- Turn your life into a game.
- Exploit desires to become consistent at goals

Desires are PRODUCTIVE FUEL

(i.e. motivation)

- As you complete your tasks you can claim a currency; you then use this currency to purchase the ability to consume desires.

Requires discipline, but provides motivation.

The concept of exploiting desires will only work if you have desires in the first place.

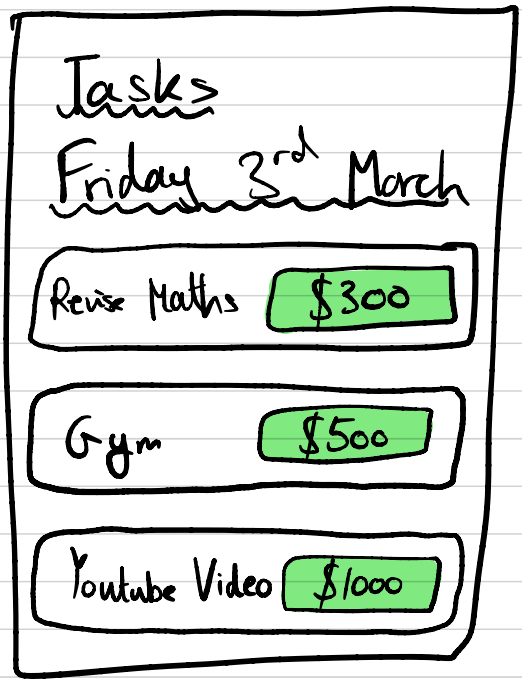
⇒ If your baseline dopamine is too high then even these desires will not provide motivation (excessive indulgence).

⇒ A dopamine detox can fix this.

Features

- Task section where you see all tasks to be completed today
- Reward section where you can 'cash in' currency for specific rewards
- Setup page.

Tasks Screen



- There will be both pre-scheduled tasks and new one-time added recently, e.g. complete homework.
- Algorithms should be used to decide how much each task is worth.
- If tasks run over:
  i) Their reward could decrease to incentivise the user to complete a task as soon as possible.
  ii) Their reward could increase to incentivise the user to complete back-logged tasks first.

Since everything is controlled by the user themselves during setup, there need to be mechanisms to prevent the user from over-pricing any tasks (which would allow them to gain access to rewards without completing sufficient tasks).

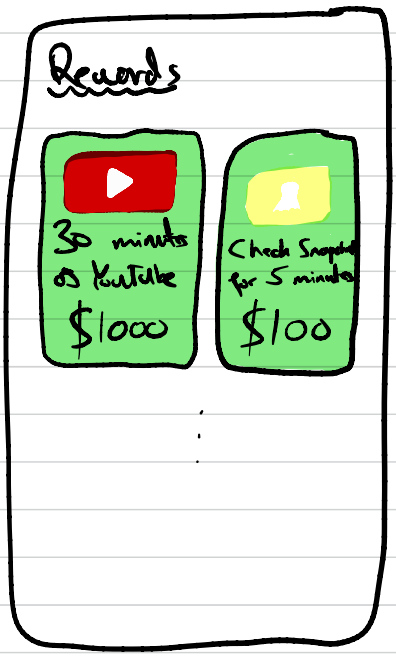
↳ Once the initial price for a task has been set, make it very difficult to change its price.

E.g. only allow a ±10% change and make user wait at least 24 hours until change comes into effect.

The main idea is to prevent users from letting their tiredness / lack of motivation cause them to make bad decisions ⇒ delay any possible decision to allow the user to reconsider it fully.

↳ To assign a fair price, an algorithm should be used which accounts for:
task benefit, difficulty, frequency (can the task be claimed multiple times?), importance, etc...

Rewards



- Use currency you have earned to purchase rewards
- Rewards will generally be priced higher than tasks since we want to incentivise completing more tasks than rewards.
- The user MUST ensure they only access rewards through the app, as otherwise this system will not work.
- Rewards can be depricating, however some should be self-rewarding, e.g. a reward could be allowing yourself to listen to music for one revision session

The reward paywall can be enforced by preventing the user from engaging in dishonest behavior.

↳ E.g. make them 'sign' something everytime they try to claim a reward, as if they are signing a contract.

Setup

This screen is where the user sets up their tasks and rewards.

