rightarrow (99, 99)$ using potions, we can calculate the actual time required to move to each next "strip" in Fig. 5.2, as done in Fig. 5.3. Strength is often trained first resulting in training times that are higher earlier on. Because of this, training is switched to attack. There are actually two competing effects at play. Attack has a small bonus to dps, but is often a lower level and is therefore easier to obtain. Conversely, strength has a large bonus, but quickly becomes too time-expensive to train before any benefit is received, so briefly attack is trained. After 99 strength is obtained 60 levels, attack is the only thing left to train until max. If the scheme was to alternate levels between attack and strength, the curve would be a single curve, but this staggered scheme results the prominent bifurcation.

Chapter 6

Improvements

Health regeneration. Analytic solution? Constant Combat (acts as reduced weapon speed so multiplicative). Health Sustainability. The last two shouldn't impact rankings since they are multiplicative and independent of the calculation method. Based on this, we can say that exact rankings is more important than exact approximations for practical purposes. Is the treatment $h = 1$ for a kill valid for all methods? Since crude and Bitterkoekje can get to 0? I've seen (enemy) hp go up in under a minute. There must be a global timer, not just every 60s after you drop below max health.

Chapter 7

Conclusion

We summarized the combat mechanics behind effective levels, accuracy, and maximum hits. We performed detailed calculations to determine piecewise average damage, based on whether overkill was present. In contrast to the Bitt-Nuke model, we maintain this piecewise relation and evaluate an recursive scheme to determine a more accurate form for the average damage. An analytic approximation was used to handle non-integer turns to kill an opponent, which can ultimately be used to calculate experience per hour. We discuss the effects which are not considered like health regeneration, and constant combat. Finally, experience per hour models are compared. The Recursive model is the best model due to its accuracy, and since it satisfies the expected upper bound property. Future work may look into forming an optimization problem.

Appendices

Appendix A

Average Damage

The average damage described in Section 2.3 can be expanded to give,

$$\langle D\rangle_{\text{overall}} = \frac{1}{h_0}\left(\sum_{n=M+1}^{h_0}\frac{M}{2} + \sum_{n=1}^{y}\frac{n}{2}\left(2-\frac{n+1}{M+1}\right)\right) \tag{A.1}$$

$$= \frac{1}{h_0}\left(\frac{M}{2}(h_0-y) + \sum_{n=1}^{y} n - \frac{1}{2}\sum_{n=1}^{y} n\frac{n+1}{M+1}\right) \tag{A.2}$$

$$= \frac{1}{2h_0}\left(Mh_0 - My + y(y+1) - \frac{1}{2(M+1)}\sum_{n=1}^{y}(n^2+n)\right) \tag{A.3}$$

$$= \frac{1}{2h_0}\left(Mh_0 - My + y(y+1) - \frac{y(y+1)}{2(M+1)} - \frac{y(n+1)(2y+1)}{6(M+1)}\right) \tag{A.4}$$

$$= \frac{1}{2h_0(M+1)}\left(M^2h_0 - M^2y + My^2 + yM + Mh_0 - My - \frac{y^3-y}{3}\right) \tag{A.5}$$

$$= \frac{y(y+1)}{h_0(M+1)}\left(\frac{M(M+1)h_0}{2y(y+1)} + \frac{(y-M)M}{2(y+1)} - \frac{y-1}{6}\right) \tag{A.6}$$

$$= \frac{y(y+1)}{h_0(M+1)}\left(\frac{M(M+1)h_0}{2y(y+1)} + \frac{(y-M)M}{2(y+1)} + \frac{y+1}{2} - \frac{1}{3}(2y+1)\right) \tag{A.7}$$

where $y = \min(M, h_0)$. In the second line, we used:

$$\sum_{a+1}^{b} 1 = \begin{cases} b-a & \text{if } b > a \\ 0 & \text{else} \end{cases} \tag{A.8}$$

$$= b - \begin{cases} a & \text{if } b > a \\ 0 & \text{else} \end{cases} \tag{A.9}$$

$$= b - \min(a, b) \tag{A.10}$$

To finish, let's focus on,

$$\frac{M(M+1)h_0}{2y(y+1)} + \frac{(y-M)M}{2(y+1)} + \frac{y+1}{2} \tag{A.11}$$

$$= \frac{1}{2y(y+1)}\Big[M(M+1)h_0 + y(y-M)M + y(y+1)^2\Big] \tag{A.12}$$

$$= \frac{1}{2y(y+1)}\Big[M^2h_0 + Mh_0 + My^2 - M^2y + y^3 + 2y^2 + y\Big]. \tag{A.13}$$

This is a hard equation to simplify since the M's and h_0's are implicitly embedded in the y's, but if you play with it long enough you can "discover" a way to simplify it - a form *power reduction* that relies on getting rid of as many y's as possible.

