

Corporation manual

Preface

Corporation is an interesting feature of Bitburner. It's the most profitable feature in the game, and it's so overpowered that many BitNodes tend to apply a penalty modifier. However, it's extremely complex and opaque for newbies. There are too many mechanisms that are intertwined with each other, so it's very easy to make a mistake, and that mistake can cripple your corporation. This manual explains all the mechanisms that you need to know to build a successful corporation.

The manual is a bit long and intimidating at first glance, but you don't need to read all the sections below at once. I recommend that you read the first 4 sections. They are the most important sections for newbies. After that, you can read the following sections at your leisure.

This manual is still under ongoing development, I will update it occasionally when I have free time.

Contents

1. Basic gameplay – Term.....	7
1.1 Basic gameplay	7
1.2 Term	8
2. FAQ.....	9
3. Industry – Supply chain	16
3.1 Basic term.....	16
3.2 Criterion.....	16
3.3 Agriculture + Chemical + Tobacco	17
3.4 Other product industries	17
3.5 Conclusion.....	18
4. General advice	19
4.1 Round 1	19
4.2 Round 2	19
4.3 Round 3+	20
5. Generate useful Corporation data	23
6. Unlocks – Upgrade – Research	24
6.1 Unlocks.....	24
6.2 Upgrade.....	24
6.3 Research	27
7. Warehouse.....	30
7.1 Formula	30
7.2 Sample code	30
8. Boost material.....	31
8.1 Division production multiplier.....	31
8.2 Optimizer.....	32
8.3 Solution.....	32
8.3.1 Non-linear constrained optimization library	32

8.3.2 Lagrange multiplier method.....	33
8.4 Proof	34
8.5 Handle low storage space	36
8.6 Sample code	36
9. Division raw production	37
9.1 Definition.....	37
9.1.1 Formula	37
9.1.2 Sample code	38
9.2 Optimizer.....	38
9.3 Sample code	38
10. Office	39
10.1 Basic information.....	39
10.2 Upgrade	40
10.3 Energy and morale	40
10.4 Employee production by job.....	42
10.5 Calculate employee's stat.....	43
10.6 Optimizer.....	44
11. Quality.....	45
11.1 Basic term.....	45
11.2 Material	45
11.3 Product.....	46
12. Smart Supply.....	47
12.1 Logic	47
12.2 Detect warehouse congestion	48
12.3 Sample code	48
13. Wilson Analytics – Advert.....	49
13.1 Awareness and popularity.....	49
13.2 Advert	49
13.3 Wilson Analytics	49

13.4 Optimizer.....	49
13.5 Sample code	51
14. Demand – Competition.....	52
14.1 Usage	52
14.2 Material	52
14.3 Product.....	52
15. Product.....	53
15.1 Overview	53
15.2 Formula	54
15.3 Approximation value of product markup	57
15.4 Sample code	57
16. Optimal selling price – Market-TA2	58
16.1 Market price and markup limit.....	58
16.2 Maximize sales volume.....	58
16.3 Optimal selling price	59
16.4 Sample code	60
17. Financial statement	61
17.1 Total assets	61
17.2 Valuation	61
17.3 Investment offer.....	62
17.4 Dividend	62
17.5 Shares.....	63
18. Miscellany.....	66
18.1 Corporation’s state.....	66
18.2 Import and export.....	66
18.3 Use mathematical library	67
18.3.1 Ceres Solver	67
18.3.2 ALGLIB.....	68
18.4 Noodles trick.....	68