It's important to make the list of tasks and rewards as exhaustive as possible, as externalities could cause dishonest behavior which would undermine the app's system.

Task Parameters:

- Task name
- Schedule, e.g. everyday, on specific days, or one time (in which case give a date).
- Task payout → This will be determined by an algorithm and user's judgement
- How many times can it be claimed per day, e.g. 'Leetcode Question' could be claimed upto 5 times per day.

Reward Parameters:

- Reward name
- Reward cost

Can also add other parameters such as image to make rewards more appealing/fun.

Goals

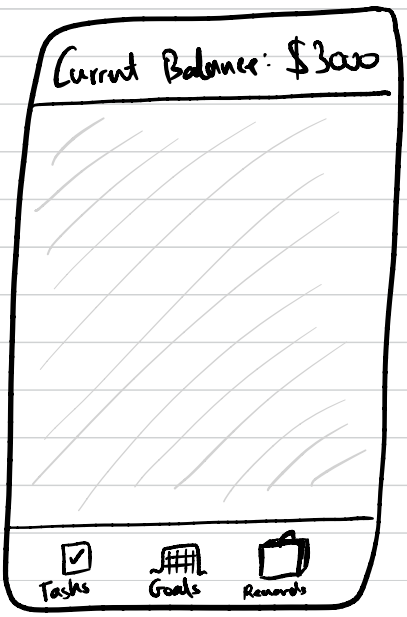
In addition to short-term tasks, a long-term goal section would also be very useful as it not only would provide feedback that progress is being made, but would also likely provide TH user with high rewards, and hence strong motivation.

Goals page prototype:

- These goals are always one-time, and hence can provide huge rewards since they are large milestones.
- User honesty is critical here as well, and to prevent excessive goals from being created:
  - ↳ Place a limit on the number of goals a user can create at once
  - ↳ Use an algorithm to determine the payout rather than having it entirely in the user's control.

Make it difficult for the user to modify the setup once it is completed to prevent 'cheating'

The complete app may look something like:



Fee

To prevent the user from just becoming dormant, the app should implement a fee system.

Everyday, a certain sum of money will be transferred out of the account.

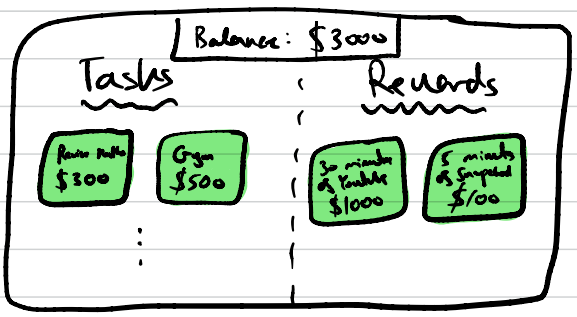
↳ This ensures the user must complete at least a few tasks per day to not lose money.

The fee will only take money until the user has $0, after which nothing happens (the user cannot go into debt).

The specific fee could be controlled by the user or an algorithm, as it essentially dictates how aggressive a user must be at completing their tasks.

Widget

A widget form of the app where the user can quickly access tasks and rewards would reduce the difficulty of logging their activity, hence reducing the chance of a leakage/externality and thus promoting honest behaviour.



Progress

Seeing progress is important for maintaining motivation. Therefore the app will incorporate some kind of progress tracker on a timeline.

The colour may indicate the quantity of currency earned per day (each dot represents 1 day).

Enforcement

This is likely to be the largest issue / breaking point. If all actions were entirely determined by the app then this would work flawlessly, however externalities (unworthy rewards) exist.

Strategies:

- Making the user sign a contract.
- Discipline / good behaviour
- Use of screen time and shortcuts to gain some control over user's behaviour

Investigate 'One sec' app

- Ways of verifying a task has been completed

Investigate 'Forfeit' app

Algorithm

TaskPayout

- The utility of this app depends on setting parameters such as task payout, reward cost and daily tax rate correctly.
- Task payout has a damping function depending on how late the task is (in days).

