# MATHEMATICAL GAME MODELING AND OPTIMIZATION

Optimizing Player Decisions in Old School Runescape

Palfore

June 23, 2024

# Contents

This text accompanies several additional resources:

1. Python open-source codebase, *OSRSmath*: https://github.com/Palfore/OSRSmath.
2. Video Series, *Optimizing Runescape*: https://www.youtube.com/watch?v=7N9UJX70Z5I&list=PLm3INE_scU5s8NQWmw0fxKtA_6SVxDOc7&ab_channel=Palfore.
3. Discord Chatroom: https://discord.gg/4SXcKQh

# Introduction

*Runescape* (RS) is a popular *Massively Multiplayer Online Role-Playing Game* (MMORPG) that was first publicly released on January 4'th, 2001 by the video game developer Jagex Limited. Ranked as the 5'th most popular MMORPG in 2020 by several sources, this game's unique mechanisms and game play make it still successful nearly 20 years after it's incarnation [1, 2, 3]. On the 20th of November 2012, a total overhaul to the game's combat system - an integral part of gameplay - caused a great divide among it's players. As a result, the game bifurcated into two versions: Runescape 3, and *Old School Runescape* (OSRS). The latter was released on February 22, 2013 and reverted to the old mechanics. The relative player counts over time can be found in Ref. [4]. OSRS currently contains the majority of players. To limit the already-large scope, this text will only focus on that version.

In typical role-playing fashion, the majority of game play centers around fighting monsters and bosses, training skills, completing quests, playing mini-games, and collecting items. This game is played over the course of months or years. In a few years, there will even be some players who have played for *decades*. As the player base gained a more comprehensive understanding of the game, their mentality has generally shifted from one of discovery to one of *efficiency*. Many tools have been created with the goal of improving player efficiency, optimizing game play, and maximizing success in difficult challenges.

There are 23 skills that a player can train [5]. A player is rewarded experience for certain actions related to a given skill. For example, cutting an `Oak Tree` yields 37.5 experience per log chopped. 83 Experience is required to go from level 1 to level 2, while reaching the maximum level of 99 requires 13,034,431 total experience [6]. The experience required to level up increases exponentially - hence the drive for efficiency [6]. There are several combat skills that directly influence a player's fighting ability. Quests are often completed for the special items, new training methods, and experience rewards they provide. They have skill requirements and often make use of combat in defeating difficult bosses. And so even in this basic overview, the complexity of the interactions and relations between different actions a player can perform becomes apparent.


**Figure 1:** Some relevant interfaces/images that play a central role in game play. The skill panel (left) shows the player's levels in the 23 skills along with their total level (Image slightly modified from Ref. [5]). The combat skills, attack, strength, defence, ranged, prayer, magic, and health (in the middle column), are respectively outlined in red. The quest panel (middle) shows the player's quests that are completed, in progress, and not started (green, yellow, and red, respectively. Image from Ref. [7]). A character that a player would control in a 3D world is shown on the right.

To understand player decisions and optimize them, we will be mathematically modeling the in-game mechanics. A surprising variety of mathematical concepts and techniques will be encountered. Additionally, algorithms derived from computer science are required to solve some of these problems. This serves as an exciting *field* to explore, with some very interesting results and visuals. Some of the details require high level mathematical solutions/descriptions.

# Part I

# General

# Chapter 1

# Experience and Levels

The experience required to reach a level $L$ is given by,

$$\mathcal{L}^{-1}(L) \equiv \left\lfloor \frac{1}{4} \sum_{l=1}^{L-1} \left\lfloor l + 300 \cdot 2^{l/7} \right\rfloor \right\rfloor \tag{1.1}$$

$$\approx \frac{1}{8} \left( L^2 - L + 600 \frac{2^{L/7} - 2^{1/7}}{2^{1/7} - 1} \right). \tag{1.2}$$

The level of a skill with $E$ experience is defined as:

$$\mathcal{L}(E). \tag{1.3}$$

No analytic form has been found for this equation, however it can be straightforwardly evaluated by numerically inverting Eq. (1.1) or by building lookup tables.

The level equation itself is exponentially slow, and this effect extends to the overall time to max all skills. For example, one youtuber, WildMudKip provides total level and date stamps through their HCIM youtube series, by scraping this data, Fig. 1.1 now shows that the total level over time is also logarithmic.

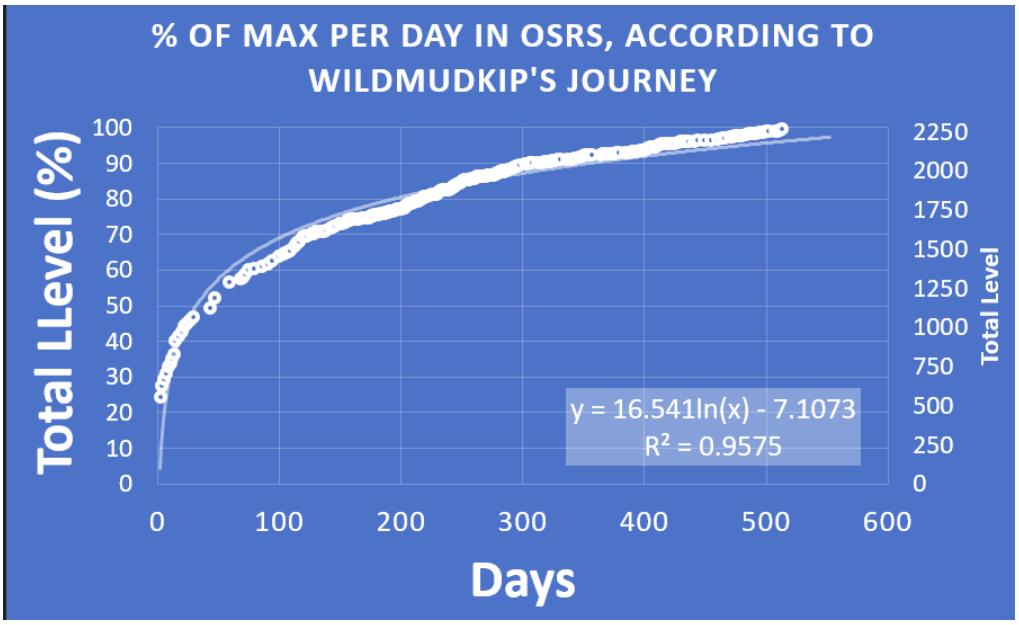



**Figure 1.1:** Youtuber WildMudKip's hardcore ironman levels over time reveal an logarithmically slow progression. Near optimal play and consistent upload and play time are all reasonable assumptions.

# Part II

# Combat

## List of Combat Related Terms and Associated Values

1. Combat Class: One of [`melee, ranged, magic`]
2. Attack Type: One of [`stab, slash, crush, ranged, magic`]
3. Combat Style: The name of the attack.
   (a) For the `melee` combat class: One of [`punch, kick, chop, hack, smash, block, pound, pummel, slash, lunge, jab, swipe, fend, spike, impale, reap, flick, lash, deflect, bash, focus, scorch`]
   (b) For the `ranged` combat class: One of [`accurate, rapid, longrange, short fuse, medium fuse, long fuse, flare`]
   (c) For the `magic` combat class: One of [`spell, spell (defensive), blaze, accurate, longrange`]. The latter two only apply to `Powered staves`.
4. Attack Style:
   (a) For the `melee` combat class: One of [`accurate, aggressive, defensive, controlled`]
   (b) For the `ranged` combat class: One of [`accurate, rapid, longrange`]
   (c) For the `magic` combat class: One of [`standard, defensive`]
5. Attack Speed: The number of ticks between attacks. Integer between 1 and 15.
6. Attack Interval: The number of second between attacks. Real number between 0.6s and 9s.
7. Attribute: When referring to an opponent/monster, their `attribute` is one of [`Demon, Draconic, Fiery, Kalphite, Leafy, Penance, Shade, Undead, Vampyre, Xerician`]. In addition, we expand this to also include properties like: [`On slayer task, In wilderness`].

# Chapter 1

# Overview

In this chapter, we will discuss the various factors involved in combat. We will consider combat in two stages. The first considers an autonomous fight in which the player performs no actions once the initial conditions of the fight have been specified. Analyzing this system will allow us to calculate quantities like the expected number of attacks required to defeat an opponent, and the probability of winning a fight. The second considers active player decisions that occur during combat. This will allow us to investigate the effect of performing actions on the aforementioned quantities. *Policies* may be defined to mathematically model a player's decision. As an example, a player may use a healing item any time throughout a fight. To handle this, we can consider a specific policy whereby the player will use a healing item when health is below some threshold. Investigating this threshold will give us insight into how players should use healing mechanics.

It is interesting to note that although the descriptions of in-game mechanics likely have no real-world connections (since they are somewhat arbitrarily decided by the game's developers), the mathematics that can be applied to the dynamic variables resulting from these mechanics can actually be applied and generalized to real-world settings. We will begin with a discussion of the most relevant mechanics, however there is an additional large body of information that can be found on the Official Wiki that provides a greater overview.

## 1.1 Autonomous Mechanics

### 1.1.1 Combat Skills, Combat Triangle and Attack Styles

Combat is built around the so-called *Combat Triangle* which describes the relation between the three classes of combat in the game [8]. A Melee fighter makes use of close quarters combat, typically wielding swords, daggers, halberds, etc. A Ranged fighter makes use of bow and arrow, crossbows, and thrown objects to deal damage at a distance. Finally, a Mage will make use of staves and magical spells to do damage, also at a distance. The combat triangle refers to the notation that melee users are (generally) weak to magic, which is weak to ranged, which is weak to melee, and is depicted in Fig. 1.1.

Some skills provide benefits to all fighters, while others are specific to the style:

1. Attack, $L_a$: Increases the accuracy of a melee attacker.
2. Strength $L_s$: Increases the maximum damage a melee attacker can do in a single attack.

3. Ranged $L_r$: Increases the accuracy and maximum damage of a ranged attacker.

4. Magic $L_m$: Most spells have a constant damage (with more powerful spells being unlocked at higher levels), also some scale with magic level. Accuracy however is generally increased with higher magic. In addition, defence against magical attacks is partially determined by the player's magic level.

5. Defence $L_d$: Decreases the probability that the opponent will have a sucessful attack.

6. Prayer $L_p$: Acts as a depleteable resource that can boost combat skills.

7. Hitpoints $L_h$: Increases the amount of damage a player can receive before they lose a fight.

The set of all combat levels is denoted $\{L\}$.

Every weapon has a set of *attack styles* that allow a player to change which combat skill they train. In addition, the attack style may provide a small bonus to combat. Prayer is the only skill that cannot be trained directly through combat. Hitpoints is another exception in that a proportion of the experience awarded to the skill associated with the player's attack style is given to hitpoints.

### 1.1.2 Equipment Bonuses

Let's begin discussing a fighter's equipment by defining an *item*, $\mathcal{I}$. Equipable items can be worn in one of 11 slots. We let $\mathcal{I}^{\text{slot}}$ represent the item in a given *slot*, where

$$\text{slot} \in \{\text{head, cape, neck, ammo, weapon, torso, shield, legs, hands, feet, ring}\}. \quad (1.1)$$

Each item has some associated equipment bonuses. Most of these are constant, however some bonuses are conditional. The constant bonuses can be represented as a vector:

$$\vec{\mathcal{I}}_c^{\text{slot}} = (A_{\text{stab}}, A_{\text{slash}}, A_{\text{crush}}, A_{\text{magic}}, A_{\text{ranged}}, \quad (1.2)$$

$$D_{\text{stab}}, D_{\text{slash}}, D_{\text{crush}}, D_{\text{magic}}, D_{\text{ranged}}, \quad (1.3)$$

$$S_w, S_r, S_m, P, w, r). \quad (1.4)$$

There are many terms to define, so we will explain them here. $A$, $D$, and $S$ refers to the attack, defensive, and strength bonuses, respectively. The attack and defence bonuses are associated with the different attack types, while the strength bonuses are associated with the combat class [SEE LIST OF TERMS]. The first three attack and defence bonuses listed are associated with melee combat, the last two are associated with magic, and ranged, respectively. There is a strength bonus associated with each combat class. In the order above we have: melee/warrior, ranged, then magic. The prayer bonus, $P$ affects how long bonuses from the prayer skill can last without recharging. $w$ is the weight of the item. Finally, if the item is a weapon, $r$ is the attack rate given by $r = 1/s$, where $s$ is the weapon attack speed. If it is not a weapon, $r = 0$. Note that we use the rate since every other bonuses improves fighter ability. This allows us to use a basic comparison operator (at the cost of using real numbers instead of integers).

The total constant equipment bonuses that a fighter has, $E_c$ is given as the sum over all the slots,

$$\vec{E}_c = \sum_{\text{slot}\in\{\text{slots}\}} \vec{\mathcal{I}}_c^{\text{slot}} \mathcal{E}. \quad (1.5)$$

The in-game interface indicating these values is shown in Fig. 1.1. There are a number of conditional effects that may not appear in this interface.

The conditional bonuses can be further divided into special/attribute bonuses and equipment set bonuses. Monsters may have a particular weakness due to their so-called attribute. For example, a `Iron dragon` would be `dragonic`, and `dragonbane` weapons would provide an accuracy and damage multiplier. In this sense, we can consider these bonuses to be dependent on information that the item itself does not know, and so we represent these special bonuses as an operator $\hat{\mathcal{I}}_s$. When acting on a fighter's environment $\mathcal{E}$, these bonuses become concrete:

$$\vec{E}_s = \hat{\mathcal{I}}_s \mathcal{E} \tag{1.6}$$

Set effects are also similar except that they are conditional on equipment the player is wearing. For this reason, (and the fact that there are other special cases), we group all these effects into the special bonus operator, $\hat{\mathcal{I}}_s$ from above. The total bonuses from all the player's items can be represented as:

$$\vec{E} = \vec{E}_c \cup \vec{E}_s \tag{1.7}$$

$$= \sum_{\text{slot} \in \{\text{slots}\}} \vec{\mathcal{I}}_c^{\text{slot}} \cup \hat{\mathcal{I}}_s \mathcal{E}, \tag{1.8}$$

The definition of environment is intentionally vague, as there are a myriad of conditions, essentially limited only by developer imagination and infrastructure. Some of these conditions/dependencies include: attacker & opponent equipment & levels, attack style (which implies combat class), whether a particular `Diary` is completed, the `attribute` of the opponent, and so on. The elements and details of $\vec{E}_s$ are also purposefully vague, as there is an additional caveat that makes these a bit trickier to handle both mathematically but more-so programatically. Unlike the constant bonuses, which can be added together, special bonuses are generally multiplicative but also make use of intermediate *flooring*. This makes the special bonus operator non-commutative, since the order does effect the rounding.[1] This means that a vector representing special bonuses would essentially have as many elements as the number of special items! So it is often easier to work on each bonus type with different methods. For this reason, special effects and constant bonuses are treated independent, making the union above more symbolic than practical.

### 1.1.3 Max Hits and Accuracy

Due to the large complexity and number of exceptions in the game, defining mathematical formulas for max hit and accuracy is very difficult/tedious. Instead, programming is the language for this. If you want to learn more about how max hits and accuracies are calculated see Ref. [9]. You can also view a python translation of that code in the combat directory of the codebase accompanying this document.

### 1.1.4 Ticks and Attack Speed

At a fundamental level the entire game operates on a tick-based system. Every 0.6 seconds (called a tick) the game updates. This discretizes the possible game states, and typically means we will be dealing with sums in place of integrals, and recursive equations in place of differential equations.

Once an attacker begins combat with an opponent, the fight continues until either is defeated, or one runs away. The attacks occur at an interval associated with the weapon. Different weapons have different *Attack Speeds*, typically between 3-9 game ticks (1.8s

[1] The *specific* ordering of the flooring operations is taken from ref. [bitter-dps calc]. Although, this author is unsure if that ordering is arbitrary, but we assume not. [Reference max hit section?]


**Figure 1.1:** The attack styles (left), equipment slots and associated equipment bonuses (middle) along with a depiction of the combat triangle (right). The attack styles for the `Dragon Claws` are Chop, Slash, Lunge, Block and give experience specifically to Attack, Strength, shared, and Defence, respectively. Shared means experience is split equally. In the equipment panel, the player is not wearing any equipment which results in 0 bonuses for all attributes. Starting with the bottom left of the combat triangle, a mage has advantage over the melee equipment typically worn by a melee fighter, a melee warrior has an advantage over the equipment typically worn by a ranged fighter, and ditto for ranged to mage.

- 5.4s). The attacker's attack speed $\mathcal{A}|s$, is the number of ticks between attacks. On each attack, the player's accuracy will determine the probability of a *successful hit*. On a successful hit, a number between 0 and the player's maximum hit will be uniformly sampled as the damage the player does.