18.5 "sudo.Assist" research	69
18.6 Dummy division.....	69
19. Standard strategy	70
19.1 How to use this section	70
19.2 Round 1	71
19.2.1 Custom Smart Supply.....	71
19.2.2 Without custom Smart Supply	71
19.3 Round 2.....	71
19.3.1 Phase 1	72
19.3.2 Phase 2.....	72
19.4 Round 3+.....	73
19.4.1 Overview	73
19.4.2 Advice	73
20. Advanced strategies.....	78
21. Strategies for other BitNodes	79
22. How to test your strategy efficiently and reliably	80
22.1 Patch the game	80
22.2 Expose Player object.....	81
22.3 Support tools.....	81
22.4 Electron.....	82
22.5 Remove RNG	82
22.6 Reliability	83
22.7 Measure the performance of strategy	84
23. Bitburner in headless mode	85
23.1 JavaScript runtime.....	85
23.2 esbuild.....	85
23.3 Write esbuild script	86
23.4 Sample esbuild script	86
23.5 Non-UI code imports UI code	86

23.6 jsdom.....	87
23.7 Manually patch invalid access.....	89
23.8 Import Blob URL.....	89
23.9 WebSocket.....	90
23.10 Worker thread.....	90
23.11 Load scripts directly.....	91
23.12 Patch Ceres.js.....	92

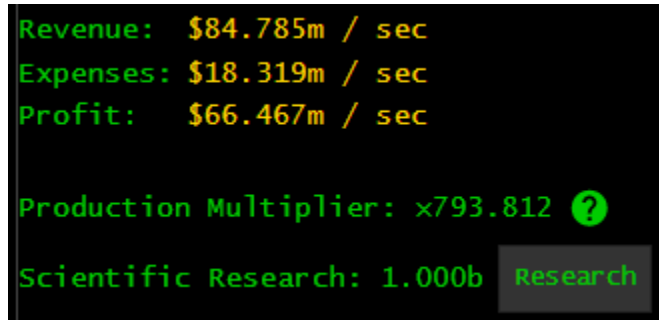
1. Basic gameplay – Term

1.1 Basic gameplay

- There is a corporation-management-handbook.lit on your home server. Read it.
- Go to City Hall in Sector-12 and create a Corporation through the UI if you want. However, you should do everything by scripting.
- You can use seed money when creating a corporation in BN3.
- There are multiple industries that you can expand into. In order to do that, you need to choose an industry and create a division. Agriculture is the best starting industry. Check this [section](#) for details.
- Each division can expand to 6 cities.
- Each industry has different input materials and output materials/products. For example: Agriculture needs Water and Chemicals to produce Plants and Food. The number next to each material is its “coefficient” (You can call it “weight” or “factor” if you want).

0.5 Water + 0.2 Chemicals \Rightarrow 1 Plants + 1 Food

- There is no “offline progress” in corporation. When you go offline, the corporation accumulates bonus time.
- A corporation continuously transitions between 5 states: START \rightarrow PURCHASE \rightarrow PRODUCTION \rightarrow EXPORT \rightarrow SALE \rightarrow START. The action occurs when the state is entered, i.e., when the state is PURCHASE, it means purchasing has just occurred. One cycle (going through one of these transitions) takes 10 seconds. If you have enough bonus time, it takes one second. Check this [part](#) for details.
- Each division has its “division product multiplier”. This multiplier can be increased by buying [boost materials](#): AI Cores, Hardware, Real Estate, and Robots.



Revenue: \$84.785m / sec
Expenses: \$18.319m / sec
Profit: \$66.467m / sec

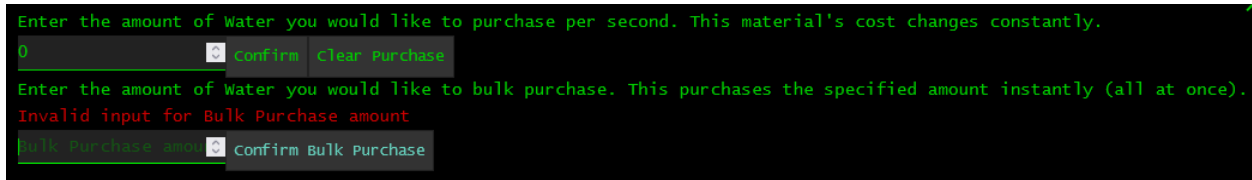
Production Multiplier: x793.812 ?