A.1 Power Reduction

I'd like to preface the next part by saying the final result can easily be determined by plugging in m as the min, and h_0 as the min and combining the result. In this instance it works out nicely, but we will focus on general machinery to solve these problems assuming the solution was not so nice. Our goal here is to pull the m's and h_0's out of y. To do this, let's see if there is a way to construct y^2 from the other variables, specifically only using y^1. We know that if $M < h_0$, we need a term like M^2, and in the opposite case, we need a term like h_0^2,

$$y^2 \sim M^2 \text{ or } h_0^2. \tag{A.14}$$

Based on this, we should be able to use min to switch between these two. So if we write the first term using y, we'd have something like My, which is true when M is the minimum. If it isn't the minimum, there should be a second term which cancels the now Mh_0 term plus the required h_0^2 term:

$$y^2 = My + h_0(y - M) \quad (!) \tag{A.15}$$

Using the same logic, we can inductively deduce,

$$\boxed{y^{n+1} = My^n + h_0^n(y - M) = My^n + h_0^n y - Mh_0^n.} \tag{A.16}$$

(and as an identity for the math people, with $\gamma = \min(a, b)$):

$$\boxed{\gamma^{n+1} = a\gamma^n + b^n(\gamma - a) = a\gamma^n + b^n\gamma - ab^n} \tag{A.17}$$

In fact, this holds for max as well, or any *similar* piece-wise function. Writing this as a recursive sequence by letting $g(n) = \gamma^n$ yields,

$$g(n + 1) = ag(n) + b^n(g(1) - a), \tag{A.18}$$

Under the initial condition $g(1) = \gamma$, WolframAlpha gives the general solution as,

$$g(n) = a^n + (\gamma - a)\frac{a^n - b^n}{a - b} \tag{A.19}$$

$$\boxed{g(n) = a^n + (\gamma - a)\sum_{i=0}^{n-1} a^{n-i-1}b^j,} \tag{A.20}$$

where the second line uses the difference of powers formula. This could have been solved by hand, but we've had enough fun with recursion in the other sections! This yields,

$$
\begin{aligned}
g(1) &= \gamma && \text{(A.21)}\\
g(2) &= a^2 + (\gamma - a)(a + b) && \text{(A.22)}\\
&= a^2 + a\gamma - a^2 + b\gamma - ab && \text{(A.23)}\\
&= a\gamma + b(\gamma - a) && \text{(A.24)}\\
g(3) &= a^3 + (\gamma - a)\frac{a^3 - b^3}{a - b} && \text{(A.25)}\\
&= a^3 + (\gamma - a)(a^2 + ab + b^2) && \text{(A.26)}\\
&= a^3 + \gamma a^2 + \gamma ab + \gamma b^2 - a^3 - a^2 b - ab^2 && \text{(A.27)}\\
&= \gamma a^2 + \gamma ab + \gamma b^2 - a^2 b - ab^2. && \text{(A.28)}
\end{aligned}
$$

These agree with the original iterative equation. Okay, so this is a bit overkill since at most y^3 appears, so having general powers isn't too helpful. Nonetheless, we can now reduce the powers of y in the original equation, and see how that simplifies things.

A.2 Simplifying

We can now reduce the bracketed term in Eq. A.13:

$$
\begin{aligned}
& M^2 h_0 + M h_0 + M y^2 - M^2 y + y^3 + 2y^2 + y && \text{(A.29)}\\
&= M^2 h_0 + M h_0 + M^2 y + h_0 y M - h_0 M^2 - M^2 y + y M^2 + y M h_0 + y h_0^2 + && \text{(A.30)}\\
&\qquad - M^2 h_0 - h_0^2 M + 2My + 2h_0 y - 2h_0 M + y && \text{(A.31)}\\
&= \left(-M^2 y + y M^2 + M^2 y + y M h_0 + y h_0^2 + 2My + 2h_0 y + y + h_0 y M\right) + && \text{(A.32)}\\
&\qquad \left(M^2 h_0 + M h_0 - h_0 M^2 + -M^2 h_0 - h_0^2 M - 2h_0 M\right) && \text{(A.33)}\\
&= \left(2My + 2h_0 y + y + 2y M h_0 + M^2 y + y h_0^2\right) + \left(-M^2 h_0 - h_0^2 M - h_0 M\right) && \text{(A.34)}
\end{aligned}
$$