A notable consequence of this tick-based system is that a series of precise player actions known as tick-manipulation allows players to perform multiple actions in a single tick, or to take advantage of mechanisms like tick-eating, allowing a player to survive otherwise fatal attacks.

### 1.1.5 Summary

A fighter has some combat skill levels and will (typically) equip some armour and a weapon. They will select an attack style, which selects the skill they will receive experience in, and which equipment bonuses plays a roll in the accuracy calculation. The problem then reduces to considering an accuracy and maximum hit. Once a fight begins, an attack occurs every couple of ticks. If the attack is successful, a uniform integer between 0 and their max hit is delivered to the opponent, reducing their current hit points. Once a fighter's health reaches 0, the fight is over.

## 1.2 Agency & Decisions

The autonomous mechanics allow us to predict the outcome of a fight given that neither the player nor the opponent can alter the game state. To describe more advanced combat, like fighting bosses, players, or high-level monsters, we must consider the ability of a fighter to perform actions. In the simplest incarnation, a fighter can be represented as a vector containing five values:

$$\vec{F} = (H, m, a, w, t), \tag{1.9}$$

where $H$ is the fighter's current health, $m$ is the max hit, $a$ is the accuracy, $w$ is the weapon attack speed, and $t$ is the tick counter. The tick counter in this context is the number of ticks until the player can attack again. This ignores mechanics like `poison` and attacks with non-standard damage distributions but acts as a good base. More complex interactions would require increasing the number of considerations. For

a fight that involves two combatants $\vec{F}_1$ and $\vec{F}_2$, the game state can be represented as a concatenation of the two fighters:

$$\vec{G} \equiv \vec{F}_1 \oplus \vec{F}_2. \tag{1.10}$$

### 1.2.1 Actions

In general, we consider a model $\hat{M}$ (this may be a computation algorithm, or a human brain) that aims to maximize certain objectives, like the probability of winning a fight. On each game tick, this model would consider the game state and produce a set of actions to perform,

$$\hat{M}\vec{G} = \vec{D}. \tag{1.11}$$

To provide a concrete representation of this decision vector, we consider that each element corresponds to an action. The value of an element could be taken to be the *confidence* in performing that action. A natural representation would be letting each confidence be a real number $\in [-1, 1]$, where values above 0 would result in that action being performed. These confidences are converted to actions through a sort of look up operator, $\hat{\mathcal{A}}$,

$$\hat{\mathcal{A}}\vec{D} = \vec{\theta}. \tag{1.12}$$

which act on the game state to generate the next state.

$$\hat{\theta}\vec{G}_n = \vec{G}_{n+1}, \tag{1.13}$$

where now a subscript $n$ is used to denote the turn number.

There are several different actions that are possible, and several may occur on the same game tick.

1. Attack: $\hat{A}_1$ — if $F_1^t = 0$: $F_1^t \to F_1^w, F_2^H - \mathcal{D}(F_1^m, F_1^a)$,
2. Tick-Down: $\hat{T}_1$ — if $F_1^t \neq 0 : F_1^t \to F_1^t - 1$,

where $\mathcal{D}$ is the damage distribution function which draws a random number corresponding to the damage inflicted.

As an example, if both fighters decide to attack, then the game state would change as follows:

$$\hat{T}_1\hat{T}_2\hat{A}_1\hat{A}_2\vec{G}_n = \vec{G}_{n+1}. \tag{1.14}$$

We will make the assumption that all actions are processed on the previous game state, i.e. the ordering of actions doesn't matter since they all act directly on $\vec{G}_n$:

$$\hat{A}_1\hat{A}_2\vec{G}_n = \hat{A}_2\hat{A}_1\vec{G}_n. \tag{1.15}$$

If we consider that the game state after $n$ turns depends on the models of both fighters, then a reasonable objective for the first fighter is:

$$\arg\min_{\hat{M}_1} n \,|\, F_2^H(G_n^{\hat{M}_1,\hat{M}_2}) = 0, F_1^H(G_n^{\hat{M}_1,\hat{M}_2}) > 0 \ \forall \ \hat{M}_2. \tag{1.16}$$

In other words, the player is trying to find the model (decision-making scheme) that minimizes the number of turns for the second fighter's health to reach 0, while staying alive, for an arbitrary opponent model. [But this is a random process].

### 1.2.2 Machine Learning

For humans, these decisions making skills can be incredibly quick and precise [cite high-level tri-briding or something]. The ingenious ways that players have been able to take advantage of the game mechanics to develop a wide range of interesting behavior is really fascinating. The intricacies and reactions involved result in numerous interesting approaches to combat:

1. Pures:
2. 1-Iteming
3. Hybriding
4. Tri-bridding
5. Dwh Pure
6. Tick-Eating
7. Poison
8. Kiting

In addition, there have been many technical feats that take advantage of intricate mechanics to win impressive challenges under heavy restrictions:

1. The succession of Combat Level [59, 40, 25, 13, 9, 4,] and ultimately 3 (the lowest possible) `Fire Capes`.
2. No-iteming Corp

To study these solutions, and to gain insight into the decisions that players make, we aim to make use of machine learning techniques to solve this complex problem. Assuming that the networks will be able to find similar solutions to these problems as the humans, it would be fascinating to formally describe what the players are processing and what features they consider when they react in certain ways. Without delving into the details, we will be employing feed-forward neural networks as the model which will be optimized to solve this problem. This will required the use of reinforcement learning techniques since exact solutions (the optimal set of decisions in a fight) is not known for any cases and would not be able to be generated in general. [the optimiZatino of this ishard ]

### 1.2.3 Genetic Algorithm

# Chapter 2

# Models

This chapter presents an assortment of approaches that have been used to attempt to model combat. Ultimately, these are all approximations. Thankfully, an exact solution can be found through a Markov Chain analysis, which will be presented in the next chapter. The models here vary in how they handle overkill and they will be presented roughly in order of increasing complexity. They operate under 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,

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

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

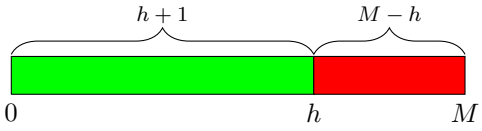
$$\boxed{n = \frac{6h_0^2(M+1)}{y(y+1)\,(3M + 3h_0 - 4y + 1))}} \tag{2.11}$$

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

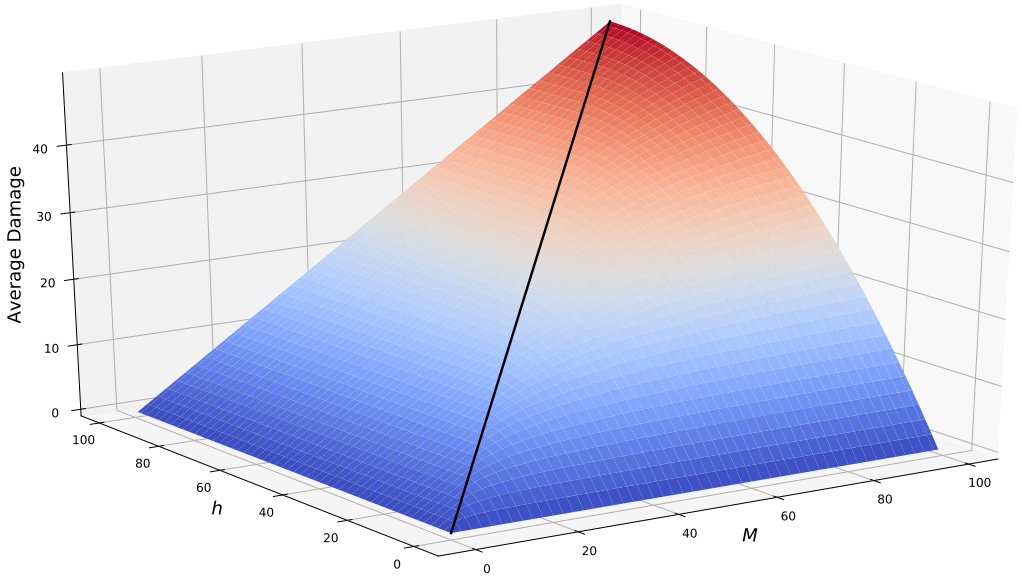



**Figure 2.1:** The average damage of an attack with a maximum hit of $M$ on an opponent with $h$ health. The black line is the piecewise boundary.

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 B. 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. 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 A. 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,

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

### 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 $\tau$ seconds, then there are $T_R/\tau$ 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 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

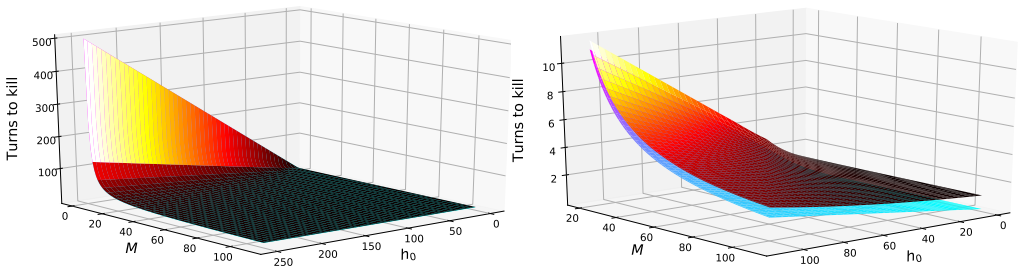



**Figure 2.2:** The number of successful turns to kill an opponent that has $h_0$ health, given that the attacker's max hit is $M$. The MarkovChain (hot colors) and Crude (cool colors) methods are shown. However, due to the scale of the left plot they are indistinguishable. The right plot shows a smaller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

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.

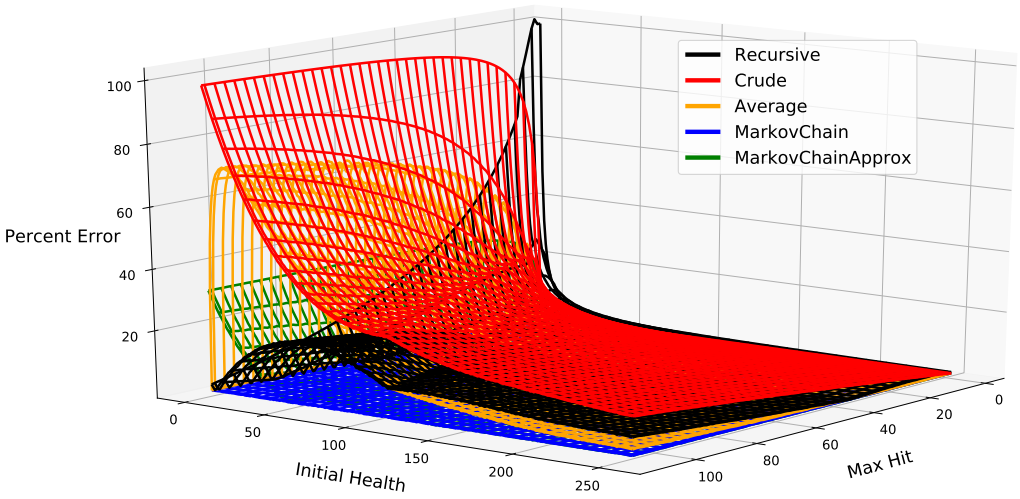



**Figure 2.3:** A comparison between different models. The percent relative error with respect to the simulations are shown.

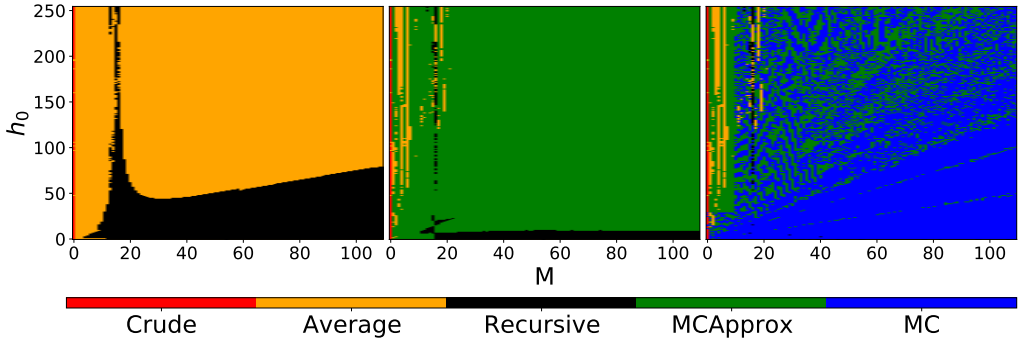



**Figure 2.4:** The model which agrees best with simulation, for different initial healths, and maximum hits. The final plot shows a comparison which includes all models. Since the MarkovChain (MC) models have very low errors, they occlude the other details. So, the other plots only include a subset 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.

# Chapter 3

# Markov Chain Solutions

## 3.1 Damage Distributions

Under *standard* conditions, an attacker's damage distribution, given by $\{c_i\}$ is a uniform distribution from 1 to their max hit with some additional weighting on 0 due to accuracy. This can be generalized to arbitrary probability distributions, allowing consideration of special equipment (like Keras, Verac's, etc.) as well as combining distributions in the case of multiple opponents/attacks. To describe this, we define the following terms:

$$c_i \text{ is the probability of doing } i \text{ damage} \tag{3.1}$$
$$c_0 \text{ is the probability of doing zero damage} \tag{3.2}$$
$$c_+ = 1 - c_0 \text{ is the probability of doing damage} \tag{3.3}$$
$$m \text{ is the largest non-zero probability from } c_i \tag{3.4}$$
$$\text{No healing allowed.} \tag{3.5}$$

Damage distributions are generally defined as any probability distribution, although in practice the in-game distributions come from a fixed set, as described in Fig. 3.1. The suggestion here is that damage distributions constitute an additional dimension for the developers to use (previously only max hit and accuracy were the only parameters). Since this suggestion several items have emerged with special damage distributions (eg: Osmumten's fang). The math further suggests variable weapon attack speeds as a way of generalizing the integer time steps. This is reflected in new weapons which strike twice (eg: Dual Macuahuitl), and in which delays are added.

## 3.2 Recursive Equation

In one attack, the opponent can be brought to a given health $i$, from their initial health $h$ according to the transition probability, [10]

$$\pi_{h,i} = \begin{cases} P(X = h - i) & \text{if } i > 0 \\ P(X \geq h) & \text{if } i = 0. \end{cases} \tag{3.6}$$

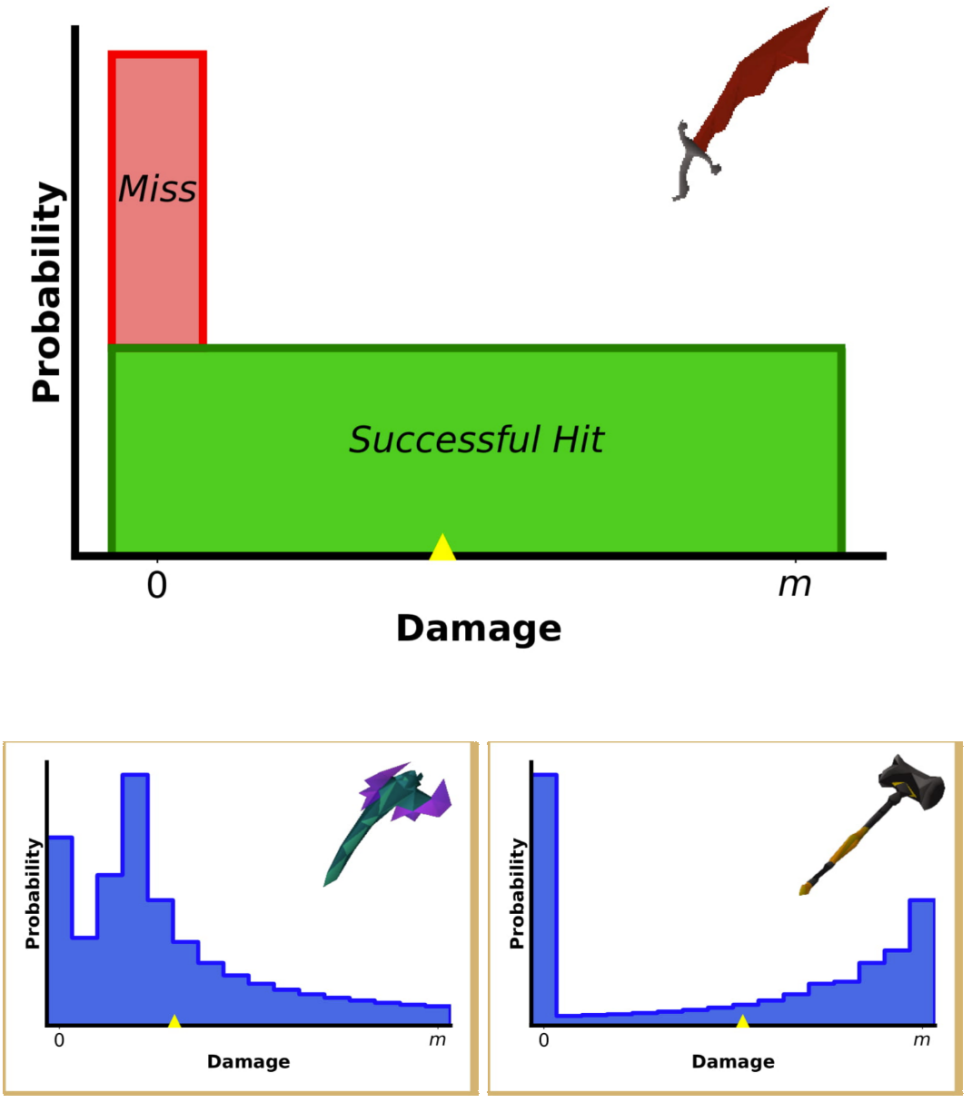



**Figure 3.1:** The standard damage distribution is shown on top, which consists of a portion to hit successfully and a chance to miss (both based on the accuracy roll). Although the developers modified this distribution in 2024 to enforce that successful hits do at least 1 damage, thankfully all the mathematics still follows (showcasing the benefit of a general solution!). In the bottom are theoretical distributions of hypothetical weapons. On the left, a quick firing weapon has a high chance to do low damage. On the right, a huge maul misses very often but when it hits the damage is very high. Other distributions already exist in-game like the scythe which has a Gaussian distribution and Keras which are considered as two merged distributions.

