An integer programming approach to RuneQuest 3e training calculations

Timothy Rice

2018

Contents

1 Introduction

RuneQuest 3rd Edition (1) features a dynamic skill system that allows characters to improve skills if and only if they exert those skills. This can be through practical experience—for example, if you successfully use first aid during combat, you will unlock a chance to improve your character's first aid skill. Another way to exert a skill is to devote quiet periods in a character's life to researching and training in the skill. One convention is to allow fifty hours training for each week in which the character is not actively adventuring.

This document will focus on the mechanics of training, rather than of practical adventurous experience.

Each time you want to increase a skill through training, your character must invest a pre-determined number of hours based on the current skill rank. Each such increase is small by itself, but the process can be repeated an arbitrary number of times, so long as hours are available.

As the skill increases, each increment takes more training time to complete. This algorithm admits fast increases for low-ranked skills, but those unwilling to go on adventures will find ultimate mastery recedes beyond their reach.

Formally, the training procedure goes like so:

1. Decide in advance whether your skill increase will take the form of a random $1\,\mathrm{D}\,6-2$ increase, or a deterministic increase by exactly 2%. (Naturally you can't roll the random outcome first and then take the 2% if the roll is bad.) Let the outcome of this decision be called i.

2. Let the current skill be denoted s (which is always an integer).

   - If $s \leq 0$ then one hour is required to raise the skill by an amount equal to i.
   - Otherwise if $s \geq 1$ the amount of training required to increase the skill from s% to $(s+i)$% is s hours.

Since the expected value of $1\,\mathrm{D}\,6-2=1.5$, most people will choose to take the flat 2% increase. As such the present document will elide discussion of the stochastic case.

At this point, those wishing to chart the course of their character's future progression will take interest in questions such as:

- If I want to increase a particular skill to a desired level, how many hours do I need to invest?

- If I have h hours to invest in a skill, what progression will I see?

Past efforts to answer these questions have involved manual iteration or computational simulation of the training process. Although such methods have their place in problems that resist analytical solutions, it turns out that closed-form expressions can be derived which answer the questions once and for all.

2 Hours required for multiple increases

In this section we explore the first question; the results we obtain will prove useful for answering the second question.

We first require some notation, then will consider the simplest case when the starting skill is greater than zero. This will ultimately lead to a general formula that encompasses any starting skill, as well as tolerating deviations from the RuneQuest 3e canonical rules.

2.1 Notation

For a given skill that we want to increase repetitively, let s_n denote the skill rank after the n^{th} increase; it is consistent with this definition to have s_0 represent the pre-training skill percentage. The following fact is worth noting:

$$s_n = s_0 + in. \tag{1}$$

Analogously to s_n, let t_n denote the cumulative time to achieve the n^{th} increase. For example, if we start from $s_0 = 0$ it will take an hour to increase the skill to two. Then $s_1 = 2$ and $t_1 = 1$. Without loss of generality, assume $t_0 = 0$.

2.2 The case when $s_0 > 0$

Let's begin with an example where $s_0 = 1$. Clearly, $t_1 = s_0 = 1$; then $s_1 = s_0 + i$ which will bring the skill up to three (using $i = 2$). The training required to achieve the next improvement is thus also three, or $t_2 = t_1 + s_1 = 1 + 3 = 4$ hours.

In other words, starting with a skill of 1%, we have invested four hours to achieve two improvements, bringing the skill up to 5%. Not bad for one morning's work!

Putting it together we can see a pattern emerge:

$$
\begin{aligned}
t_1 &= s_0 \\
t_2 &= t_1 + s_1 \\
&= s_0 + s_0 + i \\
&= 2s_0 + i \\
t_3 &= t_2 + s_2 \\
&= 2s_0 + i + s_0 + 2i \\
&= 3s_0 + 3i \\
t_4 &= t_3 + s_3 \\
&= 3s_0 + 3i + s_0 + 3_i \\
&= 4s_0 + 6_i \\
t_5 &= t_4 + s_4 \\
&= 4s_0 + 6_i + s_0 + 4_i \\
&= 5s_0 + 10_i.
\end{aligned}
$$

Notice how the coefficient of s_0 is incrementing by one at each step, but the coefficient of i is forming a pattern that looks suspiciously like triangular numbers:

$$
\begin{aligned}
1 &= 1 \\
1 + 2 &= 3 \\
1 + 2 + 3 &= 6 \\
1 + 2 + 3 + 4 &= 10.
\end{aligned}
$$

The triangular numbers are also called binomial coefficients, and there exists a concise notation to express them:

$$\binom{n}{2} = \frac{n(n-1)}{2}.$$

By substituting into this formula, we can see that $\binom{1}{2} = 0$, $\binom{2}{2} = 1$, $\binom{3}{2} = 3$, $\binom{4}{2} = 6$, and $\binom{5}{2} = 10$. These all match the coefficients of i calculated above.