$$DampingFunction(x) = e^{-0.4x}$$

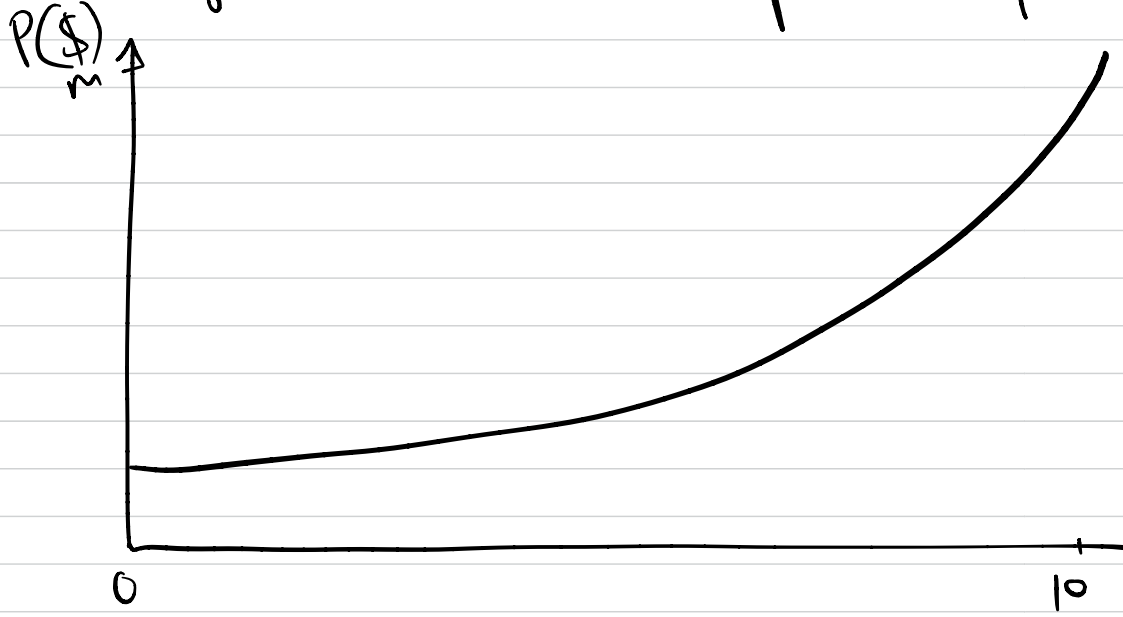
First let's set the rough magnitude we want to be working in: $\sim \$10^2$, i.e. most tasks will be a few hundred credits.

- Task payout is a function of the individual task's benefit/importance, difficulty, time.

Payout (b = benefit, d = difficulty, t = time):

↳ Assume all variables are normalised to a range between 0 and 10 inclusive, except for $\$$.

- Benefit is the most important parameter.

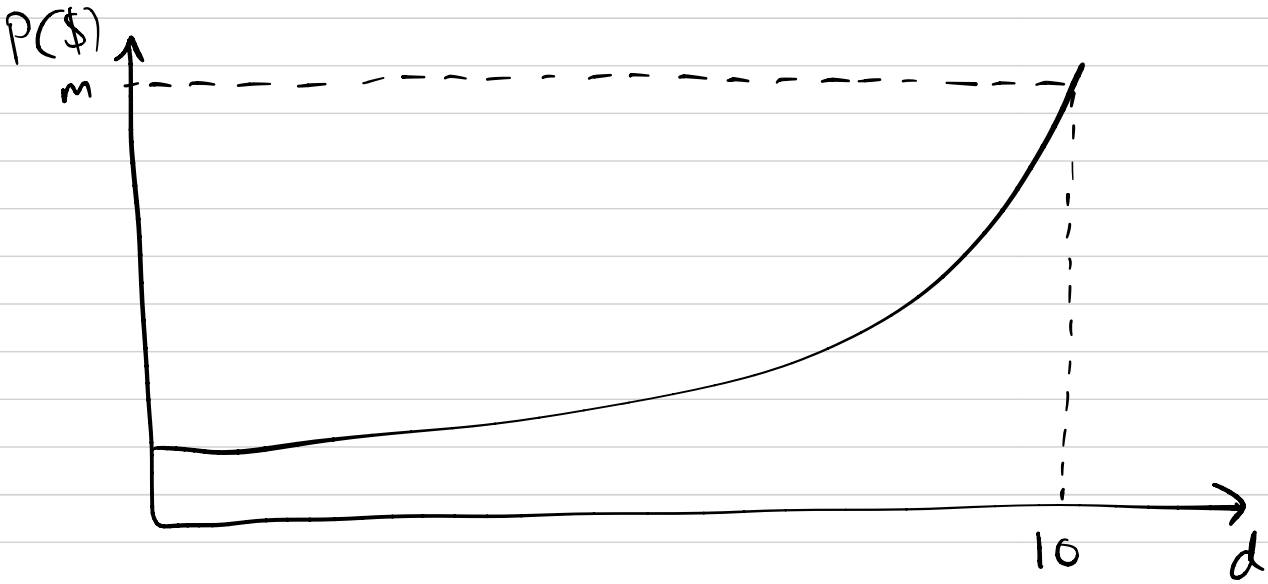


- Payout increases exponentially as benefit increases.