The probability they are killed in $L$ turns can be given by the sum of the probabilities the opponent was brought to $i$, then killed in $L-1$ turns:

$$P_{h,L} = \sum_{i=0}^{\infty} \pi_{h,i} P_{i,L-1} \tag{3.7}$$

$$= \cancel{\pi_{h,0} P_{0,L-1}} + \pi_{h,h} P_{h,L-1} + \sum_{i=1}^{h-1} \pi_{h,i} P_{i,L-1} + \sum_{i=h+1}^{\infty} \pi_{h,i} P_{i,L-1} \tag{3.8}$$

$$= c_0 P_{h,L-1} + \sum_{i=1}^{h-1} \pi_{h,i} P_{i,L-1} + \cancel{\sum_{i=h+1}^{\infty} \pi_{h,i} P_{i,L-1}} \tag{3.9}$$

$$P_{h,L} = c_0 P_{h,L-1} + \sum_{i=\max(h-m,1)}^{h-1} c_{h-i} P_{i,L-1} \tag{3.10}$$

$$\tag{3.11}$$

where in the second line, the $i = 0, h$ terms are explicitly considered, and the remaining sum is split in two. In the third line, $\pi_{h,i}$ would correspond to healing which we are taking to be zero. In the final line, we get the lower bound by considering that

$$1 \leq i \leq h-1 \implies \pi_{h,i} = c_{h-i} \text{ if } 1 \leq h-i \leq m \text{ otherwise } 0, \tag{3.12}$$

$$\implies 1 \leq h-i \text{ and } h-i \leq m \tag{3.13}$$

$$\therefore i \leq h-1 \text{ and } h-m \leq i. \tag{3.14}$$

Since the first condition is already met, we have that $i \geq h-m$, but $i$ also cannot be below 1, hence $i \geq \max(h-m, 1)$.

## 3.3 Solution

The recursive equation to solve is:

$$P_{h,L} = c_0 P_{h,L-1} + \sum_{i \in I_h} c_{h-i} P_{i,L-1}, \;\; L \geq 2, h \geq 1 \tag{3.15}$$

$$\tag{3.16}$$

where $I_h$ is the set of integers satisfying $h-1 \geq i \geq \max(h-m, 1)$. The boundary conditions are given by:

$$P_{h,1} = \sum_{i=h}^{m} c_i \text{ (P of doing more than h damage)} \tag{3.17}$$

$$= \theta(m-h) \sum_{i=h}^{m} c_i \tag{3.18}$$

$$P_{1,L} = c_+ c_0^{L-1} \text{ (P of missing } L-1 \text{ times, then hitting any damage} \tag{3.19}$$

$$P_{1,1} = c_+ \text{ (P of doing any damage)} \tag{3.20}$$

Using a generating function:

$$g(x,y) = \sum_{h=1}^{\infty}\sum_{L=1}^{\infty} P_{h,L} y^h x^L \tag{3.21}$$

$$= \sum_{h=1}^{\infty}\left(P_{h,1} y^h x + \sum_{L=2}^{\infty} P_{h,L} y^h x^L\right) \tag{3.22}$$

$$= \sum_{h=1}^{\infty} P_{h,1} y^h x + \sum_{h=1}^{\infty}\sum_{L=2}^{\infty} P_{h,L} y^h x^L \tag{3.23}$$

$$= xyP_{1,1} + \sum_{h=2}^{\infty} P_{h,1} y^h x + \sum_{L=2}^{\infty}\left(P_{1,L} y x^L + \sum_{h=2}^{\infty} P_{h,L} y^h x^L\right) \tag{3.24}$$

$$= xyP_{1,1} + x\sum_{h=2}^{\infty} P_{h,1} y^h + \sum_{L=2}^{\infty} P_{1,L} y x^L + \sum_{L=2}^{\infty}\sum_{h=2}^{\infty} P_{h,L} y^h x \tag{3.25}$$

$$= xyP_{1,1} + x\sum_{h=2}^{\infty} P_{h,1} y^h + y\sum_{L=2}^{\infty} P_{1,L} x^L + \sum_{L=2}^{\infty}\sum_{h=2}^{\infty} P_{h,L} y^h x^L \tag{3.26}$$

These are the boundaries of a grid (corner + top + left) plus a sum over the interior.

### 3.3.1 Interior

Let's first tackle the interior:

$$\sum_{L=2}^{\infty}\sum_{h=2}^{\infty} P_{h,L} y^h x^L = \sum_{L=2}^{\infty}\sum_{h=2}^{\infty}\left(c_0 P_{h,L-1} + \sum_{i\in I_h} c_{h-i} P_{i,L-1}\right) y^h x^L \tag{3.27}$$

$$= \sum_{L=2}^{\infty}\sum_{h=2}^{\infty} c_0 P_{h,L-1} y^h x^L + \sum_{L=2}^{\infty}\sum_{h=2}^{\infty}\sum_{i\in I_h} c_{h-i} P_{i,L-1} y^h x^L \tag{3.28}$$

$$\mathcal{I}(x,y) = G(x,y) + R(x,y). \tag{3.29}$$

Let us also tackle this individually, starting with the $G$ term:

$$G(x,y) = \sum_{L=2}^{\infty}\sum_{h=2}^{\infty} c_0 P_{h,L-1} y^h x^L = c_0\sum_{L=1}^{\infty}\sum_{h=2}^{\infty} P_{h,L} y^h x^{L+1} \tag{3.30}$$

$$= c_0 x\sum_{L=1}^{\infty}\sum_{h=2}^{\infty} P_{h,L} y^h x^L \tag{3.31}$$

$$= c_0 x\sum_{L=1}^{\infty}\left(\sum_{h=1}^{\infty} P_{h,L} y^h x^L - P_{1,L} y x^L\right) \tag{3.32}$$

$$= c_0 x\sum_{L=1}^{\infty}\sum_{h=1}^{\infty} P_{h,L} y^h x^L - c_0 xy\sum_{L=1}^{\infty} P_{1,L} x^L \tag{3.33}$$

$$= c_0 x g(x,y) - c_0 xy\sum_{L=1}^{\infty} P_{1,L} x^L \tag{3.34}$$

$$\tag{3.35}$$

For $R$, we will first need a term that tells us whether $h$ is in the set $I$, i.e. does $h$ satisfy $h - 1 >= \max(m - h, 1)$? You will actually see that we need the more general

$h-1 >= \max(m-h+n, 1)$. We will call this condition $\delta^n_{m,h}$ which is 1 if satisfied and 0 otherwise. Empirically, this can be expressed as:

$$\delta^n_{m,h} = \begin{cases} 0 & \text{if } h = 1 \\ 0 & \text{if } n > m \\ 1 & \text{otherwise} \end{cases} \tag{3.36}$$

Since $h >= 2$,

$$\delta^n_{m,h} = \delta^n_m = \begin{cases} 1 & \text{if } n \leq m \\ 0 & \text{otherwise} \end{cases} \tag{3.37}$$

Now solving $R(x,y) = \sum_{L=2}^{\infty} \sum_{h=2}^{\infty} \sum_{i \in I} c_{h-i} P_{i,L-1} y^h x^L$ gives:

$$R(x,y) = \sum_{L=2}^{\infty} \sum_{h=2}^{\infty} \sum_{i=\max(h-m,1)}^{h-1} c_{h-i} P_{i,L-1} y^h x^L \tag{3.38}$$

$$= x \sum_{L=1}^{\infty} \sum_{h=2}^{\infty} \sum_{i=\max(h-m,1)}^{h-1} c_{h-i} P_{i,L} y^h x^L \tag{3.39}$$

$$= xy \sum_{L=1}^{\infty} \sum_{h=1}^{\infty} \sum_{i=\max(h-m+1,1)}^{h} c_{h-i} P_{i,L} y^h x^L \tag{3.40}$$

$$= xy \sum_{L=1}^{\infty} \sum_{h=1}^{\infty} \left( c_0 P_{h,L} \delta^1_m + \sum_{i=\max(h-m+1,1)}^{h-1} c_{h-i} P_{i,L} \right) y^h x^L \tag{3.41}$$

$$= c_0 xy \sum_{L=1}^{\infty} \sum_{h=1}^{\infty} P_{h,L} x^L \delta^1_m + xy \sum_{L=1}^{\infty} \sum_{h=1}^{\infty} \sum_{i=\max(h-m+1,1)}^{h-1} c_{h-i} P_{i,L} y^h x^L \tag{3.42}$$

$$\tag{3.43}$$

This part is outdated/incomplete, each $y$ should have a $c$. Notice that the $h = 1$ term

in the second set of sums yields 0.

$$= c_0 xyg(x,y)\delta_h^1 + xy\sum_{L=1}^{\infty}\sum_{h=2}^{\infty}\sum_{i=\max(h-m+1,1)}^{h-1} c_{h-i}P_{i,L}y^h x^L \tag{3.44}$$

$$= c_0 xyg(x,y)\delta_h^1 + xy^2\sum_{L=1}^{\infty}\sum_{h=1}^{\infty}\sum_{i=\max(h-m+2,1)}^{h} c_{h-i}P_{i,L}y^h x^L \tag{3.45}$$

$$= c_0 xyg(x,y)\delta_h^1 + xy^2\sum_{L=1}^{\infty}\sum_{h=1}^{\infty}\left(\sum_{i=\max(h-m+2,1)}^{h} c_{h-i}P_{i,L}\right)y^h x^L \tag{3.46}$$

$$= c_0 xyg(x,y)\delta_h^1 + xy^2\sum_{L=1}^{\infty}\sum_{h=1}^{\infty}\left(P_{i,L}\delta_m^2 + \sum_{i=\max(h-m+2,1)}^{h} c_{h-i}P_{i,L}\right)y^h x^L \tag{3.47}$$

$$= c_0 xyg(x,y)\delta_h^1 + xy^2\sum_{L=1}^{\infty}\sum_{h=1}^{\infty}P_{i,L}\delta_m^2 + c_+ xy^2\sum_{L=1}^{\infty}\sum_{h=1}^{\infty}\sum_{i=\max(h-m+2,1)}^{h} c_{h-i}P_{i,L}y^h x^L \tag{3.48}$$

$$= c_0 xyg(x,y)\delta_h^1 + xy^2 g(x,y)\delta_m^2 + xy^2\sum_{L=1}^{\infty}\sum_{h=1}^{\infty}\sum_{i=\max(h-m+2,1)}^{h} c_{h-i}P_{i,L}y^h x^L \tag{3.49}$$

$$\tag{3.50}$$

These series continues until the 'engine' producing terms has no more. The number of terms in this series is given by the maximum $n$ that is non-zero:

$$\arg\max_n \delta_m^n = m, \tag{3.51}$$

and so,

$$R(x,y) = xg(x,y)\sum_{i=1}^{m} c_i y^i \tag{3.52}$$

Now let's combine everything:

$$g(x,y) = xyP_{1,1} + x\sum_{h=2}^{\infty}P_{h,1}y^h + y\sum_{L=2}^{\infty}P_{1,L}x^L + c_0 xg(x,y) - c_0 xy\sum_{L=1}^{\infty}P_{1,L}x^L + xg(x,y)\sum_{i=1}^{m}c_i y^i \tag{3.53}$$

$$= xyP_{1,1} + x\sum_{h=2}^{\infty}P_{h,1}y^h + \left(y\sum_{L=2}^{\infty}P_{1,L}x^L - c_0 xy\sum_{L=1}^{\infty}P_{1,L}x^L\right) + xg(x,y)\sum_{i=0}^{m}c_i y^i \tag{3.54}$$

$$= x\sum_{h=1}^{\infty}P_{h,1}y^h + y\left(\sum_{L=2}^{\infty}P_{1,L}x^L - c_0\sum_{L=1}^{\infty}P_{1,L}x^{L+1}\right) + xg(x,y)\sum_{i=0}^{m}c_i y^i \tag{3.55}$$

$$= x\sum_{h=1}^{\infty}P_{h,1}y^h + y\left(\sum_{L=2}^{\infty}P_{1,L}x^L - c_0\sum_{L=2}^{\infty}P_{1,L-1}x^L\right) + xg(x,y)\sum_{i=0}^{m}c_i y^i \tag{3.56}$$

$$= x\sum_{h=1}^{\infty}P_{h,1}y^h + y\sum_{L=2}^{\infty}\left(P_{1,L}x^L - c_0 P_{1,L-1}\right)x^L + xg(x,y)\sum_{i=0}^{m}c_i y^i \tag{3.57}$$

Here, we will make a simplifying assumption that

$$P_{1,L}x^L = c_0 P_{1,L-1}, \;\; L \geq 2 \tag{3.58}$$

which is the condition that [killing an opponent] in $L$ turns is done by missing once then killing them in $L-1$ turns. This does *not* hold if healing is allowed. Under this assumption we are left with:

$$g(x,y) = x\sum_{h=1}^{\infty} P_{h,1}y^h + xg(x,y)\sum_{i=0}^{m} c_i y^i \tag{3.59}$$

$$g(x,y) - xg(x,y)\sum_{i=0}^{m} c_i y^i = x\sum_{h=1}^{\infty} P_{h,1}y^h \tag{3.60}$$

$$g(x,y)\left[1 - x\sum_{i=0}^{m} c_i y^i\right] = x\sum_{h=1}^{\infty} P_{h,1}y^h \tag{3.61}$$

$$g(x,y) = \frac{x\sum_{h=1}^{\infty} P_{h,1}y^h}{1 - x\sum_{i=0}^{m} c_i y^i} \tag{3.62}$$

To simplify, let's define:

$$T(x) \equiv \sum_{h=1}^{\infty} P_{h,1}y^h \equiv \sum_{h=1}^{m} t_h y^h \tag{3.63}$$

$$B(y) \equiv \sum_{i=0}^{m} c_i y^i, \tag{3.64}$$

Now $g(x,y)$ can be written as,

$$g(x,y) = \frac{xT(y)}{1 - B(y)x}. \tag{3.65}$$

### 3.3.2 Obtaining Power Series

Let us start with:

$$\frac{1}{1 - B(y)x} = \sum_{k=0}^{\infty} B^k(y)x^k \tag{3.66}$$

$$\tag{3.67}$$

We need to workout $B^k(y)$:

$$B^k(y) = \left(\sum_{j=0}^{m} c_j y^j\right)^k \tag{3.68}$$

$$= \sum_{j=0}^{km} d_j^k y^j, \tag{3.69}$$

where $d_j^k$ is the $j$'th element of the $k$'th convolution of $c$, which doesn't have an analytic form. In practice the `numpy` function `numpy.polynomial.polynomial.polypow` can be

used to compute this. Then,

$$\frac{1}{1-B(y)x} = \sum_{k=0}^{\infty}\left[\sum_{j=0}^{km} d_j^k y^j\right] x^k \tag{3.70}$$

$$\implies \frac{x}{1-B(y)x} = \sum_{k=0}^{\infty}\left[\sum_{j=0}^{km} d_j^k y^j\right] x^{k+1} \tag{3.71}$$

$$= \sum_{k=1}^{\infty}\sum_{j=0}^{(k-1)m} d_j^{k-1} y^j x^k \tag{3.72}$$

Multiplying by $T(y)$ re-obtains our generating function:

$$g(x,y) = \frac{T(y)}{1-B(y)x} = T(y)\sum_{k=1}^{\infty}\sum_{j=0}^{(k-1)m} d_j^{k-1} y^j x^k \tag{3.73}$$

$$= \sum_{i=1}^{m} t_i y^i \sum_{k=1}^{\infty}\sum_{j=0}^{(k-1)m} d_j^{k-1} y^j x^k \tag{3.74}$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{\infty}\sum_{j=0}^{(k-1)m} t_i d_j^{k-1} y^{i+j} x^k \tag{3.75}$$

$$= \sum_{k=1}^{m}\left[\sum_{i=1}^{\infty}\sum_{j=0}^{(k-1)m} t_i d_j^{k-1} y^{i+j}\right] x^k \tag{3.76}$$