Such observations lead us to postulate a general formula for t_n:

$$t_n = ns_0 + \binom{n}{2} i.$$

We can use induction to prove it holds not just for these few n but for all integer $n > 0$, so long as $s_0 > 0$.

Lemma 2.1. *Assume we have an example n for which we know that $t_n = ns_0 + \binom{n}{2}i$. (We already showed above that such examples exist.)*

Now consider t_{n+1}, which we know can be calculated as $t_{n+1} = t_n + s_n$. (I.e. if we've already worked at this skill for t_n hours, it is going to require another s_n hours to achieve the next skill increase.)

Then,

$$\begin{aligned}
t_{n+1} &= t_n + s_n \\
&= ns_0 + \binom{n}{2}i + s_0 + ni \\
&= (n+1)s_0 + \frac{n(n-1)}{2}i + \frac{2n}{2}i \\
&= (n+1)s_0 + \frac{n(n-1)+2n}{2}i \\
&= (n+1)s_0 + \frac{n(n-1+2)}{2}i \\
&= (n+1)s_0 + \frac{(n+1)n}{2}i \\
&= (n+1)s_0 + \binom{n+1}{2}i
\end{aligned}$$

This follows the same pattern as noticed for t_n but all n have increased by one; thus by induction it is proved to be a general formula for all $n > 0$.

2.2.1 Examples

Suppose we currently have a skill at 5% and we want to increase it 2% at a time until we obtain 15%. Then we use $s_0 = 5$, $i = 2$, and $n = 5$:

$$\begin{aligned}
t_5 &= \left(ns_0 + \binom{n}{2}i\right)\bigg|_{n=5} \\
&= 5 \times 5 + \frac{5 \times (5-1)}{2} \times 2 \\
&= 25 + 20 \\
&= 45.
\end{aligned}$$

Thus our character would need to spend most of a week to achieve this improvement.

Using the same formula, a character with a current skill of 50% would have to spend an entire week of training to go up to 52%.

2.3 The case $s_0 \leq 0$

Observe the case when $s_0 = 0$:

$$\begin{aligned}
t_1 &= 1 \\
s_1 &= i \\
t_2 &= t_1 + s_1 \\
&= 1 + i \\
s_2 &= 2i \\
t_3 &= t_2 + s_2 \\
&= 1 + i + 2i \\
&= 1 + 3i \\
s_3 &= 3i \\
t_4 &= t_3 + s_3 \\
&= 1 + 3i + 3i \\
&= 1 + 6i
\end{aligned}$$

We see the same pattern of binomial coefficients; the proof that $t_n = 1 + \binom{n}{2} i$ for all $n > 0$ is left as an exercise for the reader.

More generally, if $s_0 \leq 0$, we need to increase it an hour at a time until it is positive, and then we can expect to see the usual progression of binomial coefficients begin.

We can establish a general formula if we first define some more notation. Let:

- $x \vee y$ denote the *maximum* of x and y and let $x \wedge y$ denote the *minimum*. (This notation is commonly used in probability theory.) Example: $-2 \vee 1 = 1$; $-2 \wedge 1 = -2$. (This will be useful for imposing a minimum of one hour training even if the current skill is negative.)

- $|x|$ denote the *absolute value*: $|-2| = 2$. That is, turn any negative numbers into positive numbers.

- $\lfloor x \rfloor$ denote the *floor* function, which means *rounding down*. Examples: $\lfloor 2.718282 \rfloor = 2$; $\lfloor 3.1415927 \rfloor = 3$; $\lfloor -3.1415927 \rfloor = -4$.

- $I(x)$ denote the *indicator* of expression x, which equals one if x is true and 0 otherwise. Example: $I(3 \leq 0) = 0$ because $3 \not\leq 0$; but $I(3 > 0) = 1$ because it is true that $3 > 0$.