When drawing these graphs, assume all other parameters are constant or 0.

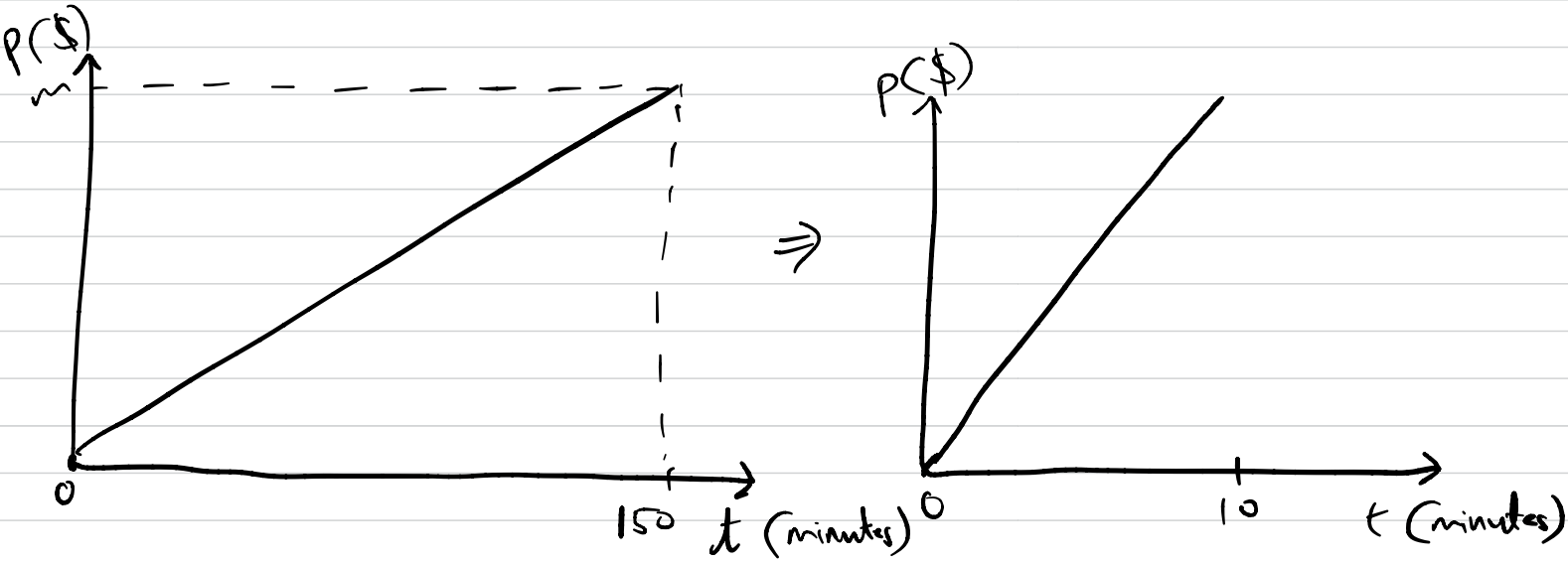
m is the maximum payout associated with a certain task

- Difficulty is also another important factor, and it will also be modelled under an exponential curve.



Time should be a linear variable (analogous to real life pay per hour), however we also need to consider the normalisation between 0 and 10.

Assume the greatest time spent on any task is 2.5 hours = 150 minutes (greater than this and the task should be split).



$$\left.\begin{array}{l} P \propto e^{b} \\ P \propto e^{d} \\ P \propto t + C \end{array}\right\} \Rightarrow P = A_1 e^{k_1 b} + A_2 e^{k_2 d} + A_3 t + k_3$$

Not proper use of $\propto$ sign; just showing rough relationships.