This identity (visual proof omitted):

$$\sum_{i=1}^{N}\sum_{j=0}^{M} a_{i,j} y^{i+j} = \sum_{p=1}^{N+M}\left[\sum_{j=\max(1,p-M)}^{\min(p,N)} a_{j,p-j}\right] y^p \tag{3.77}$$

with $N \to m, M \to (k-1)m, a_{i,j} \to t_i d_j^{k-1}$ can be used to obtain:

$$g(x,y) = \sum_{k=1}^{\infty}\sum_{p=1}^{km}\left[\sum_{j=\max(1,p-(k-1)m)}^{\min(p,m)} t_j d_{p-j}^{k-1}\right] y^p x^k \tag{3.78}$$

$$= \sum_{L=1}^{\infty}\sum_{p=1}^{Lm}\left[\sum_{j=\max(1,h-mL+m)}^{\min(h,m)} t_j d_{h-j}^{L-1}\right] y^h x^L \tag{3.79}$$

$$= \sum_{L=1}^{\infty}\sum_{p=1}^{\infty}\left[\theta(Lm-h)\sum_{j=\max(1,h-mL+m)}^{\min(h,m)} t_j d_{h-j}^{L-1}\right] y^h x^L \tag{3.80}$$

$$\tag{3.81}$$

where $\theta(x)$ is the Heaviside function. Extracting out $P_{h,L}$, we get,

$$P_{h,L} = \theta(Lm-h)\sum_{j=\max(1,h+m-mL)}^{\min(h,m)} t_j d_{h-j}^{L-1} \tag{3.82}$$

And so we end with:

$$\boxed{\therefore P_{h,L} = \theta(Lm - h) \sum_{j=\max(1,h+m-mL)}^{\min(h,m)} P_{j,1} d_{h-j}^{L-1}} \tag{3.83}$$

This describes the probability that an opponent with $h$ health is killing on the $L$'th attack, given some boundary conditions (assuming there is no healing).

### 3.3.3 Applying Boundary Conditions

Those boundary conditions depend on the damage distribution. For an arbitrary distribution characterized by $c_i$, $i \in [1, m]$,

$$P_{j,1} = \sum_{i=j}^{m} c_i \tag{3.84}$$

which gives,

$$P_{h,L} = \theta(Lm - h) \sum_{j=\max(1,h+m-mL)}^{\min(h,m)} \sum_{i=j}^{m} c_i d_{h-j}^{L-1}. \tag{3.85}$$

The above work has determined the probability of defeating an opponent after $L$ attacks. However, more applications open up if the probability of bringing an opponent from $h$ health to $f$ health after $L$ attacks is known. This is given as

$$P_{h,L}^{f} \tag{3.86}$$

for which the solution has not yet been determined. To carry this out, the derivation in this chapter needs to be modified.

## 3.4 Fight Outcomes

To determine the outcome of a fight, we have to consider how the attacker and defender's attacks interplay. Let's define the first tick in a fight as $t = 0$. Then, the attacker may have some initial delay, $\Delta_1$, while attacking every $w_1$ ticks. Then, the attacker makes an attack on the following ticks:

$$T_1 \equiv \{w_1 i + \Delta_1 \,|\, 0 \leq i \leq T_1^{\max}\}. \tag{3.87}$$

where $T_1^{\max}$ is introduced as a cut-off where $P^1(t) < \epsilon \, \forall t \geq T_1^{\max}$, to avoid summing to infinity. Similarly, for the defender:

$$T_2 \equiv \{w_2 i + \Delta_2 \,|\, 0 \leq i \leq T_2^{\max}\}. \tag{3.88}$$

Then the probability of winning can be given by:

$$P_{\text{win}} = \sum_{t \in T_1} \sum_{\substack{\tau \in T_2 \\ \tau > t}} P_\tau^1 P_t^2 \tag{3.89}$$

$$\tag{3.90}$$

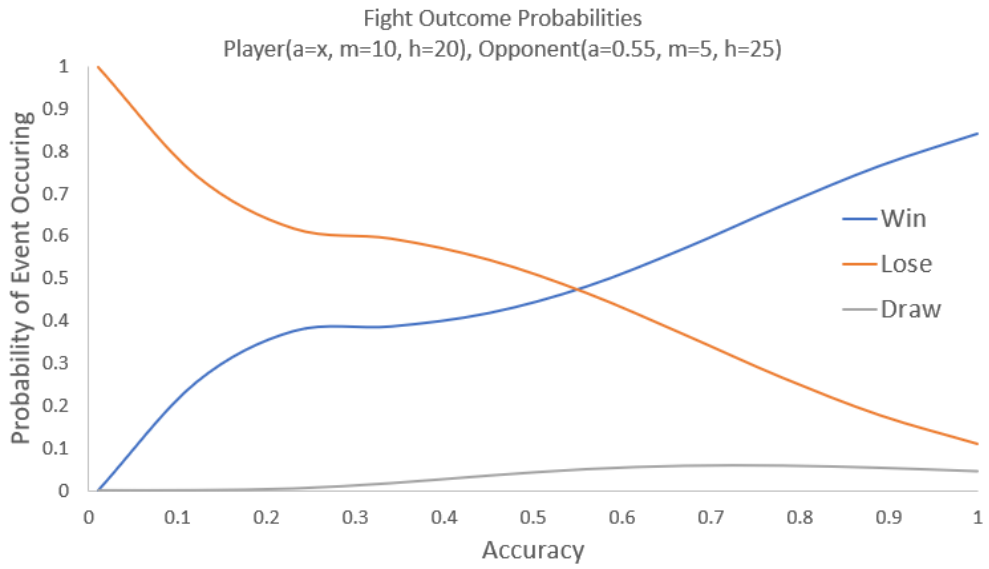



**Figure 3.2:** The probability of the player winning, losing, and drawing during their fight with an opponent. Both are using the standard weapon distribution. The player's accuracy is varied which shows that the player needs at least 55% accuracy to have a higher chance of winning than their opponent. With 0% accuracy the player is guaranteed to lose, but with 100% accuracy there is still a chance the opponent wins.

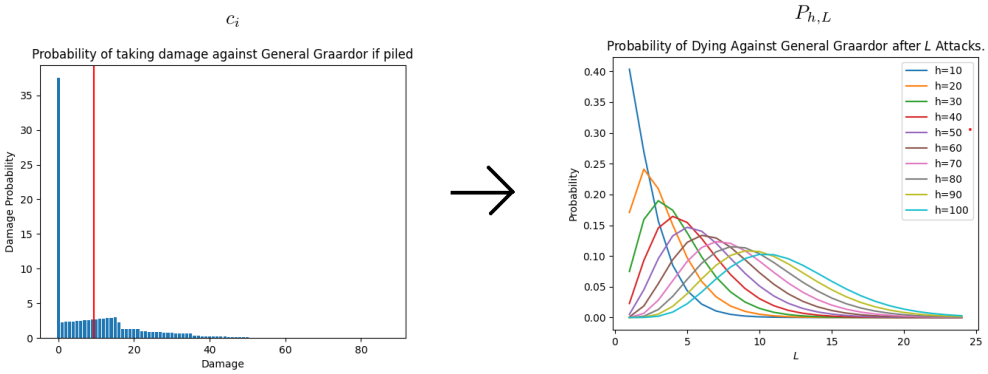



**Figure 3.3:** The damage distribution of General Graardor (plus minions) assuming protect from melee (the average is in red). Under the mechanics of Eq. (3.83), the probability of dying after L attacks is shown for different starting hitpoints. Near 0hp the player is expected to loss immediately. After about 50hp, it becomes impossible to get killed in one shot, and we get a Gaussian survival probability.

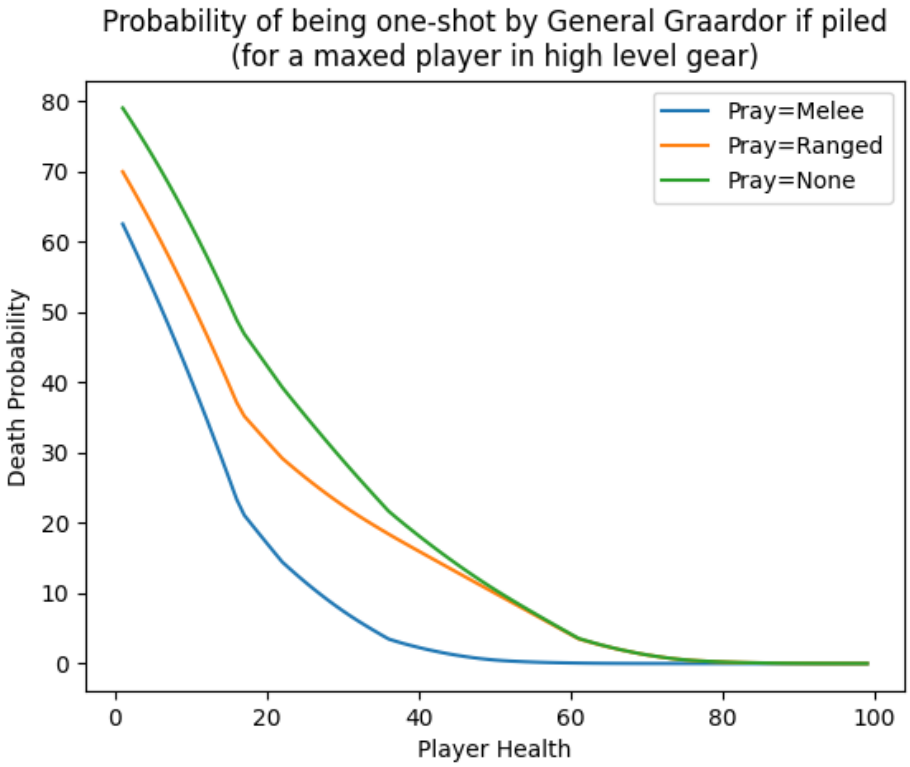



**Figure 3.4:** The best protection is melee regardless of player health. Over 80hp, it doesn't matter what you pray - you cannot be one shot. Just below that, if you are not praying melee you might get one-shot and range pray doesn't help you. If you are below 50hp, then you might get one-shot by ranged and so that prayer starts to matter again.

# Chapter 4

# Optimizing Player Equipment

In Section 1.1.2 we discussed the basic formulation of equipment bonuses. In this chapter, we will be interested in optimizing the player's choice of equipment to maximize some metric. Some examples of these metrics would be combat experience per hour, number of kills per hour, probability of winning, and so. Since the goal of the player may vary heavily, we aim to produce a general framework that can maximize arbitrary objective functions.

There are a large number of items that a player can equip in each slot, typically ranging from 10-100 considerations. Exhaustively considering each possible combination of equipment would result in roughly $10^{10}$ to $100^{10}$ possible sets. It is obvious then, that brute force would not work. We aim to reduce the number of possible equipments *loadouts*, $\mathcal{L}$ that we must consider. A loadout is simply the set of equipment that a fighter is wearing:

$$\mathcal{L} = \{I_{\text{slot}} \,|\, \text{slot} \in \text{slots}\}. \tag{4.1}$$

To do this, we will reduce redundant equipment choices for each slot such that we end with a set of *possibly* optimal loadouts. The only way to determine which of those is the actual optimal solution, $\mathcal{L}^*$ is to numerically evaluate the value of the objective, $f$, for each possibly optimal loadout. This defines our optimization problem as:

$$\mathcal{L}^* \leftarrow \underset{\mathcal{L}\in\{\mathcal{L}\}}{\operatorname{argmin}} f(\mathcal{L}). \tag{4.2}$$

So we will begin by reducing the size of the set $\{\mathcal{L}\}$.

## 4.1 The Projection Vector

Different optimization problems require different information about the player's equipment. For example, when straightforwardly optimizing pure damage output (typically measured in kills per hour), the defensive bonuses of the player are irrelevant. By contrast, trying to maximize the probability of killing an opponent would require consideration of those defensive bonuses. Furthermore, a fighter only attacks with one attack style at a time. So clearly, not all bonuses are required for every optimization. For this reason, we introduce a *projection* vector, $\vec{p}$, that will *select* the bonuses that matter. Recall that our bonuses are divided into two groups: constant bonuses, and special bonuses. In Section 1.1.2, we discussed how we treat special bonuses differently. So we will ignore them for now, and focus on the constant bonuses, $\vec{\mathcal{I}}_c^{\text{slot}}$. To select out the desired bonuses, we make $\vec{p}$ the same length as $\vec{\mathcal{I}}_c^{\text{slot}}$ so that each element in $\vec{p}$

corresponds to an equipment bonus. We can do so-called one-hot encoding, where we set elements to either 0 or 1 depending on whether the corresponding equipment bonus should be considered.

An element-wise multiplication (also known as a Hadamard product) of these two vectors, $\vec{\mathcal{I}}_c^{\text{slot}} \odot \vec{p}$ contains the bonuses of the item that we want, and 0's for bonuses we don't want to consider. To compare different items we need to define a comparison operator, $\hat{C}(\cdot,\cdot)$ such that:

$$\hat{C}(\vec{\mathcal{I}}_{c,1}^{\text{slot}}, \vec{\mathcal{I}}_{c,2}^{\text{slot}}; \vec{p}) = \begin{cases} 1 \text{ if } \exists\, i \in [1,n] \,|\, (\vec{\mathcal{I}}_{c,1}^{\text{slot}} \odot \vec{p})_i > (\vec{\mathcal{I}}_{c,2}^{\text{slot}} \odot \vec{p})_i \\ 0 \text{ otherwise,} \end{cases} \tag{4.3}$$

where $n$ is the number of bonuses for an item can have, and $(\cdot)_i$ represents the $i$'th bonus. By simply comparing the individual item bonuses, this term indicates whether some item, $\vec{\mathcal{I}}_{c,1}^{\text{slot}}$, provides some advantage over another, $\vec{\mathcal{I}}_{c,2}^{\text{slot}}$. If a given item doesn't provide some advantage over any other item then it doesn't need to be included in the set of equipment to consider. Note that $\hat{C}(\vec{\mathcal{I}}_{c,1}^{\text{slot}}, \vec{\mathcal{I}}_{c,1}^{\text{slot}}; \vec{p}) = 0$.

## 4.2 Set Reduction

With this, we can reduce the set of possible items in given slot, $\{\mathcal{I}_j^{\text{slot}}\}_{j=1}^{N_{\text{slot}}}$, where $N_{\text{slot}}$ is the number of possible items in that slot. We can define the following matrix,

$$\boldsymbol{A}^{\vec{p}} = A_{ij}^{\vec{p}} = \hat{C}(\vec{\mathcal{I}}_{c,i}^{\text{slot}}, \vec{\mathcal{I}}_{c,j}^{\text{slot}}; \vec{p}), \tag{4.4}$$

that is also one-hot encoded with 1's representing that the item in column $i$ provides some advantage over the item in row $j$, for our optimization problem.