Using that notation, we can explore the progression of t_n like so:

$$
\begin{aligned}
t_1 &= 1 \vee s_0 \\
t_2 &= t_1 + 1 \vee s_1 \\
&= 1 \vee s_0 + 1 \vee (s_0 + i) \\
t_3 &= 1 \vee s_0 + 1 \vee (s_0 + i) + 1 \vee s_2 \\
&= 1 \vee s_0 + 1 \vee (s_0 + i) + 1 \vee (s_0 + 2i) \\
\vdots \quad & \vdots \quad \vdots \quad \vdots \quad \vdots \\
t_n &= 1 \vee s_0 + 1 \vee (s_0 + i) + \ldots + 1 \vee (s_0 + (n-1)i) \\
&= \sum_{j=0}^{n-1} 1 \vee (s_0 + ji). \qquad (2)
\end{aligned}
$$

Now let λ be such that $s_{\lambda-1} \leq 0$ but $s_\lambda > 0$; after a little thought we realise this can be calculated as:

$$\lambda = I(s_0 \leq 0) \left(\left\lfloor \frac{|s_0|}{i} \right\rfloor + 1 \right).$$

(The indicator at the front sets $\lambda = 0$ whenever $s_0 > 0$.)

For example, if you start with a skill at -3%, you need to raise it to -1%, then raise it again to 1% before it will become positive, thus we expect $\lambda = 2$; and indeed:

$$
\begin{aligned}
\lambda &= \left\lfloor \frac{|-3|}{2} \right\rfloor + 1 \\
&= \left\lfloor \frac{3}{2} \right\rfloor + 1 \\
&= \lfloor 1.5 \rfloor + 1 \\
&= 1 + 1 \\
&= 2.
\end{aligned}
$$

With this notation defined, we can decompose the sum in formula (2) as:

$$
\begin{aligned}
t_n &= I(\lambda > 0) \sum_{j=0}^{(\lambda \wedge n)-1} 1 + I(n > \lambda) \sum_{j=\lambda}^{n-1} (s_0 + ji) \\
&= I(\lambda > 0)(\lambda \wedge n) + I(n > \lambda) \left((n-\lambda) s_0 + i \sum_{j=\lambda}^{n-1} j \right) \\
&= I(\lambda > 0)(\lambda \wedge n) + I(n > \lambda) \left((n-\lambda) s_0 + i \sum_{k=0}^{n-\lambda-1} (k+\lambda) \right) \\
&= I(\lambda > 0)(\lambda \wedge n) + I(n > \lambda) \left((n-\lambda) s_0 + i \binom{n-\lambda}{2} + (n-\lambda) i \lambda \right) \\
&= I(\lambda > 0)(\lambda \wedge n) + I(n > \lambda) \left((n-\lambda)(s_0 + i\lambda) + i \binom{n-\lambda}{2} \right).
\end{aligned}
$$

However, we can note that $I(\lambda > 0)(\lambda \wedge n) = \lambda \wedge n$ regardless of whether $\lambda > 0$. Additionally, we can re-express $\lambda \wedge n$ in a form that will facilitating collecting terms, admitting further simplification:

$$
\begin{aligned}
\lambda \wedge n &= I(n > \lambda)\lambda + I(n \le \lambda) n \\
&= I(n > \lambda)\lambda + (1 - I(n > \lambda))\, n \\
&= I(n > \lambda)(\lambda - n) + n \\
&= n - I(n > \lambda)(n - \lambda). \qquad (3)
\end{aligned}
$$

Using equation (3) we can now rewrite t_n in the form:

$$
\begin{aligned}
t_n &= I(\lambda > 0)(\lambda \wedge n) + I(n > \lambda) \left((n-\lambda)(s_0 + i\lambda) + i \binom{n-\lambda}{2} \right) \\
&= n - I(n > \lambda)(n-\lambda) + I(n > \lambda) \left((n-\lambda)(s_0 + i\lambda) + i \binom{n-\lambda}{2} \right) \\
&= n + I(n > \lambda) \left((n-\lambda)(s_0 + i\lambda) - (n-\lambda) + i \binom{n-\lambda}{2} \right) \\
&= n + I(n > \lambda) \left((n-\lambda)(s_0 + i\lambda - 1) + i \binom{n-\lambda}{2} \right). \qquad (4)
\end{aligned}
$$