- Now we need to decide the constants of proportionality A_1, k_1, A_2, k_2, A_3 and k_3

Axioms

- The individual factors' importance varies, e.g. benefit is more important than time.

Weightage:

Benefit	40%
Difficulty	25%
Time	35%

'Best' task: $P(10, 10, 10) = 1500$
e.g. Revising $2\frac{1}{2}$ hours for a very hard test tomorrow

'Average' task: $P(5, 5, 5) = 300$
e.g. Completing difficult school homework

'Worst' task: $P(0, 0, 0) = 20$
e.g. Completing a quick favour

Distribution (P(b), P(d), P(t))

600, 375, 525

120, 75, 105

8, 5, 7

Using exponential and linear functions with 2 constants each, we cannot 'match' all 3 points.

Therefore I will match the 'best' and 'worst' task, and evaluate the 'average' task's payout.

$P(b) = A_1 e^{k_1 b}$; $P(0) = 8$, $P(10) = 600$

$\therefore A_1 = 8, k_1 = 0.432$

$P(d) = A_2 e^{k_2 b}$; $P(0) = 5$, $P(10) = 375$

$\therefore A_2 = 5, k_2 = 0.432$

$P(t) = A_3 t + k_3$; $P(0) = 7$, $P(10) = 525$

$\therefore A_3 = 52, k_3 = 7$

These values are rounded and the methodology is very rough, however it provides us with a payout function.

$P(b, d, t) = 8e^{0.432b} + 5e^{0.432d} + 52t + 7$

'Average' case: $P(5, 5, 5) = 380$

- This is a reasonable average payout.

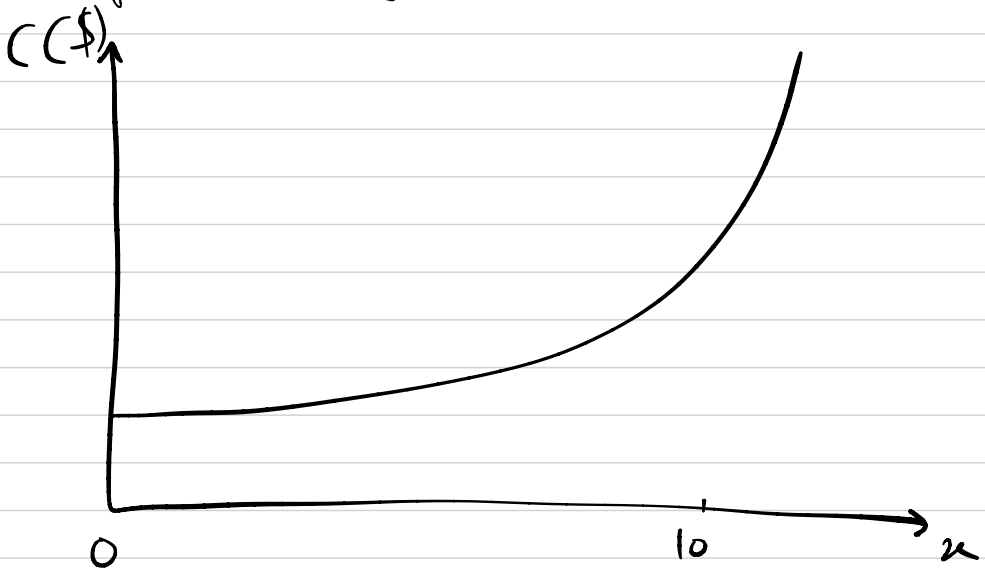
Reward Cost

- Corresponding with the task payout function, we can create the reward cost function.
- Bear in mind, the average reward should be more expensive than the average cost.
- Reward cost is a function of the reward's enjoyment and time.