These individual item comparisons aren't too important, instead we care if an item may provide some advantage over *any* other in the set. So, an item shouldn't be considered if the comparison yields 0 when compared against all other items:

$$\sum_{i=1}^{N} \hat{C}(\vec{\mathcal{I}}_{c,j}^{\text{slot}}, \vec{\mathcal{I}}_{c,i}^{\text{slot}}; \vec{p}) = 0 \tag{4.5}$$

$$\implies \sum_{i=1}^{N} A_{ij}^{\vec{p}} = 0 \tag{4.6}$$

$$\implies \mathbf{1} \cdot \boldsymbol{A}_j^{\vec{p}} = 0 \tag{4.7}$$

where $\mathbf{1}$ is a $N_{\text{slot}}$-dimensional vector containing ones, and $\boldsymbol{A}_j^{\vec{p}}$ is the $j$'th column in $\boldsymbol{A}^{\vec{p}}$. Then the reduced set of items for a given slot is then:

$$\{\mathcal{I}_j^{\text{slot}}; \vec{p}\}_{j=1}^{\bar{N}_{\text{slot}}} = \{\mathcal{I}_j^{\text{slot}} \,|\, \mathbf{1} \cdot \boldsymbol{A}_j^{\vec{p}} \neq 0,\ \ 1 \leq j \leq N_{\text{slot}}\}, \tag{4.8}$$

where $\bar{N}_{\text{slot}}$ is the number of items left in this reduced set. The possibly optimal loadout set can be constructed as a "$n$-ary" Cartesian product:

$$\{\bar{\mathcal{L}}; \vec{p}\} = \{(\mathcal{I}^{\text{head}}, ..., \mathcal{I}^{\text{ring}}) | \mathcal{I}^{\text{slot}} \in \{\mathcal{I}_j^{\text{slot}}; \vec{p}\}_{j=1}^{\bar{N}_{\text{slot}}}\ \forall\, \text{slot} \in \{\text{slots}\}\} \tag{4.9}$$

As mentioned earlier, the choice of $\vec{p}$ depends on the problem at hand. However, only one attack style is generally considered and so often $\vec{p}$ can be chosen to only consider one attack style.[1] A natural formulation would be to iterate over each attack style. We

[1] If the fighter switches weapons, or has multiple attacks, considering more than one attack style would then be required.

can, for example, use $\vec{p}_{\text{stab}}$ to denote a projection vector which has stab as the only non-zero attack bonus. Then, the total set of considerations is the union of the specific attack style sets:

$$\{\bar{\mathcal{L}}; \vec{p}\} = \bigcup_{a \in \{\text{attack styles}\}} \{\bar{\mathcal{L}}; \vec{p_a}\}. \tag{4.10}$$

This makes our optimization:

$$\mathcal{L}^* \leftarrow \underset{\mathcal{L} \in \{\bar{\mathcal{L}}; \vec{p}\}}{\text{argmin}}\, f(L), \tag{4.11}$$

which has reduced a search space containing billions of possibilities to one containing (empirically) less than 1,000.

## 4.3 Gear Optimization Implementation & GUI

The ability to determine the optimal gear load-out for a given player-opponent-environment combination is possible through the previous sections. Set reduction makes brute force feasible albeit with special equipment complicating the implementation. With a reduced set, one can simply evaluate the theoretical calculations for the desired metric (eg: xp/h, kills per hour, $P_{\text{win}}$) for every "possibly-optimal" gear load-out. The solution is simply the load-out with the maximum value. The full interface with all considerations is shown in Fig. 4.1.

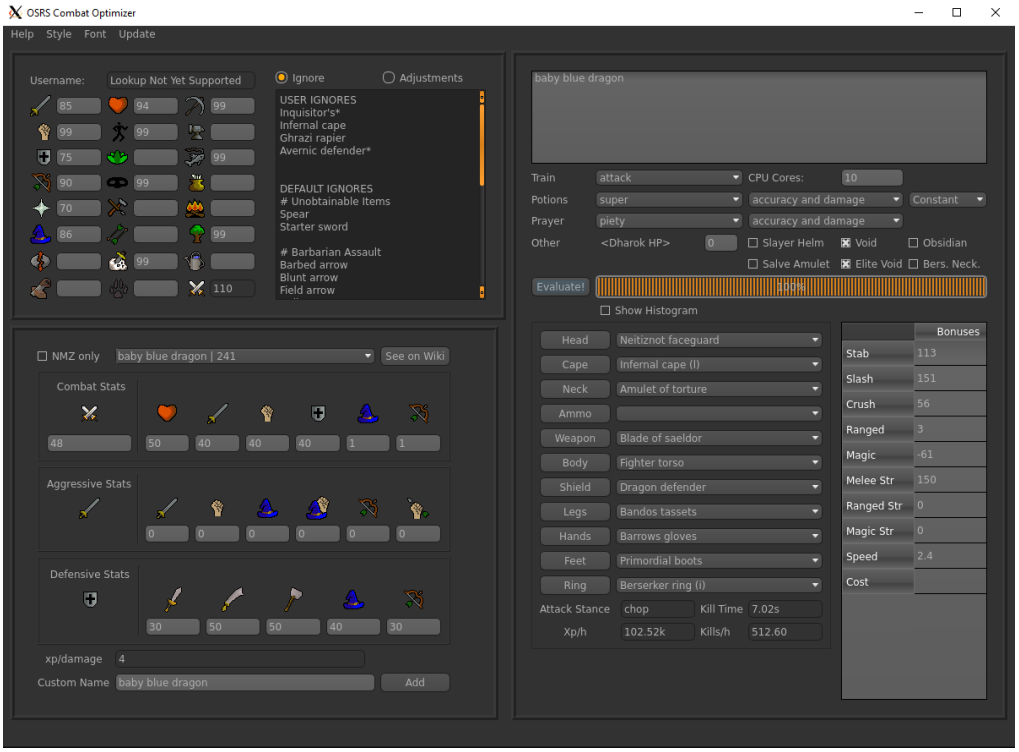



**Figure 4.1:** The optimal equipment gear optimizer tool written in PyQt5. The user inputs their skill levels, which not only determine combat skills but also auxiliary skill requirements (eg: combat axes require woodcutting levels). The player can iteratively ignore items as the optimal solution suggests items the player cannot obtain. The panel for level requirements and stat adjustments is hidden. On the bottom, the player also selects their opponent and respective stats; they can also add multiple opponents to be fought in sequence. Environmental information (eg: potions, slayer task), along with which skill to train can be selected in the top right. CPU cores reflects the multi-threaded implementation which brute forces over the reduced equipment sets. Finally, the optimal item is identifier for each equipment slot, which is stated alongside the bonuses and metrics. Special equipment effects are only partially treated, which can impact the reliability, however it should be considered accurate for all normal combat.

# Chapter 5

# Optimizing Training Order

## 5.1 Dijkstra's algorithm

Any time the player is training combat, they have to also make a choice about what skill to train. The skill choices are attack, strength, ranged, magic, and defence. Hitpoints is automatically trained regardless of style, and prayer is not generally trained during combat (although it is totally possible). Furthermore, attack options that offer "shared xp" significantly complicate the model since it forces the problem space into xp (which exceeds 10 million) whereas without that only levels are needed (less than 100), as a result it will also be ignored.

Let's assume a player in a standard loadout (using a dscim within the Nightmare Zone) starts at levels (attack=60, strength=60) and their goal is obtaining (99, 99). The player's first choice is between training to (61, 60) or (60, 61). Repeatedly asking that question all the way to (99, 99) gives the tree-like graph depicted in Fig. 5.1.

Since every level exponentially increases the amount of time it takes to train, training one skill results in diminishing returns. Shortest Path algorithms are specially designed to solve this problem, with Dijkstra's algorithm being able to compute exact solutions. The interplay between incremental boosts and longer training times produces optimal solutions like the one shown in Fig. 5.2. This is further illustrated in Fig. 5.3.

The human player typically optimizes these choices heuristically, and so it is interesting to compare the optimal graph solution with other options. The table below compares different strategies, most notably the optimal solution is only a 0.5% improvement over simply training strength to 99 then training attack. By contrast, maxing attack first increases training time by 20%, since the max hit cannot increase. Most players intuitively understand that strength is much more important than attack.

| **Strategy** | **Total Time (h)** | **Additional Time (h)** | **% Improvement** |
|---|---|---|---|
| Inverted | 356.6 | 30.1 | 8.4 |
| $(1, 99) \rightarrow (99, 99)$ | 328.3 | 1.8 | 0.5 |
| $(99, 1) \rightarrow (99, 99)$ | 393.4 | 66.9 | 17 |

**Table 5.1:** Comparison of different training orders

The main reason defence training has been omitted is that if defence is trained the player may unlock new armor. So the solution described in the Optimizing Player Equipment chapter is required. At each level, the optimal equipment is determined by set reduction and the experience rates are then calculated. In Fig. 5.4 the effect of equipment upgrades is shown.

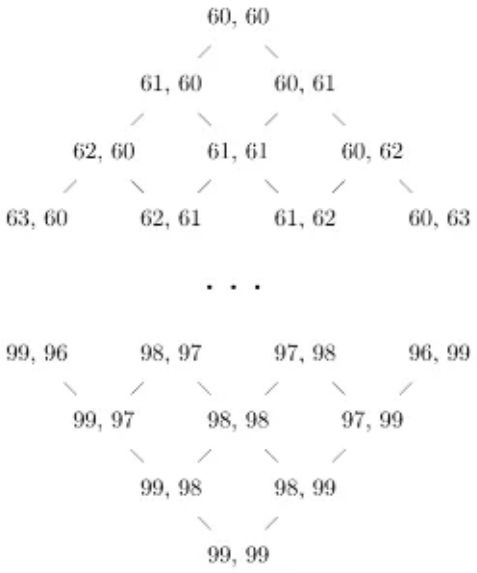



**Figure 5.1:** A depiction of the choices the player has to make as they train their attack and strength skills from (60, 60) to (99, 99). Starting at the top, each tier represents a choice to train either attack (right-down) or strength (left-down).

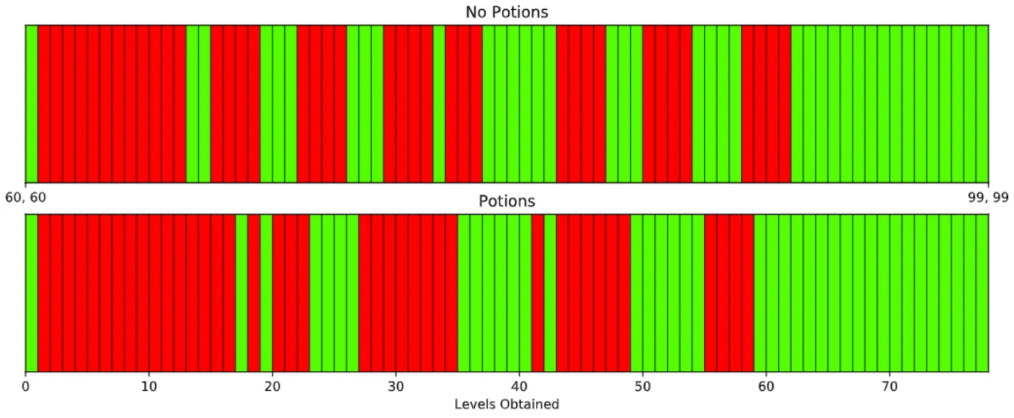



**Figure 5.2:** The sequence of training strength (red) and attack (green) for a player both using potions and not. It is clear that starting with strength training is ideal, with the last levels all being attack.

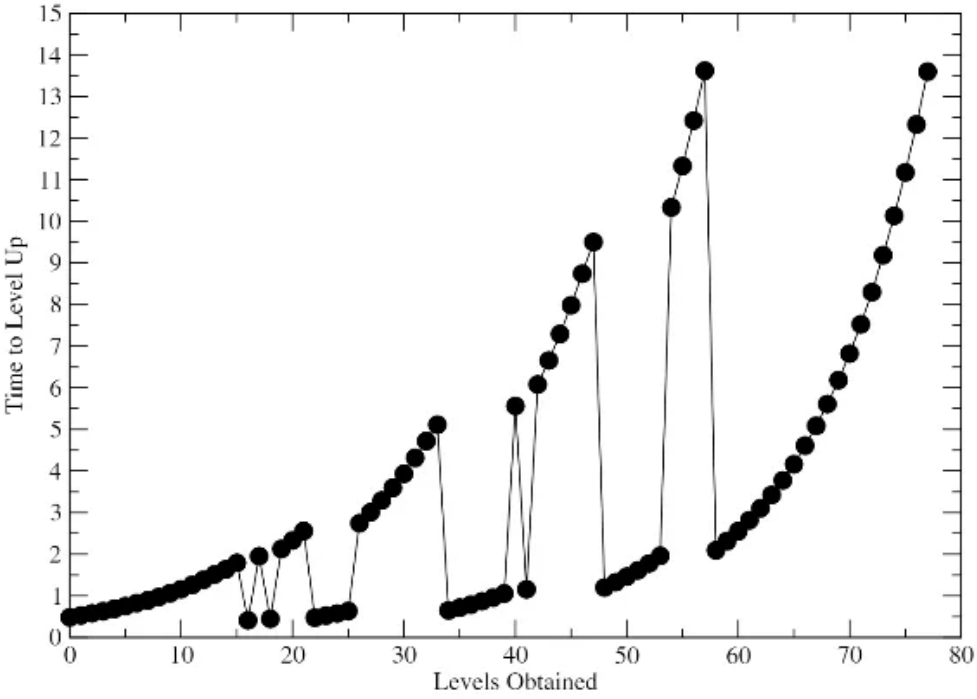



**Figure 5.3:** The time taken to level up plotted against the number of levels the player has trained. Two bands clearly emerge which are associated with strength (top) and attack (bottom). The player initially trains strength (driving up the time to level) until about 15 levels are gained, at which point the minor improvement from attack out-competes the large amount of time to get the next strength level.

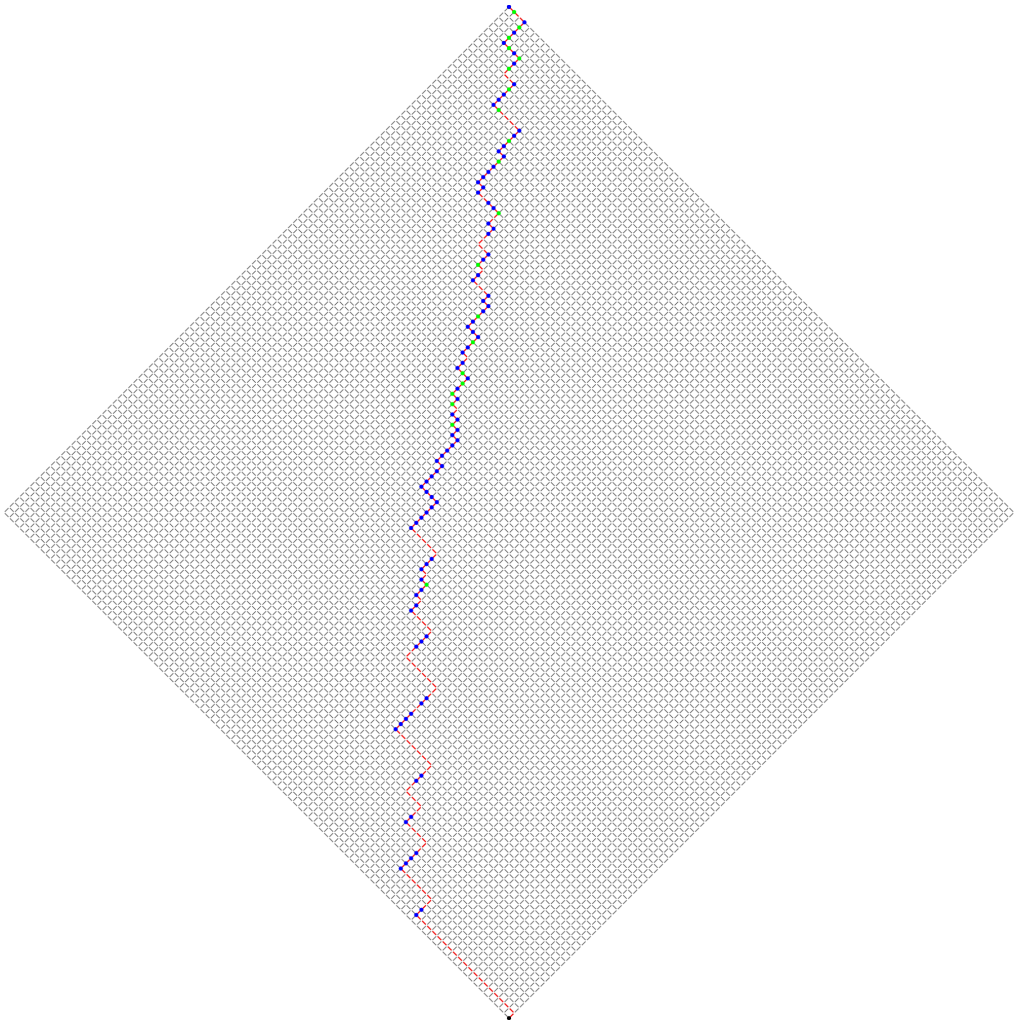


**Figure 5.4:** The optimal training order solution from level (1, 1) to (99, 99), while allowing for optimal equipment to be determined at each level. Green nodes indicate that a previously unseen/unused piece of equipment is now being used by the player. Blue nodes indicate that the player changed their gear to something they previously had. The consistent leftward tilt of the training order suggests that keeping your strength $\sim$10% over your attack level is near-optimal.