In the case $i = 2$, this simplifies further:

$$\begin{aligned} t_n &= n + I(n > \lambda)\left((n-\lambda)(s_0 + i\lambda - 1) + i\binom{n-\lambda}{2}\right) \\ &= n + I(n > \lambda)\left((n-\lambda)(s_0 + 2\lambda - 1) + 2\binom{n-\lambda}{2}\right) \\ &= n + I(n > \lambda)\left((n-\lambda)(s_0 + 2\lambda - 1) + (n-\lambda)(n-\lambda-1)\right) \\ &= n + I(n > \lambda)(n-\lambda)(n + s_0 + \lambda - 2). \end{aligned} \tag{5}$$

Finally, when $s_0 > 0$ (which is most of the time) it reduces to:

$$\begin{aligned} t_n &= n + n(n + s_0 - 2) \\ &= n(n + s_0 - 1). \end{aligned} \tag{6}$$

3 Increases attained after set hours

The inverse problem to the previous section is to set in advance the number of hours that may be allocated to increasing a skill, and then ask what will this raise the skill too. Again we assume that i is a fixed positive integer, n is a non-negative integer, and the s_n are integers. We also let h denote the fixed number of hours allocated for training the skill in question.

The problem amounts to finding the n satisfying the following two constraints:

$$\begin{aligned} t_n &\le h \\ t_{n+1} &> h \end{aligned}$$

That is, what is the n such that if we train that many times, we will not run out of hours, but if we try to do any more training (i.e. $n + 1$ times) we will exhaust the available hours and not be able to complete the final round?

This is equivalent to maximising n subject to the constraint that $t_n \le h$; amongst mathematicians this is referred to as an *integer programming* problem.

To solve this integer programming problem, it shall be useful to partition the available hours into two parts: those used bringing the skill up to above zero (if possible), and those which apply to skills already above zero.

This intuition motivates some new notation: let h^- be hours spent on the skill while it is non-positive, and let h^+ be any remaining hours spent on the skill after it becomes greater than zero. If $s_0 > 0$, $h^- = 0$ and $h = h^+$.

We also note that if $h < \lambda$, then automatically we should set $n = h$.

3.1 When $s_0 > 0$

Regardless of whether in fact $s_0 > 0$, solving this case allows us to also solve for h^+ in the more general case described above.

We start with the inequality constraint $t_n \leq h$ and unpack it using the general formula (4); then use $\lambda = 0$ to obviate the indicator; then notice the quadratics; which motivates completing the square; before taking the square root of both sides:

$$\begin{aligned}
& & t_n &\leq h \\
\Rightarrow & & n + n(s_0 - 1) + i\binom{n}{2} &\leq h \\
\Rightarrow & & ns_0 + i\frac{n(n-1)}{2} &\leq h \\
\Rightarrow & & \frac{i}{2}n^2 + \left(s_0 - \frac{i}{2}\right)n &\leq h \\
\Rightarrow & & n^2 + 2\frac{2s_0 - i}{2i}n &\leq \frac{2h}{i} \\
\Rightarrow & & n^2 + 2\frac{2s_0 - i}{2i}n + \left(\frac{2s_0 - i}{2i}\right)^2 &\leq \frac{2h}{i} + \left(\frac{2s_0 - i}{2i}\right)^2 \\
\Rightarrow & & \left(n + \frac{2s_0 - i}{2i}\right)^2 &\leq \frac{2h}{i} + \left(\frac{2s_0 - i}{2i}\right)^2.
\end{aligned}$$

We have two cases to consider here:

- If $2s_0 - i = 0$, the inequality immediately simplifies to $n^2 \leq \frac{2h}{i}$.
- On the other hand, if $2s_0 - i \neq 0$, that will allow us to bring together the $\frac{2s_0 - i}{2i}$ terms from both sides.

3.1.1 If $2s_0 - i = 0$

In this case $2s_0 = i$, so the inequality devolves to:

$$\begin{aligned}
& & n &\leq \pm\sqrt{\frac{2h}{i}} \\
\Rightarrow & & n &\leq \pm\sqrt{\frac{2h}{2s_0}} \\
\Rightarrow & & n &\leq \pm\sqrt{\frac{h}{s_0}}.
\end{aligned}$$

Rejecting the solution where $n < 0$, this becomes,

$$n \leq \sqrt{\frac{h}{s_0}}.$$

Then maximising n while satisfying this inequality leads us to the solution:

$$n = \left\lfloor \sqrt{\frac{h}{s_0}} \right\rfloor.$$

For example, take $s_0 = 1$, $i = 2$, so that $2s_0 = i$ is satisfied. Now suppose $h = 50$. Then $n = \lfloor\sqrt{50}\rfloor$, where $\sqrt{50} \approx 7.07$, leading to the result $n = \lfloor 7.07 \rfloor = 7$.