Having eliminated the "hidden" variables, let's try to re-group into powers of y:

$$
\begin{aligned}
&= y^2 + \left(My + h_0 y + y + 2y M h_0 + M^2 y + y h_0^2\right) + \left(-M^2 h_0 - h_0^2 M\right) && \text{(A.35)}\\
&= 2y^2 + y + 2y M h_0 + M^2 y + y h_0^2 + -M^2 h_0 - h_0^2 M + M h_0 && \text{(A.36)}\\
&= y^2 M + y + y^2 + y^2 h_0 + y^2 + M h_0 && \text{(A.37)}\\
&= y\left(yM + y h_0 + y + M + h_0 + 1\right) && \text{(A.38)}\\
&= y\left(y + 1\right)\left(M + h_0 + 1\right) && \text{(A.39)}
\end{aligned}
$$

Putting this into the corresponding term in Eq. A.7 gives

$$
\frac{1}{2y(y+1)} y(y+1)\left(M + h_0 + 1\right) = \frac{1}{2}\left(M + h_0 + 1\right) \qquad \text{(A.40)}
$$

and so finally we arrive at,

$$
\boxed{\langle D \rangle_{\text{overall}} = \frac{y(y+1)}{h_0(M+1)}\left[\frac{1}{2}\left(M + h_0 + 1\right) - \frac{1}{3}(2y+1)\right]}. \qquad \text{(A.41)}
$$

Appendix B

Recursive Analytic Solution Attempt

Here we will attempt to derive a more useful form of Eq. 2.32. We will start with a slightly simpler relation,

$$h_{n+1} = h_n^2 + h_n. \tag{B.1}$$

We note that in general, this will be an expansion in terms of h_0,

$$h_n = \sum_{i=1}^{m} a_i h_0^i. \tag{B.2}$$

Our goal then, is to determine the set of coefficients a_i. These coefficients implicitly depend on n. Our next iteration will look like,

$$h_{n+1} = \left(\sum_{i=1}^{m} a_i h_0^i\right)^2 + \sum_{i=1}^{m} a_i h_0^i. \tag{B.3}$$

The squared term is actually very messy to deal with, requires the use of the Multinomial Theorem which (for the specific case $p = 2$) states,

$$\left(\sum_{i=1}^{m} x_i\right)^2 = 2 \sum_{\{k_m\}=2} \prod_{t=1}^{m} \frac{x_t^{k_t}}{k_t!}, \tag{B.4}$$

where $\{k_m\} = 2$ denotes the sets of m integers which add to two. For $m = 4$ one example might look like: $(1, 0, 0, 1)$. For those of you familiar with this, these sets arise from integer partitioning, but we won't go over how these are generated, but just know there are algorithms for generating them. We will drop the $= 2$ portion for brevity. Our form can be obtained with the transformation $x_i \to a_i h_0^i$,

$$\left(\sum_{i=1}^{m} a_i h_0^i\right)^2 = 2 \sum_{\{k_m\}} \prod_{t=1}^{m} \frac{a_t^{k_t} h_0^{t k_t}}{k_t!} \tag{B.5}$$

$$= 2 \sum_{\{k_m\}} \prod_{t=1}^{m} \frac{a_t^{k_t}}{k_t!} h_0^{\sum_{t=1}^{m} t k_t}. \tag{B.6}$$

Now let's return to the general n'th term (Eq. B.3),

$$h_n = 2 \sum_{\{k_m\}} \prod_{t=1}^{m} \frac{a_t^{k_t}}{k_t!} h_0^{\sum_{t=1}^{m} t k_t} + \sum_{i=1}^{m} a_i h_0^i. \tag{B.7}$$

As-is, these cannot be combined. To handle this, we impose an ordering on $\{k_m\}$ so they they are given in increasing values of $\sum t k_t$, we will denote this as $\{k_m\}_i^L \uparrow$, where L indicates the sum's value for the current set, and i is the index of the current set. Now,

$$h_n = 2 \sum_{\{k_m\}_i^L \uparrow} \prod_{t=1}^{m} \frac{a_t^{k_t}}{k_t!} h_0^{L_m} + \sum_{i=1}^{m} a_i h_0^i. \tag{B.8}$$