# Appendices

# Appendix A

# Justifying the Recursive Model Approximation

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

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

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

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

After some algebra we find that,

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

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

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

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

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

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

The error, $\epsilon$ in the next iteration of Newtons method is given as,

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

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

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

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

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

which simplifies the error equation to,

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

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,

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

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

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 $\epsilon$ is small is violated and the Taylor series formulation no longer holds.

# Appendix B

# Power Reduction in the Piecewise Recursive Model

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

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

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

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

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

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

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

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

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

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

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

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

To finish, let's focus on,

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

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

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

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.

## B.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 $h_0^2$,

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

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:

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

Using the same logic, we can inductively deduce,

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

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

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

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,

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

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

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

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

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,

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

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.

## B.2 Simplifying

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

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

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

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

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

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

and so finally we arrive at,

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

# Part III

# Woodcutting

# Chapter 1

# Traditional Woodcutting

Woodcutting focuses on collecting logs by chopping them with a given axe. The main modifiers to experience are the type of axe and the type of tree. In Fig. 1.1 you can see an experience landscape that shows the experience rate per hour based on the tree (by level, eg: 15 is oak) and wooductting level. For this section, a small dataset was collected to measure various tree chop rates at a few levels. The rates are interpolated to work for any level.

Each tree requires data to be collected at two points and a linear interpolation can be used to determine the rate at any level. The sample was collected on two accounts at level 45 and 77 woodcutting. Since trees above 45 cannot be cut, they represented extrapolated values. Since the highest level that can analyzed by this data collection is 45, parts of the landscape aren't possible (eg: chopping Oak at level 1).

For more details, you can watch the Optimizing Runescape Woodcutting Video. Furthermore, the wiki team has recently provided a complete and comprehensive set of values in the Theoretical Woodcutting XP Rates page which now supersedes this work.

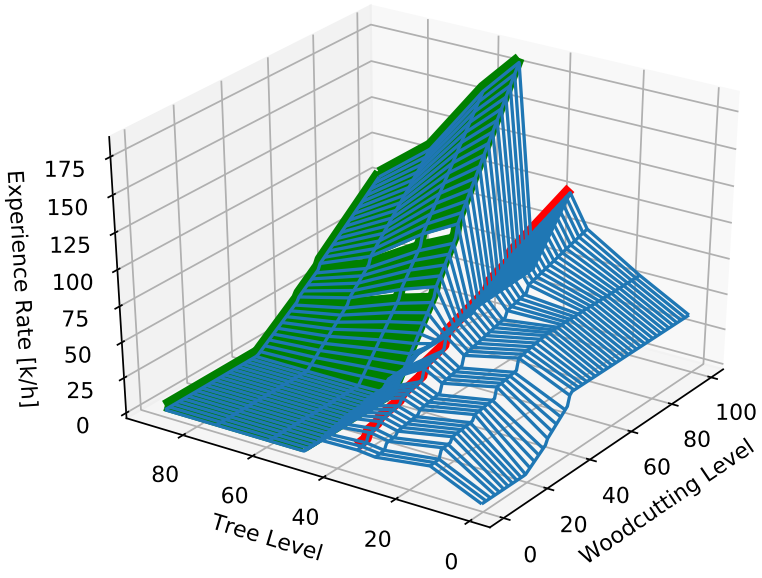



**Figure 1.1:** The experience levels for *Teak Trees* in red are exactly known and not estimated by a sample. The portion in green represents extrapolations out side of the dataset (due to the 45 woodcutting limitation). As a result the extrapolations near $L = 50$ are misleading.

# Part IV

# Firemaking

# Chapter 1

# Wintertodt

Wintertodt is a skilling boss that is primarily fought (in a manner similar to a minigame moreso than combat) for firemaking experience and resources, either solo or as part of a (large) group. Experience is gained from actions during the fight, with a bonus awarded for players that have contributed enough. The bonus provides experience and supply crates which contain valuable resources.

The objective of the fight is to defeat the Wintertodt. This boss takes damage over time while any of the four braziers in the room are lit. Actions that the player performs are centered on maintaining and feeding these braziers. Feeding the braziers provides the majority of the firemaking experience. They are occasionally extinguished (by the cold wind), and broken (by the flame's heat). Additionally the Pyromancer associated with each brazier may need healing before a player can relight an extinguished brazier. In a group fight, the maintenance actions can be performed by others. To feed the braziers, players will chop Bruma roots and optionally fletch them into kindling. This provides more experience and points when burned, but requires more time to process. Several of these actions are shown in Fig. 1.1.

Most of these actions award the player with points and provide experience in some skill. The experience scales linearly with the respective skill level, with the proportionality constant, $M$, determined by the action. If the player reaches a threshold of 500 points, they gain bonus experience and a supply crate at the end of a fight. A summary of these actions can be found below in Table. 1.1.

**Table 1.1:** Actions and the associated skills, XP multipliers, and points awarded.

| Action | Skill | XP Multiplier ($M$) | Points |
|---|---|---|---|
| Lighting braziers |  | 6.0x | 25 |
| Chopping Root |  | 0.3x | - |
| Burning Root |  | 3.0x | 10 |
| Burning Kindling |  | 3.8x | 25 |
| Repairing Brazier |  | 4.0x | 25 |
| Healing Pyromancer | – | 0.1x | 30 |
| Creating Potion |  | - | - |
| Picking Bruma Herb |  | - | - |
| Win with 500+ Points |  | 100x | - |

There are several problems that would be interesting to solve. The mechanics to model and important problems vary depending on whether the boss is fought in a large

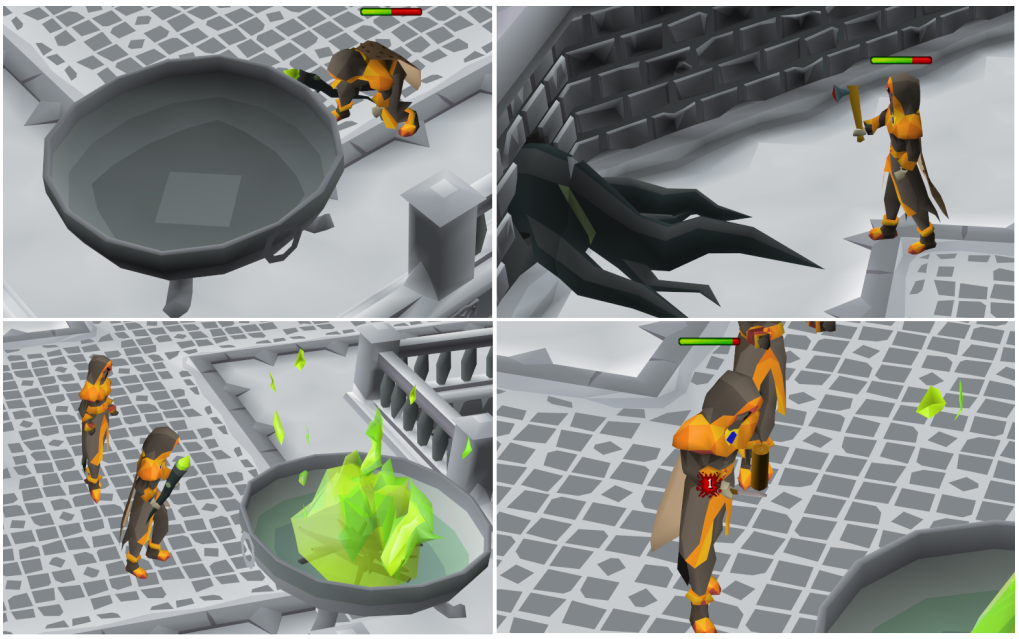


**Figure 1.1:** A player performing several actions during a Wintertodt fight. From top-left to bottom-right the player is: lighting a brazier, chopping tree roots, burning roots/kindling, and fletching roots into kindling. Actions that are not shown including creating potions, healing the Pyromancer, and fixing/repairing the brazier.

group or solo. For solo fights, the optimal strategy (i.e. which actions to perform when) could be investigated. For large groups, we can ignore the emergent behavior of the players by making certain assumptions. For example, the root chopping rate can be ignored if we assume that the player always reaches a target number of points. We will tackle the latter since group fights are (empirically) more common. The quantities of interest are the number of kills required to obtain a certain level, the number of crates/value/resources obtained after a certain number of kills, and the time required for a kill. We will only be concerned with firemaking experience for now.

## 1.1 Large groups

We will be attempting to determine the number of games required to get to a desired firemaking level. Since the game play is relatively involved, we will apply a constraint to the player's actions to make it more manageable. We will limit the actions to: burning, fletching, and cutting (so ignoring: healing, relighting, and repairing). Although this isn't exactly how players would play, relighting is the only action that provides firemaking experience, and all ignored actions can be performed by other players. Furthermore, all those actions are infrequent in comparison to burning and so these shouldn't be too significant.

Let $M_{\text{root}} = 3$, $M_{\text{kindling}} = 3.8$, and $M_{\text{bonus}} = 100$ be the XP multiplier for burning a root, burning a kindling, and obtaining 500+ points, respectively. We also have that $P_{\text{root}} = 10$ points are given for burning a root, and $P_{\text{kindling}} = 25$ are given for burning kindling. Since the player may level up during a fight (and thereby change their experience rate), we consider this problem in two stages: within a fight and across fights. Finally, we will need to make use of the level equation, $\mathcal{L}$ from Chapter 1, that tells us what level a skill is, given its experience.

### 1.1.1 Within a Fight

Since a player may level up over the course of a fight, we need to consider the experience gained on a per action basis. We will label the total player's firemaking experience after $a$ actions are performed as $E_a$. The firemaking actions during a fight come from burning either roots or kindling. To be general, let's define an indicator function $\delta^n_{\text{action}}$ that is 1 if the $n$'th action performed by the player matches the subscripted value, and is zero otherwise. For example, $\delta^n_{\text{root}} = 1$ if the $n$'th action was burning a root. Then, the value of the XP multiplier for the $n$'th firemaking action can be defined as:

$$c_n \equiv \delta^n_{\text{root}} M_{\text{root}} + \delta^n_{\text{kindling}} M_{\text{kindling}}. \tag{1.1}$$

Starting with $E_0$ experience, the experience after $a$ actions during a kill is governed by:

$$E_{n+1} = E_n + c_n \mathcal{L}(E_n) \tag{1.2}$$

This equation *may* have a solution except that $\mathcal{L}$ has no analytic form, and so this must be evaluated iteratively. Practically, we can introduce player policies or strategies, that model how a player acts, by labeling the coefficients as $c^{\text{policy}}_n$. A policy essentially generates the sequence of player actions, for example: an *Only Burning Roots* (OBR) policy could be defined as:

$$c^{\text{OBR}}_n \equiv \begin{cases} M_{\text{root}} & \text{if } n \leq T/P_{\text{root}} \\ 0 & \text{otherwise.} \end{cases} \tag{1.3}$$

Regarding notation, the labeling $E^{\text{policy}}_n$ may now be used to denote the experience gained according to some policy. More advanced policies can also be defined. A reasonable strategy to maximize experience while reaching the point threshold would be to burn kindling until the player obtains 500 points followed by burning logs until they reach some target number of points, $T$. We can call this the *Kindle Till Bonus* (KTB) policy, which could be defined as:

$$c^{\text{KTB}}_n \equiv \begin{cases} M_{\text{kindling}} & \text{if } n \leq 500/P_{\text{kindling}} \\ M_{\text{root}} & \text{if } n \leq \frac{T}{P_{\text{root}}} - \frac{P_{\text{kindling}}}{P_{\text{root}}} \lceil 500/P_{\text{kindling}} \rceil \\ 0 & \text{otherwise.} \end{cases} \tag{1.4}$$

The first bound is the number of kindling needed to reach 500 points. The second bound is the number of points remaining, divided by the points per root which gives the number of roots left.

### 1.1.2 Across Fights

We now have the ability to calculate the firemaking experience gained during a fight. We can take this further to calculate the firemaking experience gained across fights. The experience that a player has after $k$ kills can be denoted as $\mathcal{E}^k$. There are two contributions for this: one is the experience that a player gains during a fight, the other is the bonus experience gained after a kill. And so,

$$\mathcal{E}^{k+1} = \mathcal{E}^k + E^{\text{policy}}_{N(T;\text{policy})} + \theta(T - 500) M_{\text{bonus}} \mathcal{L}(\mathcal{E}^k), \tag{1.5}$$

where $N(T;\text{policy})$ is the number of actions to perform given the target and policy, and $\theta$ is the Heaviside step function. This equation can, again, only be evaluated numerically.

### 1.1.3 Kill Count, Time to Level, and Reward Value

Equation (1.5) can be used to calculate the experience gained after a certain amount of kills. Programmatically inverting this yields the kills needed to obtain a certain amount of experience. The kills needed to reach the highest firemaking level for different policies is compared in Fig. 1.2. The only-burning-roots, and only-burning-kindling policies act as upper and lower bounds for the number of kills required, regardless of which policy the player obeys. This is because the former gives the maximum experience per kill, while the latter gives the least. It's important to note that in practice, it is more difficult to obtain the desired target points ($T$) through roots alone. See Section 1.2 for possible future work.

To get some concrete numbers for the kills required, we can use the upper and lower bounds. Assuming 5 minute games (see Section 1.2 for a discussion), leveling from 50 to 99 takes:

1. For 500 Point Games:
    - 591 - 839 (roots only - kindling only) kills
    - 49.25 - 69.92 hours
2. For 750 Point Games:
    - 455 - 690 kills
    - 37.92 - 57.50 hours
3. For 1000 Point Games:
    - 369 - 586 kills
    - 30.75 - 48.83 hours

This author reached 99 after 546 kills. Several different playstyles could yield that: 500 points through roots only, 1000 points using kindling only, or 750 points using roughly equal amounts of each - which is anecdotally most similar.

In this time, the player will be accumulating supply crates as rewards. Each crate gives a certain number of *rolls* on a reward table. The number of rolls given from a crate depends on the points earned in the game, $p$, according to:

$$\text{Rolls} = \begin{cases} \min(1 + p/500, 28) & \text{if } p \geq 500 \\ 0 & \text{otherwise,} \end{cases} \tag{1.6}$$

where fractional values denote probabilities. Reference [11] provides a (manual) calculator that be used to calculate the value of each roll, and therefore crate. The total value earned after a given number of kills can be calculated as:

$$\text{Total Value} = \text{Kills} \times \text{Rolls} \times \text{Roll Value}. \tag{1.7}$$

Based on Fig. 1.3 (bearing in the small number of evaluation points), the value earned increases roughly linearly until about base 70s, after which there are diminishing returns. The value increases exactly linearly as a function of the points earned, since this is simply multiplicative.

Reference [12] provides a case study: with base 1's (except for 46-64 woodcutting) they obtained 5.5 million gp in 702 kills with 500-800 (estimated) point games, in $\sim$72 hours (which includes initial training and collecting food). It is difficult to discern agreement with the value earned here since prices have since increased/changed since 2016, and the variance in value is unknown. Despite this, the current estimates according

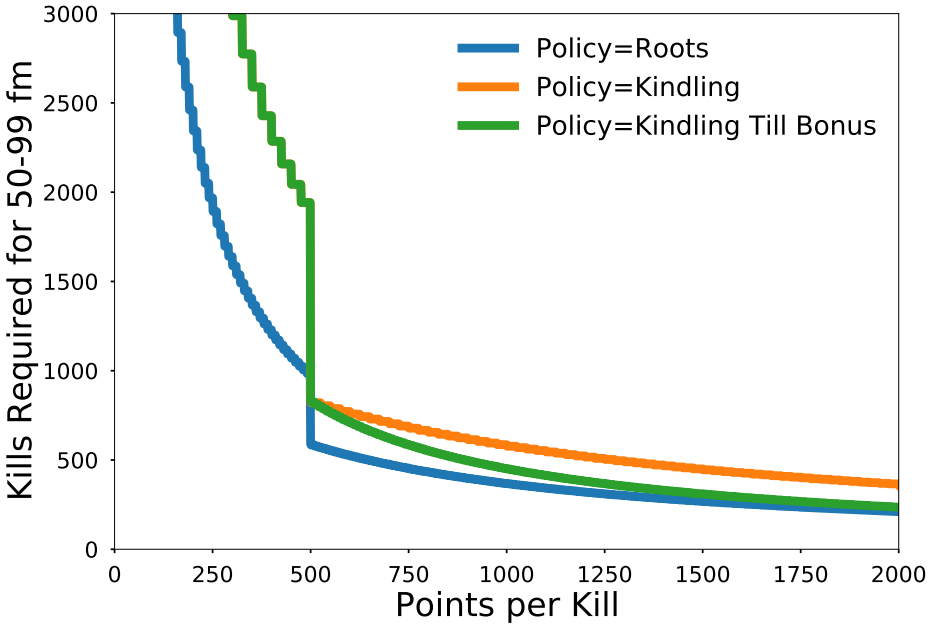



**Figure 1.2:** A comparison of the kills required to max the firemaking skill at wintertodt for different policies. Before reaching the point threshold, the kindling and kindling-till-bonus policies are equal. Failing to reach the threshold (every game) drastically increases the number of points required, implying that reaching this threshold is essential for players to maximize experience rates. On the other side of the scale, having a very large number of points gives diminishing returns. The roots and kindling policies act as lower and upper bounds for the number of kills, as discussed in the main text.