This concords with a manual calculation which show that in a week of training, a character starting with a skill at 1% could raise that skill seven times, achieving 15% by the end of the week.

3.1.2 If $2s_0 - i \neq 0$

In this case the earlier full inequality (4) can be manipulated like so:

$$\begin{aligned}
& n + \frac{2s_0 - i}{2i} \leq \pm\sqrt{\frac{2h}{i} + \frac{(2s_0 - i)^2}{4i^2}} \\
\Rightarrow \quad & n + \frac{2s_0 - i}{2i} \leq \pm\frac{2s_0 - i}{2i}\sqrt{\frac{8hi}{(2s_0 - i)^2} + 1} \\
\Rightarrow \quad & n \leq -\frac{2s_0 - i}{2i} \pm \frac{2s_0 - i}{2i}\sqrt{\frac{8hi}{(2s_0 - i)^2} + 1} \\
\Rightarrow \quad & n \leq \frac{2s_0 - i}{2i}\left(-1 \pm \sqrt{\frac{8hi}{(2s_0 - i)^2} + 1}\right). \qquad (7)
\end{aligned}$$

The square root is always strictly greater than one, so we always end up with a single positive and single negative solution. Again we need to deal with it as two separate cases: $2s_0 - i > 0$ and $2s_0 - i < 0$.

Case One: $2s_0 - i < 0$: In this case the positive solution is obtained by choosing:

$$
\begin{aligned}
& n \leq \frac{2s_0 - i}{2i}\left(-1 - \sqrt{\frac{8hi}{(2s_0-i)^2} + 1}\right) \\
\Rightarrow \qquad & n \leq \frac{i - 2s_0}{2i}\left(1 + \sqrt{\frac{8hi}{(2s_0-i)^2} + 1}\right).
\end{aligned}
$$

When $i = 2$ the only time this case arises is when $s_0 = 0$. Since we need to treat any $s_0 \leq 0$ as a special case regardless of i, we will defer further discussion of this scenario until Subsection 3.2.

Case Two: $2s_0 - i > 0$**:** In this case the positive solution is chosen by writing the inequality (7) as:

$$
n \leq \frac{2s_0 - i}{2i}\left(-1 + \sqrt{\frac{8hi}{(2s_0-i)^2} + 1}\right).
$$

Again electing for $i = 2$:

$$
n \leq \frac{s_0 - 1}{2}\left(-1 + \sqrt{\frac{4h}{(s_0-1)^2} + 1}\right).
$$

An example with an initial skill of 5% and a week of training:

$$
\begin{aligned}
& n \leq \frac{4}{2}\left(-1 + \sqrt{\frac{200}{16} + 1}\right) \\
\Rightarrow \qquad & n \leq 2\left(-1 + \underbrace{\sqrt{\frac{27}{2}}}_{\approx 3.67}\right) \\
\Rightarrow \qquad & n = \lfloor 5.35 \rfloor \\
\Rightarrow \qquad & n = 5.
\end{aligned}
$$

Thus after a week, a character starting with a skill of 5% could raise it five times, to a total of 15%.

3.2 For general values of s_0

We've already established that h can be decomposed as $h = h^- + h^+$. It turns out they should furthermore have values $h^- = h \wedge \lambda$ and $h^+ = 0 \vee (h - \lambda)$. This is based on an intuitive realisation:

- If you have a negative skill, you can pump hours into it until it becomes positive, if the hours are available.
- If you do this, it removes hours from your total training time.

Just as we decompose h into h^- and h^+, we now do the same for n: let n^- be number of improvments obtained for the skill by spending hours from h^-, and let n^+ be the number of improvements obtained by spending hours from h^+.

Since hours from h^- are spent one at a time, we must have that $n^- = h^-$.

It is also useful to let s_0^+ denote the skill percentage after spending h^- hours on improving it. That is, it is the value of the skill immediately after it switches from non-positive to positive.