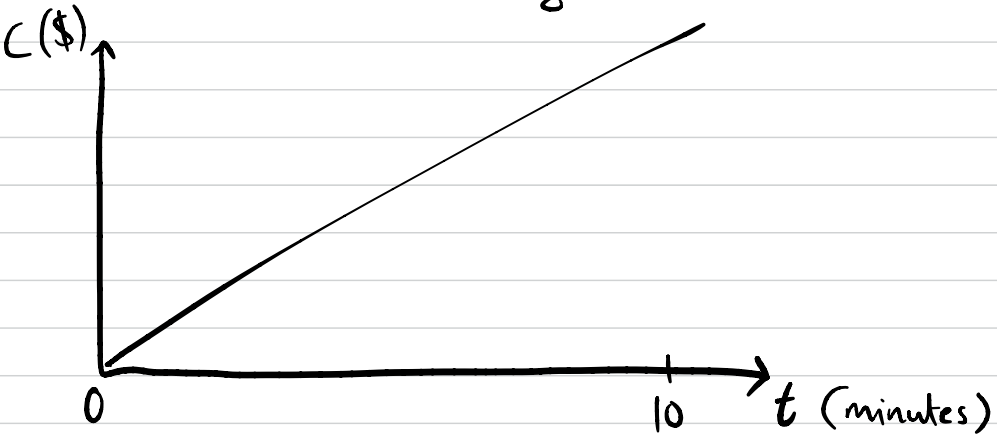
Cost (x = enjoyment , t = time):

↳ Variables are normalised to between 0 and 10 inclusive

- Enjoyment of a reward is a strong incentiviser, and typically we adopt an 'all or nothing' approach to rewards; therefore, enjoyment will be modelled on an exponential function.



- Time is linear again.



$\therefore \quad C \propto e^{x} \quad , \quad C \propto t + c$

$\Rightarrow C = A_1 e^{k_1 x} + A_2 t + k_2$

Axioms

'Best' case : $C(10, 10) = 2500$ e.g. 2.5 hours of gaming

'Average' case $C(5, 5) = 500$ e.g. 30 minutes of watching YouTube

'Worst' case $C(0, 0) = 0$ There is no cost for a reward which takes no time

'Semi-worst' case $C(3, 1) = 100$ e.g. 10 minutes of browsing Snapchat

- Time is a much larger factor than enjoyment.

Weightage : Enjoyment : 35%
Time : 65%.

- To match A_1, k_1, A_2 and k_2 I will use the best and 'semi-worst' case.

$C(x) = A_1 e^{k_1 x}$; $C(3) = 35$, $C(10) = 875$

$\therefore A_1 = 8.809$, $k_1 = 0.460$

$C(t) = A_2 t + k_2$; $C(1) = 65$, $C(10) = 1625$

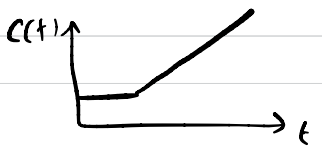
Although $C(0)$ should be 0, I am adding a constant to fit the data from $1 \leq t \leq 10$ better.

$$\begin{aligned} A_2 + k_2 &= 65 \\ 10A_2 + k_2 &= 1625 \\ \hline 9A_2 &= 1560 \\ A_2 &= 173.333 \\ k_2 &= -108.333 \end{aligned}$$

This implies $C(t=0) < 0$ which we cannot allow, thus we will use a piecewise function of $C(t)$

$\therefore A_2 = 173.333$, $k_2 = -108.333$

$\Rightarrow C(t) = \max(30, A_2 t + k_2)$



$$\therefore C(x,t) = 8.809e^{0.46x} + \text{Max}(30, 173.333t - 108.333)$$

This gives an average case $C(5,5) \approx 850$, which is simply too high given the average task payout is 380.

Modified the function to have a slower increase proportional to time.

$$\Rightarrow C(x,t) = 8.809e^{0.46x} + \text{Max}(30, 120t - 75)$$

- This has an average around 600, and a best case around 2000.

Alpha Testing

With a basic version of the app running, I will test it within my life for 1 week and provide thoughts/feedback at the end of each day!

Before starting, I must initialise all routine tasks and rewards, trying to be as exhaustive as possible.

Routine Tasks		P(b,d,t)
• Revision	→ Computing	P(6,4,8) = 558
	→ Economics	P(7,7,8.5) = 716
	→ Physics	P(9,6.5,9) = 948
	→ Maths	P(6,5,8) = 573
• Crimson	→ Revise for SAT	P(10,5,5) = 912
	→ University Research	P(3,4,9) = 532
	→ Create wireframe of school assistant	P(3.5,4,7) = 435
• Youtube	→ Programming Video	P(4,4,3) = 236
	→ Maths Video	P(6,5,4) = 365
• Morning	→ Come down for breakfast by 07:00	P(7,6,2) = 342

Any new tasks will be mentioned in reports.