Our only hope of combining these terms is to match up the powers. The good news is that now each term on the left, and right correspond to a single power.

$$h_n = 2 \sum_{\{k_m\}_i^L \uparrow} \left(\prod_{t=1}^{m} \frac{a_t^{k_t}}{k_t!} + a_i \right) h_0^{L_m}, \tag{B.9}$$

Now, we should start from the bottom ($n = 0$) and see if we can build up. For $n = 0$, $h_1 = h_0^2 + h_0$. So the coefficients are $\{a\}^1 = (1, 1, 0, 0, 0, ...)$, where the zero padding accounts for the fact that the powers on the linear sum are less than the powers on the quadratic sum.

We can read off the coefficient is we sort-of invert the sum notation. Let's instead sum across i.

$$h_n = 2 \sum_{i} \left(\prod_{\{k_t\}_i}^{m} \frac{a_t^{k_t}}{k_t!} + a_i \right) h_0^{L_i}. \tag{B.10}$$

Now the coefficient for the L_i'th power for the next iteration is

$$a_{L_i} = 2 \left(\prod_{\{k_t\}_i}^{m} \frac{a_t^{k_t}}{k_t!} + a_i \right). \tag{B.11}$$

This actually works out really well since the coefficients for our original problem simply get multiplied by γ! Let's make another simplification by letting k_t be represented by a vector, $\vec{k}$ and constructing respective operations,

$$a_{i,n+1} = 2 \frac{\boldsymbol{a_n^{k_i}}}{\boldsymbol{k_i!}} + 2 a_{i,n}. \tag{B.12}$$

Here a vector to the power of a vector is the product of the element-wise powers, and the factorial of a vector is the product of the element-wise factorials. This isn't very useful since it still requires an iterative scheme.

Appendix C

Recursive Approximation Justification

We begin with Eq. (2.34) which we will restate here with $h_m \to x_n$:

$$x_{n+1} = x_n - \frac{x_n}{2}\left(2 - \frac{x_n + 1}{M + 1}\right). \tag{C.1}$$

This formulation looks similar to Newtons method for finding the root of $f(x)$, which is given as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{C.2}$$

After some algebra we find that,

$$\frac{f(x_n)}{f'(x_n)} = x_n(1 - \gamma x_n). \tag{C.3}$$

This can be easily solved for $f(x)$ since this is a separable equation:

$$\frac{df}{f} = \frac{dx}{x(1 - \gamma x)} \tag{C.4}$$

$$\int \frac{df}{f} = \int \frac{dx}{x(1 - \gamma x)} \tag{C.5}$$

$$\ln(f) = \ln|x| - \ln|\gamma x - 1| + C \tag{C.6}$$

$$f(x) = e^C |x||\gamma x - 1| \tag{C.7}$$

The error, ϵ in the next iteration of Newtons method is given as,

$$\epsilon_{n+1} = \epsilon_n^2 \left|\frac{f''(r)}{2f'(r)}\right|, \tag{C.8}$$

where r is the root we desire. In this case the root we want is,

$$\lim_{n\to\infty} x_n = 0 \tag{C.9}$$

since this corresponds to the health reaching zero. The ratio becomes,

$$\left|\frac{f''(r)}{2f'(r)}\right| = \gamma, \tag{C.10}$$

which simplifies the error equation to,

$$\epsilon_{n+1} = \gamma \epsilon_n^2. \tag{C.11}$$

This is exactly the same recursive equation we had before! Except that now we have error in health instead of health. So how does this connect? Since we already know the root value (which is 0), the error becomes the upper bound on the health. This means that if we have an error of, for example, 1 that the health must be below that. So, by solving $\epsilon = h$ we can find the number of iterations required to reach below that health. This solution is already given earlier but in the context of error we have,

$$\epsilon_n = \frac{1}{\gamma}\left(\frac{1}{2} - \gamma\right)^{2^n} \tag{C.12}$$

$$n = \log_2 \log_{\frac{1}{2}-\gamma}(\gamma\epsilon). \tag{C.13}$$

This has done two things: justify the use exclusion of the h_m term in the original recursive equation (instead of excluding h_m^2), and provided a second interpretation of the meaning of $h = 1$. The error comes from a Taylor series, but interestingly all higher order terms die off so this is actually exact. This suggests that for higher precision, the assumption that ϵ is small is violated and the Taylor series formulation no longer holds.