We can use the formula $s_0^+ = s_0 + n^- i$ since we gained $i\%$ for every hour spent on the skill while it was non-positive.

For n^+, we reuse the formulas derived for $s_0 > 0$:

$$n^+ = \begin{cases} \left\lfloor \frac{i-2s_0^+}{2i}\left(1+\sqrt{\frac{8h^+ i}{(2s_0^+-i)^2}+1}\right)\right\rfloor & \text{if } s_0^+ < \frac{i}{2}. \\ \left\lfloor \sqrt{\frac{h^+}{s_0^+}} \right\rfloor & \text{if } s_0^+ = \frac{i}{2}. \\ \left\lfloor \frac{2s_0^+-i}{2i}\left(-1+\sqrt{\frac{8h^+ i}{(2s_0^+-i)^2}+1}\right)\right\rfloor & \text{if } s_0^+ > \frac{i}{2}. \end{cases} \tag{8}$$

Or when $i = 2$ (eliding the case $s_0^+ < 1$ since that is subsumed by n^-):

$$n^+ = \begin{cases} \left\lfloor \sqrt{\frac{h^+}{s_0^+}} \right\rfloor & \text{if } s_0^+ = 1. \\ \left\lfloor \frac{s_0^+-1}{2}\left(-1+\sqrt{\frac{4h^+}{(s_0^+-1)^2}+1}\right)\right\rfloor & \text{if } s_0^+ > 1. \end{cases} \tag{9}$$

3.3 A summarised procedure

We'll now summarise what needs to be done when using the standard $i = 2$.

3.3.1 If $s_0 < 1$

1. Calculate $\lambda = \left\lfloor \frac{|s_0|}{2} \right\rfloor + 1$.
2. Calculate $h^- = h \wedge \lambda$ and $h^+ = 0 \vee (h - \lambda)$.
3. Calculate $n^- = h^-$.
4. Calculate $s_0^+ = s_0 + 2n^-$.
5. Calculate $n^+ = \left\lfloor \sqrt{\frac{h^+}{s_0^+}} \right\rfloor$ if $s_0^+ = 1$ or $n^+ = \left\lfloor \frac{1-s_0^+}{2}\left(1+\sqrt{\frac{4h^+}{(1-s_0^+)^2}+1}\right)\right\rfloor$ otherwise.

3.3.2 If $s_0 = 1$

$$n = \left\lfloor \sqrt{\frac{h}{s_0}} \right\rfloor.$$

3.3.3 If $s_0 > 1$

$$n = \left\lfloor \frac{s_0 - 1}{2} \left(-1 + \sqrt{\frac{4h}{(s_0 - 1)^2} + 1} \right) \right\rfloor.$$

This procedure lends itself to programmatic automation; a reference implementation can be found at (2). A table of results generated from that program is in Appendix A; likewise, a plot of cumulative time required per skill percentage attained is in Appendix B.

4 Conclusion

We derived novel closed-form expressions for certain quantities of interest to RuneQuest players. There are a couple of directions future work of this nature could explore:

- Analysis of the stochastic case where one chooses to roll $1\,\mathrm{D}\,6 - 2$ rather than deterministically take 2%. This could include deriving the distributions of the stochastic analogues of s_n and t_n, as well as the distribution of the first-hitting time to reach a desired skill level.

  Working with the distributions of sums of large numbers of dice can be laborious, but the Central Limit Theorem would permit one to draw on existing results about Brownian motion.

- Profile-guided optimisation of decisions about which skills to train for how long. Since most skills can be increased through practical adventuring, one will often wish to save training hours for skills not amenable to the former. These include skills at either low percentages, as well as some academic skills that cannot benefit from adventuring.

  Some back-of-the-envelope calculations suggest training should not be done on practical skills above about 30%. Using statistics collected from real gameplay would assist with the development of an optimal regime for strategic character development.