Available Rewards	$C(x,t)$
30 minutes of YouTube	$C(8,2) = 514$
15 minutes of Snapchat	$C(7,1) = 265$
Waste a free period (50 minutes)	$C(8,4) = 754$
Other	$C(10,2) = 1041$

If any new rewards come up, I will mention them during feedback.

Testing / Feedback

Sunday 3rd March 2024

- When determining the payout and cost of certain tasks, I was usually exaggerating the parameters (very unlikely I placed a parameter under 4). This may simply be because it's hard to judge a parameter without a comparison point (i.e. is the initial comparison point is high, all other payouts/costs are likely to be high).
  - ↳ Provide useful comparison points for each parameter on the 0 → 10 scale.
  - ↳ Use a 1 → 5 scale instead, to reduce cognitive load.

- I also found myself switching parameters after looking at the resultant payout/cost; this should not happen as the price should be dependent on the parameters (not the other way around).
  - ↳ Don't display price until all parameters have been finalised.

- I will test without daily fees and then evaluate
  - ↳ Daily tax rate.

- Added 'listening to music while studying' as a reward; assumed it reduces productivity by 30% => 30% of a 2 hour study session is 36 minutes => cost = $C(8,2) = 514$

- Started testing at 12:59 on Sunday 3rd March 2024.

- When adding work with a deadline, e.g. homework, you only want the damping function to start once the deadline has passed.

E.g. I learn about a task today but I don't want to do it until Tuesday next week. I may still want to add it to today's tasks in case I have some extra time; however I won't if the damping function takes effect immediately.

↳ Add another parameter to a task: date to start damping payout from.

- Task and reward cells can overflow.
  ↳ Re-design cells to make them more 'robust'.
- Clean up setup tasks.
  ↳ Remove 'oneTime' tasks with nextIteration = '-'.
- Add confirmation when user tries to buy reward.

Monday 4th March 2024

- Not all 'rewards' are accounted for, e.g. lunchtime break.
  ↳ Account for all possible rewards in setup, to create a constant outflow of credits.
- Certain tasks may be undervalued. E.g. I want to revise everyday but it gives me a relatively low payout.
  ↳ Reconsider payouts for tasks every few weeks/months

Tuesday 5th March 2024

- Can become overwhelming when there are so many tasks.
  ↳ Implement a 'burnout warning' if someone earns too much/completes too many tasks.
- Certain tasks must be completed no matter the urgency.
  ↳ Different damping functions for different types of tasks.

Goals' payout algorithm

Payout ranges from $500 → $10000

Parameters: Time, Accomplishment

- Time ranges from 1 week to 3 months.
- Accomplishment ranges from 1 (Participate in class) to 6 (get A* in a subject) to 10 (Hero/Heroine in society)

50% 50%

- Time is no longer linear as it is difficult to maintain consistency for many weeks.
- Accomplishment is also non-linear

		Weightage t	Weightage a
P(0, 0) = 0			
P(1, 1) = 500		250	250
P(5, 5) = 5000	Modelling using these	2500	2500
P(10, 10) = 10000		5000	5000

Let $P(t) = Ae^{kt}$

$P(a) = Ae^{ka}$

$P(1) = 250$; $P(10) = 5000$

$\therefore Ae^{k} = 250$ $Ae^{10k} = 5000$

$$\Rightarrow \frac{e^{k}}{e^{10k}} = \frac{250}{5000}$$

$$e^{-9k} = \frac{1}{20}$$

$$k = -\frac{1}{9}\ln\left(\frac{1}{20}\right) \approx 0.333$$

$$A = \frac{250}{e^{k}} \approx 179.21$$

$\therefore P(t, a) = 179.21e^{0.333t} + 179.21e^{0.333a}$

$P(5,5) = 1894$

Too low.

- Model assume $P(x)$ for t and a are linear, since the user linearises them by putting them on a scale from $0 \rightarrow 10$.

$P(1) = 250 \quad ; \quad P(10) = 5000$

$P(x) = Ax + k$

$\therefore \quad A + k = 250$

$-\ 10A + k = 5000$

$-9A = -4750$

$A = 527.777 \qquad k = 250 - A = -277.777$

$\Rightarrow P(t,a) = \text{Max}(0, 527.777(t) - 277.777) + \text{Max}(0, 527.777(a) - 277.777)$

$P(5,5) = 4722$

- Much better...

Progress

Screen to view long term trends