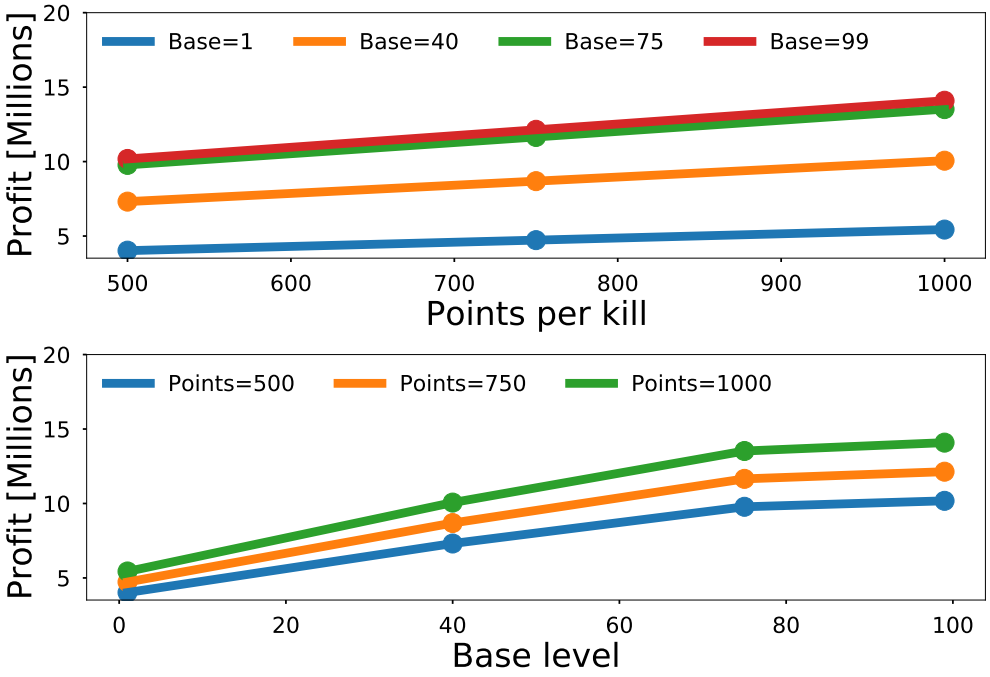



**Figure 1.3:** The profit a player should expect to gain from 50-99 firemaking, based on their base level, points earned per kill, and only burning roots. The upper panel shows the profit for different base levels. The lower panel shows the profit for different points earned.

to Fig. 1.3 are still around 5 million gp. 702 kills implies closer to 600-700 points mostly using kindling. 5 minute games implies about 58 hours and suggests about 10-15h of preparation, which is reasonable. All in all, these estimates seem to agree with this example.

## 1.2 Future Work

These are possible future avenues to consider:

- Include Action Rates: By including woodcutting, fletching, and burning rates, the action sequence that a player performs could be modeled and optimized (for points or experience). However, the woodcutting rate for Bruma roots is not given.
- Food Needed: During the fights the player takes damage that scales with their firemaking and hitpoints level. Some of the attacks are avoidable. Reference [13] provides a solid overview of the damage at Wintertodt. However, attack speeds/frequencies don't seem known. If this was known, the total amount of damage taken could be calculated. By accounting for natural regeneration, the amount of food required could be known in advance. This is important since many players train firemaking at Wintertodt right from the start (due to the rewards) and need to collect some preliminary food.
- Programmatic Crate Value: More advanced value calculations could be performed if determining crate value could be automated, although it doesn't seem like these equations are given. The wiki team can likely provide insight here. This would also allow the variance and confidence intervals to be determined.

- Real Game Duration Data: A stationary recording of wintertodt gameplay could be generated and analyzed to better approximate the distribution of game times. The mean and variance (as a function of player count, which could be approximated by world count) could be used for better estimates.
- Large Scale Statistics: The highscores could be analyzed to gain insight into player behaviour. There would have to be some selection criteria to assume a player hasn't gain much experience apart from at Wintertodt. There are standard skills expected for accounts that rush this boss that could be used (although it might be a biased sample). The kill count and total experience can be used to determine the average points per game, and average time spent.

# Part V

# Mining

# Chapter 1

# Motherload Mine

Mining focuses on collecting ores by extracting them from rocks with a given pickaxe. The main modifiers to experience are the type of pickaxe and the type of ore. Mining success rates are actually much harder to find than woodcutting for example. As a result calculating experience rates is much more difficult. Instead, in this section we look at the Motherload Mine which is a very common mining technique.

This activity enhances mining by providing a chance to obtain *Gold Nuggets* which can be used to purchase rewards like the *Prospector Outfit* that increases experience per ore mined. Each piece costs a certain amount, and each piece provides an experience boost, with more costly pieces providing more bonuses. So the natural question here is what order should the four pieces be purchased. With only four pieces a brute force approach is sufficient here as Fig. 1.1 reveals the optimal order to be

$$\text{Jacket} \to \text{Legs} \to \text{Hat} \to \text{Boots}, \tag{1.1}$$

which is exactly the order of most expensive to least expensive items. The worst ordering is

$$\text{Boots} \to \text{Hat} \to \text{Legs} \to \text{Jacket}, \tag{1.2}$$

which is the reverse of the optimal solution. Another way to visualize this is shown in Fig. 1.2.

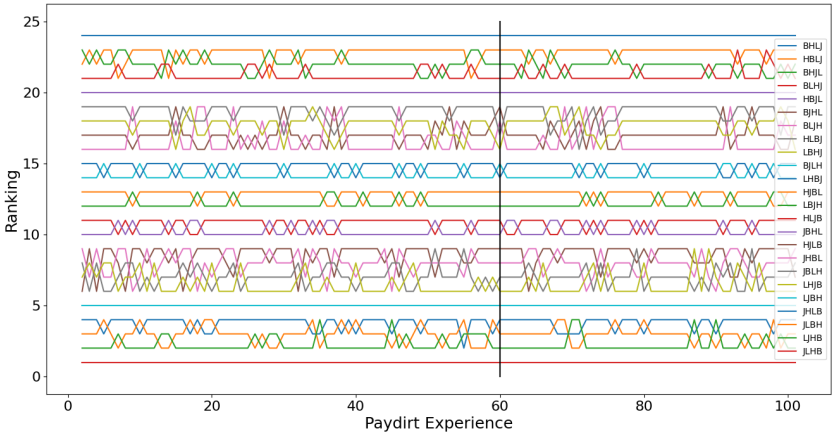



**Figure 1.1:** A ranking of all possible orderings to obtain the full outfit. Notice that the x-axis here is the experience from the paydirt although this is fixed in game at 60, varying this parameter reveals that the order is very stable (rankings only change by 1 or 2). The order "JLHB" is the optimal, which "BHLJ" is the worst as it is the reverse order.

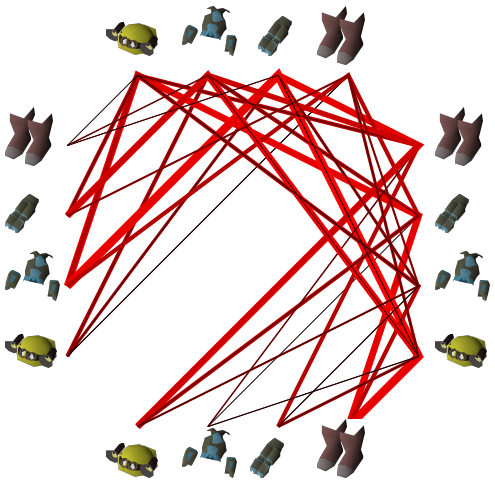


**Figure 1.2:** Starting on the left side, each line represents a possible ordering of obtaining the full set. The thickness of each line represents its rank. Clearly starting with the jacket (3rd row) is optimal, whereas starting with the boots (1st row) is worst. Said another way (looking at the bottom row), getting the jacket last is the worst and getting the boots last is the best.

# Part VI

# Quests

# Chapter 1

# Graph/Network Analysis

Quests are a massive part of playing the game with tons of content being unlocked by them. Moreover, their large experience rewards drive efficient players to empirically determine the optimal order of completing quests to obtain any particular goal (most notably the Quest Point Cape).

The difficulty in optimization arises from the complex dependencies between the quests. To handle this, we can treat each quest as an abstract node. A node has two standard prerequisites: the skill level requirements and the quest requirements. The skill requirements are a property of the node itself, but the quest requirements form edges between these nodes. Since quests reward experience unlocks skill levels, the graph becomes a dynamic, directed acyclic graph (DAG).

Other considerations like item requirements and combat (including bosses) further complicate this process and are not yet considered. Since item requirements can mostly be seen as level requirements, combat becomes the bigger focus. A player could define various constraints for all required combat (eg: $P_{\text{win}} > 0.01$), but this would require modeling equipment progression etc. Although, it would be a major achievement to include all these effects for the purpose of solving the quest problem, for this initial work quests will only depend on skills and other quests.

Quests can only depend on another quest in one direction and as a result they are naturally assigned tiers. For example, tier 0 quests have no quest dependencies, while tier 1 can only depend on Tier 0 quests (or no quests). Each tier depends exclusively on quests in lower tiers. This organizational hierarchy can be seen in Fig. 1.1.

The Runescape Universe is an interactive tool to explore the quest tree in both OSRS and RS3, created by this project to provide users with an interactive way to explore the dynamic, complex, and intertwined quest lines as shown in Fig. 1.2. For example, the iconic quest Monkey Madness I is shown in Fig. 1.3 to connect to a central network. In fact, all quests are connected to the large central island, except for isolated nodes (typically only 1 or 2 nodes). Only trivial quest islands existed until the Varlamore continent was released. This island is likely temporary since the same thing happened when the Kourend continent was released, but was later connected to the main island through DSII.

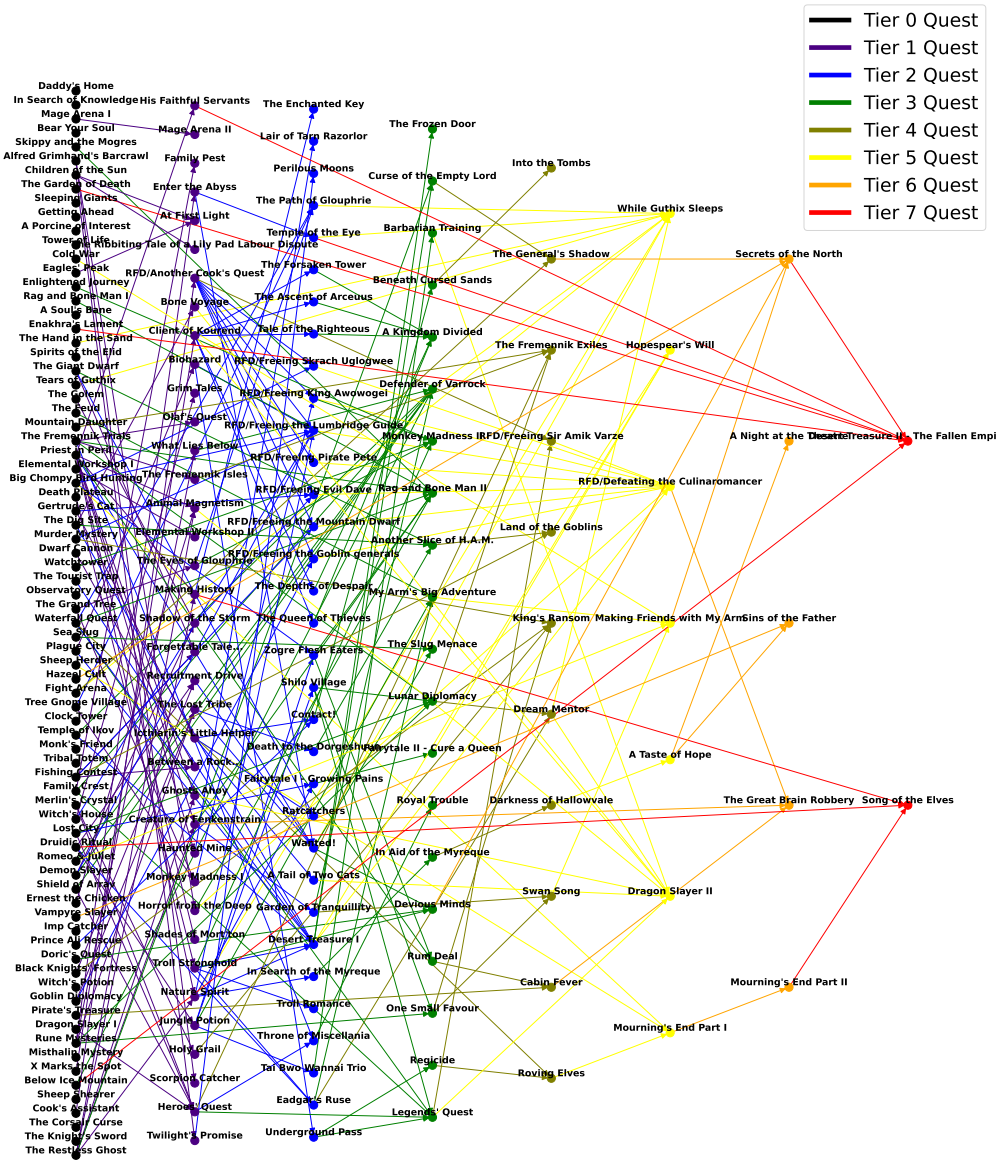



**Figure 1.1:** The OSRS quest tree, which organizes all quests into tiers, with higher tier quests depending exclusively on lower tier quests. With the release of *Song of the Elves* in 2019, OSRS gained a 7th quest tier. By contrast RS3 has 11 tiers (Aftermath and Sliske's Endgame). Edges are colored by the quest tier they unlock. Increasing tiers diminish in size, although this is not required and stems from game design choices.

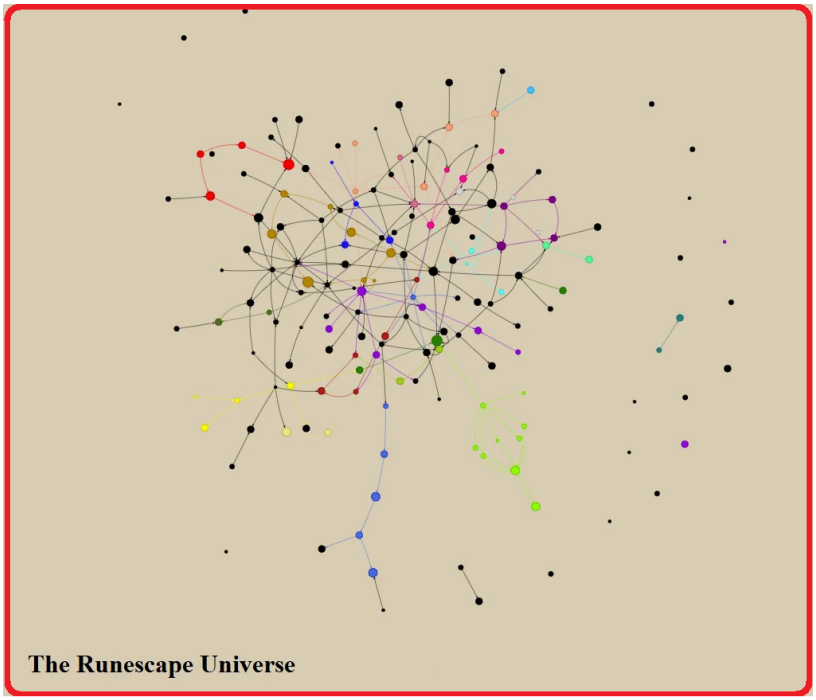



**Figure 1.2:** The Runescape Universe renders all quests and mini-quests as nodes in a graph. The quest prerequisites form edges between the nodes. Isolated islands form quest chains consisting of only one or two nodes. The colors here represent the quest series and the size represents the official length of the quest. It is within this picture that nearly all the adventures of our player character are experienced.