A Tabulated results

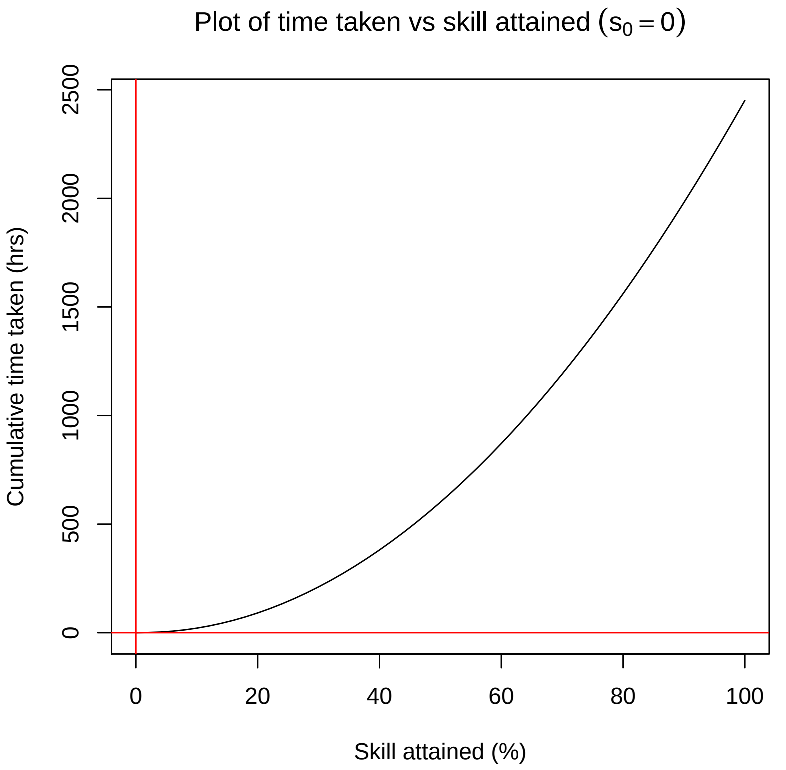
	Number of times to increase skill									
Skill	5	10	15	20	25	30	35	40	45	50
-5	7	52	147	292	487	732	1027	1372	1767	2212
0	21	91	211	381	601	871	1191	1561	1981	2451
5	45	140	285	480	725	1020	1365	1760	2205	2700
10	70	190	360	580	850	1170	1540	1960	2430	2950
15	95	240	435	680	975	1320	1715	2160	2655	3200
20	120	290	510	780	1100	1470	1890	2360	2880	3450
25	145	340	585	880	1225	1620	2065	2560	3105	3700
30	170	390	660	980	1350	1770	2240	2760	3330	3950
35	195	440	735	1080	1475	1920	2415	2960	3555	4200
40	220	490	810	1180	1600	2070	2590	3160	3780	4450
45	245	540	885	1280	1725	2220	2765	3360	4005	4700
50	270	590	960	1380	1850	2370	2940	3560	4230	4950
55	295	640	1035	1480	1975	2520	3115	3760	4455	5200
60	320	690	1110	1580	2100	2670	3290	3960	4680	5450
65	345	740	1185	1680	2225	2820	3465	4160	4905	5700
70	370	790	1260	1780	2350	2970	3640	4360	5130	5950
75	395	840	1335	1880	2475	3120	3815	4560	5355	6200
80	420	890	1410	1980	2600	3270	3990	4760	5580	6450
85	445	940	1485	2080	2725	3420	4165	4960	5805	6700
90	470	990	1560	2180	2850	3570	4340	5160	6030	6950
95	495	1040	1635	2280	2975	3720	4515	5360	6255	7200
100	520	1090	1710	2380	3100	3870	4690	5560	6480	7450

Table 1: Number of hours to increase skills chosen number of times

	Hours invested									
Skill	5	10	15	20	25	30	35	40	45	50
-5	4	5	6	7	7	8	8	9	9	9
0	2	3	4	4	5	5	6	6	7	7
5	1	1	2	2	3	3	4	4	5	5
10	0	1	1	1	2	2	2	3	3	3
15	0	0	0	1	1	1	2	2	2	2
20	0	0	0	1	1	1	1	1	2	2
25	0	0	0	0	0	1	1	1	1	1
30	0	0	0	0	0	0	1	1	1	1
35	0	0	0	0	0	0	1	1	1	1
40	0	0	0	0	0	0	0	1	1	1
45	0	0	0	0	0	0	0	0	0	1
50	0	0	0	0	0	0	0	0	0	1
55	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0

Table 2: Number of skill increases possible for hours invested

B Plots



References

[1] Steve Perrin, Greg Stafford, Steve Henderson, and Lynn Willis. *RuneQuest*. Avalon Hill, 1984.

[2] Timothy Rice. RQ3 Training Calculator. notabug.org/cryptarch/rqt, 2018.