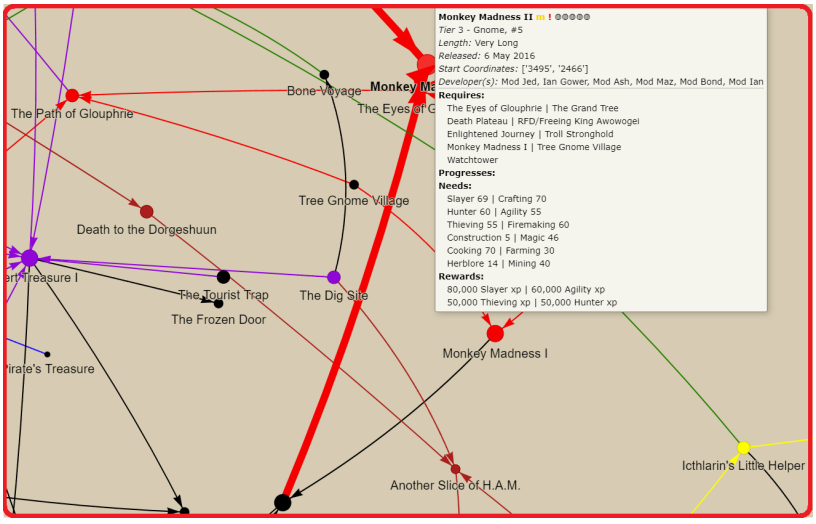



**Figure 1.3:** A closer view on the Gnome series, which highlights Monkey Madness II, detailing its information in a tooltip. The thick edges represent the direct connections to this quest. Another iconic quest, Desert Treasure I is also pictured. The physical proximity in the graph tends to represent their relatedness i.e. quests in the desert tend to be near each other.

# Part VII

# Chunks

# Chapter 1

# Solving the Lumbridge Chunk

# Chunk Solver

This document provides a basic approach to solving chunk accounts mathematically. This includes describing the gamestate before entering the chunk, the gamestate after completing the chunk, and the time taken to complete the chunk (where the latter two are taken to be probability distributions). By solving each chunk generically, it would be feasible to build an exact solver for any sequence of chunks.

## 1. The Chunk Picker

A "chunk account" has the restriction that it can only access a set number of chunks, which divide the 2D surface map.

$$S_N = \{(x_1, y_1), (x_2, y_2), \dots (x_N, y_N)\}$$

It is more common to enumerate the chunks themselves instead of using coordinates, which we can then express as:

$$S_N = \{c_1, c_2, \dots, c_N\}.$$

In most cases, players begin with a single chunk $c_1 = (x_1, y_1)$ (although it's easy to imagine starting with an initial set). Within each chunk, the player must achieve certain objectives that have requirements. Due to the severe restriction (especially early on), these tasks may take very long times. Regardless of what objectives are set, the chunks evolve in distinct epochs:

$$S_N(\sigma_N, t_N) \to S_{N+1}(\sigma_N + \Delta\sigma_N, t_N + \tilde{t}\,)$$

Where here we denote the game state as $\sigma$, which includes the player's stats, bank, inventory, etc., with $\Delta\sigma$ representing its change. Finally, we start the chunk at time $t_0 \in \mathbb{N}$, which is measured in ticks, and end according to a distribution $\tilde{t}$ since the game has random elements. Once a chunk is completed $t_{N+1}$ is set, and the equation iterates until all chunks are completed or some other end-state (eg: maxing) is achieved.

## 2. Chunk Evolution

We can denote the operator $\hat{P}$, which evolves the chunk set:

$$\hat{P}S_N = S_{N+1}.$$

We can assume that one chunk is selected at a time, so that:

$$|\hat{P}S_N| = |S_N| + 1$$

$$|S_N| = N.$$

The set of chunks to choose from are normally taken to be the adjecent set of chunks to $S_N$. Here is a tool by Source Chunk to generate chunks.

# 3. Within a Chunk

The set of chunks a player has is largely random while generally being much easier to model than the description of the solution for a chunk, so the focus is more on solving individual chunks. This means taking, as input, the game state at the begining of any chunk, and determining both the output game state, and the time distribution to complete it (an average estimate could also be considered a solution).

Let's consider an example. Here are the basic requirements specified by Limpwurt on his account that starts with $c_1$ as the initial game spawn point, Lumbridge. In the picture below, you can see that he effectively sets out these requirements to be accomplished within the initial chunk:

- 99 woodcutting
- 99 defence
- 72 fletching (boostable)
- 60 firemaking


Let's just focus the analysis on one of these, like 60 firemaking. In fact, if we restrict our account even further, it will simplify things; its best to start with the simplest case. If the account cannot use shops as well for example, it would make woodcutting not possible and would sigificantly restict the source of firemaking material. Furthermore if a bank cannot be used, as is the case of an ultimate ironman, it further means that techniques to store items cannot be used and again we simplify the solution by restricting the search space (set of possible actions). This is important since it would be possible that from a previous chunk we obtained lots of various logs (eg: oak log), then we would need to plan for every possible input item that may provide experience. Meaning, we would need to start with an initial vector $(x, y, z, \dots)$, where each element presents the number of (normal logs, oak logs, willow logs, etc.). This is so that we can connect any two chunks together ($\sigma_N \rightarrow \sigma_{N+1}$), but again for simplicity we assume no other items to begin with.

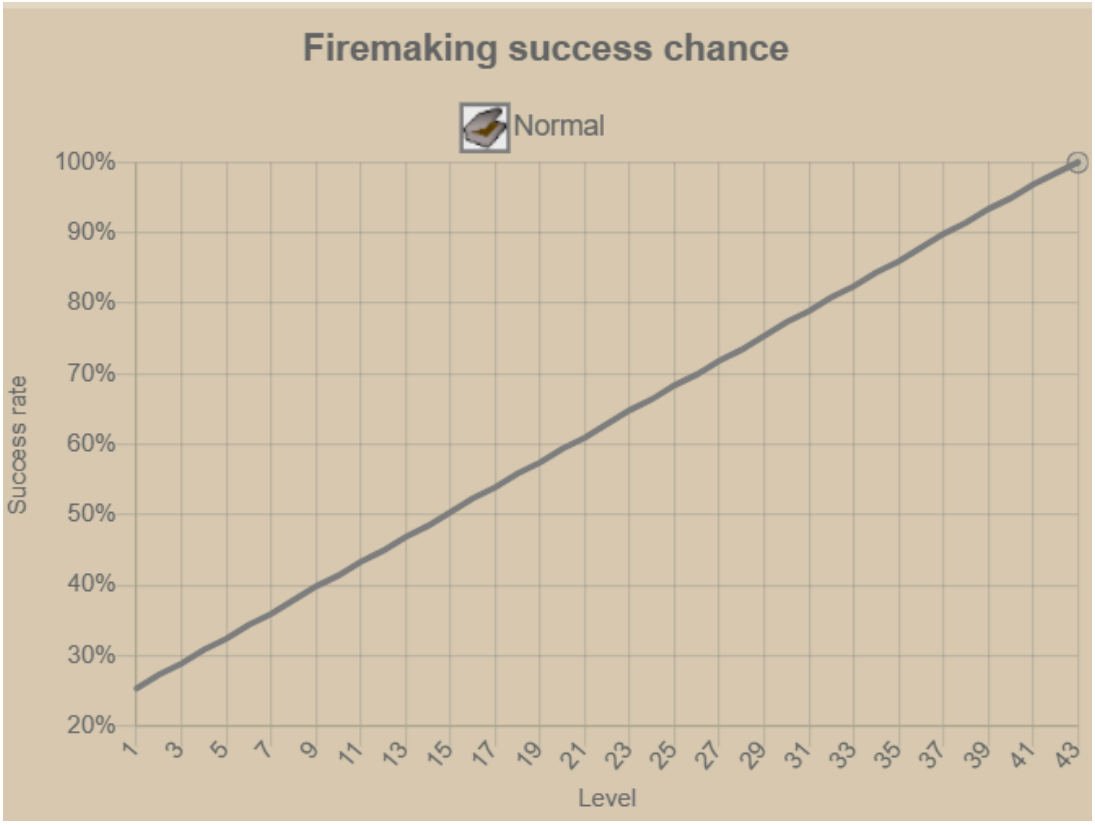
Under these most restrictive rules, the only way to train firemaking is by picking up from the four log spawns in the castle and simply burning them. We will also assume the character switches worlds instantly and does so after every time it picks up the 4 logs. Furthermore, since we expect to consume each log, there is no change in $\sigma$ other than the player's firemaking level.


The player starts with level one firemaking, and we can assume that it took $T_1$ ticks to obtain the tinderbox and to then stand on the first log spawn location.

$$L_1^{FM} = 1; t_1 = T_1$$

Now, the only actions the player can perform is to pick up the logs and burn them. There are four, so there could be the question of what sequence should the logs be picked up. The number of ticks required to pick up the logs depends on the sequence of tiles. The picture above denotes the most logical order. Here is the sequence of number of tiles travelled, where 2 represents a run, and 1 represents a walk: 2; 2, 1; 1, 1. The player can run up to 2 tiles per tick, so we simply have 5 ticks to pick up the logs. The time to return to the initial spot is ignored since we assume instant switching to a fresh world. To get from 1 to 60 requires 6844 logs, which would add 34,220 ticks for collection time (call this $T_2$).

The next step is to calculate how long it would take to burn 6844 logs without pause. This turns into a recursive equation because the burn rate depends on the level. 25 experience per log and according to the following picture, the rate goes from "65/256 at level 1 and 513/256 at level 99 (with 256/256 being achieved at 43), interpolating linearly in between". From the runescape wiki, it seems like it takes 4 ticks per attempt.




So the number of ticks is expressed as:

$$t_L = 4 * r_L * n_L$$

where $n_L$ is the number of logs required to level up from level $L$ to $L+1$, and $r_L$ is probability of sucessfully burning the logs. In fact at level 43, the rate is 100%, which means that we always burn sucessfully and the number of ticks is simply 4 ticks per log, with 43-60 requiring 5586 logs and so $T_4 = 22,344$ ticks. This leaves 1,259 logs from level 1 to level 43 in a stepwise fashion checking the number of logs for each increment. This is done in the code below, which calculates $\tilde{T_3} = 7,191$, on average (in practice it is a distribution, so we denote with tilde). The time it takes while learning the skill is given by

$$\tilde{T_3} = \sum_{L=1}^{43} t_L,$$

where $t_L$ is the time taken per level.

Putting everything together, the solution for this account is $\tilde{t} = T_1 + T_2 + \tilde{T_3} + T_4$ and $\sigma_N = \sigma_{N+1}$ meaning $\Delta\sigma = \vec{0}$. Finally, we assume the player retains their initial tinderbox and takes about 16s (27 ticks) to get up the staircases estimating:

$$\tilde{t} = T_{\text{tinderbox}} + T_{\text{collecting}} + \tilde{T}_{\text{learning}} + T_{\text{mastery}}$$

$$\tilde{t} = 27 + 34,220 + 7,191 + 22,344$$

$$\tilde{t} = 63,782.$$

Therefore is takes about 10.6h to beat this version of the initial lumbridge chunk, and 60 firemaking (273780 experience) is the only thing earned.

# 5. Discussion

This shows a proof of concept for the ability to solve a chunk exactly. It's possible to include probability distributions and to introduce more objectives and freedom into the calculations. Ultimately, the key to solving the chunk account more completely is to iteratively lift restrictions starting with the most specific set. This provides a systematic way to solve each chunk, and therefore estimate the minimum number of ticks possible to complete a set of chunks. In fact, if done in totality, this method would also solve the game generically. Said another way, as the number of chunks solved approaches the total number of chunks in the game, the solution becomes more exact for the general game (any input to any output).

Another example of this type of solution is solving the the optimal questing order, which involves assessing the time distribution (eg: how many ticks of walking, or running, etc does each quest take?), along with what was consumed and what was obtained (again, captured by $\sigma$ and $\tilde{t}$).

In [13]:
```
## Calculating T3 (learning)
# Here is the number of logs required between each level from 1, 43
TIME_PER_LOG = 4
logs_needed_per_level = [
    3, 3, 3, 3, 4, 4, 4, 5, 5, 6,
    6, 7, 7, 8, 9, 10, 11, 12, 13, 14,
    16, 17, 19, 21, 23, 25, 28, 31, 34, 37,
    41, 45, 50, 55, 61, 67, 74, 81, 90, 99,
    109, 121, 133
]
starting_scaling_factor = 65 / 256  # Scaling factor at level 1
ending_scaling_factor = 1  # Scaling factor at level 43 (256/256)
rate = (ending_scaling_factor - starting_scaling_factor) / (43 - 1)

total_time = 0
for level, logs_needed in enumerate(logs_needed_per_level, 1):
    r = 0.25 + rate * (level - 1)
    time = TIME_PER_LOG * logs_needed / r
    total_time += time
```

```
    print(level, total_time, time)

print("total_time: ", total_time)
```

```
1 48.0 48.0
2 92.8155609586662 44.815560958666204
3 134.8429225221841 42.027361563517914
4 174.40869989108933 39.565777368905245
5 224.2441575967672 49.83545770567787
6 271.46677631760167 47.222618720834475
7 316.3368858637676 44.870109546165885
8 369.7629728202893 53.42608695652174
9 420.7686654199098 51.005692599620495
10 479.32278386783355 58.554118447923756
11 535.4445759513482 56.12179208351457
12 598.3086394301538 62.8640634788056
13 658.7616514783466 60.453012048192775
14 725.2988783203501 66.53722684200348
15 797.4868678764859 72.18798955613578
16 874.9368948889116 77.4500270124257
17 957.2990118805551 82.36211699164346
18 1044.2570573733942 86.9580454928391
19 1135.5244422901426 91.26738491674828
20 1230.84057336502 95.31613107487732
21 1336.5762832605333 105.73570989551321
22 1445.7173490912542 109.141065830721
23 1564.3170588154921 118.59970972423804
24 1691.8651452439626 127.54808642847057
25 1827.8915478842266 136.02640264026402
26 1971.9622969127674 144.0707490285408
27 2129.2949334426867 157.332636529919
28 2299.2436906128587 169.9487571701721
29 2481.208847407284 181.9651567944251
30 2674.6324526096664 193.4236052023824
31 2884.1035858954824 209.4711332858161
32 3108.910183642027 224.8065977465443
33 3353.2738200056633 244.36363636363637
34 3616.3635764287533 263.0897564230898
35 3902.0843344335453 285.720758004792
36 4209.5137593775335 307.42942494398807
37 4542.28163892584 332.76787954830615
38 4899.395734261565 357.11409533572527
39 5288.569271361906 389.1735371003419
40 5708.594130787772 420.02485942586566
41 6162.493433653767 453.89930286599537
42 6657.214224603477 494.7207909497101
43 7191.300499113281 534.086274509804
total_time:  7191.300499113281
```

# Bibliography


[1] 10 most played mmorpgs of 2020. https://bestreamer.com/gaming/most-played-mmorpg-2019/2/. Retrieved October 02, 2020.

[2] Top 6 most popular mmorpgs sorted by population (2020). https://altarofgaming.com/all-mmos-sorted-by-population-2018/. Retrieved October 02, 2020.

[3] Ranking the 15 best mmorpgs of all time. https://www.thegamer.com/best-mmorpgs-ever-wow-runescape/. Retrieved October 02, 2020.

[4] Misplacedme. Relative player counts in runescape. https://www.misplaceditems.com/rs_tools/graph/.

[5] Wiki. Skills. https://oldschool.runescape.wiki/w/Skills.

[6] Wiki. Experience. https://oldschool.runescape.wiki/w/Experience.

[7] Wiki. Quests. https://oldschool.runescape.wiki/w/Quests.

[8] Wiki. Combat triangle. https://oldschool.runescape.wiki/w/Combat_triangle.

[9] Dps calculator by bitterkoekje. https://docs.google.com/spreadsheets/d/1wzy1VxNWEAAc0FQyDAdpiFggAfn5U6RGPp2CisAHZW8/edit#gid=158500257.

[10] Nukelawe. Markov chain analysis of overkill and natural regeneration in osrs combat. https://github.com/Nukelawe/osrs-markov/blob/master/main.pdf.

[11] Wiki. Calculator:wintertodt supply crate. https://oldschool.runescape.wiki/w/Calculator:Wintertodt_supply_crate. Retrieved November 04, 2020.

[12] HereticAcinonyx. Loot from 50-99 fm at wintertodt post-nerf (level 1 skills). https://imgur.com/r/2007scape/WKrov.

[13] OSRSBox. Analysis of wintertodt damage at 10 hitpoints. https://www.osrsbox.com/blog/2020/07/04/analysing-wintertodt-damage-at-10-hitpoints/.