Scientific Research: 1.000b Research

- You should look around to get familiar with the UI. One confusing thing for newbies is how to setup the “buy value” to buy materials. We have “Purchase” and “Bulk purchase”:
 - Purchase: This is “buy per second” value. For example: In cycle 1, you enter “100”, then in cycle 2, at PURCHASE state, you will have 100*10 units in your inventory. You can buy more

than what your funds allows (and go into debt) with this option. Important note: when you have enough units that you want, you must press “Clear purchase”, otherwise it will buy forever until you run out of storage space.

- Bulk purchase: You buy exactly what you want. Must pay upfront.



- If you want to buy something, write a script to do that. It’s too error-prone to do it manually.
- When you hover your mouse over a warehouse, you will see these numbers. They are not the numbers that I mention in the [standard strategy](#). They are the space that those materials take up in the warehouse. We never use them and only use number of units. For example: if I say “Buy 1000 Hardware”, it means buying 100 units/second for 1 cycle.

Hardware:	144.2
Robots:	11.5
AI Cores:	211.4
Real Estate:	624.8

1.2 Term

- Smart Supply: Automatically buy optimal quantities of input material units.
- Export: Allow export/import materials between divisions.
- Wilson: Wilson Analytics upgrade.
- Market-TA2: Automatically set optimal prices for your output materials/products.
- RP: Research point.

2. FAQ

- *I tried to use your code, but it threw syntax errors.*

I use TypeScript. Follow these steps to transpile my TypeScript code to JavaScript:

- Install Node.js.
 - Clone my repository: <https://github.com/catloversg/bitburner-scripts>
 - Run “npm i” to install the dependencies.
 - Run “npm run build” to build. If you want to use watch mode, run “npm run watch”.
- *Why does your code use some functions that does not exist in NS API?*

I use the latest development build.

- *What is a corporation good for?*

Generating ridiculously massive income. With this income, you can buy whatever you want, e.g., augmentations or bribing factions for reputations.

- *How many investment rounds should I take?*

There are 4 rounds, and you should take all of them. Investment funds greatly boost your corporation development. Your corporation won't be able to reach its full potential in a reasonable time without them.

- *Investors take too many shares. Can I buy them back later?*

No.

- *The government takes too many shares when I use the “Seed money” option. Can I buy them back later?*

No.

- *Why can I not buy back anything? It's ridiculous.*

You can only buy back shares that were issued. The shares that were owned by the government (when we use seed money) and investors cannot be bought back.

One thing to remember: this is a game, not real life. In this game, you always have total control over your corporation. When your profit hits exponential growth, the shares percentage means nothing. If your corporation's profit is 1e90/s and you only have 1% shares, you still have 1e88/s.

- *My corporation generates profit. Why does my money not increase?*

Go public and set dividend.

- *How many shares should I issue?*

0

- *Why is my “earnings as a shareholder” lower than my calculation (“Dividends per share” * “Owned Stock Shares”)?*

You have to pay tax. ShadyAccounting and GovernmentPartnership reduce tax. Check this [part](#) for details.

- *All corporation API requires too much RAM. How do I deal with it?*

Earn money by any normal means: hacking, committing crimes, cheating in casino, etc. There is also another way:

- Follow the guide on round 1, but at the end, do not accept the investment offer.
- Go public immediately.
- Sell all your shares immediately.
- Sell CEO position and start a new corporation.

Even with my worse strategy (without custom Smart Supply), you can still get ~600b if you go public and sell shares fast enough. It's ~1t if you use the standard strategy.

Also note that you don't have to do everything in one script. You can make smaller scripts that do less and use fewer APIs to keep the RAM usage down, and use `run()` to chain them together.

- *Why can I not create a corporation with the government's seed money (“Use seed money”)?*

That option is only available in BN3.

- *Can I sell my corporation via API?*

No.

- *Can I transfer my (personal) money to the corporation's funds?*

No. However, with SF9, you can sell hashes for corporation's funds or RP.

- *Why can I not bribe factions for reputations? What's the exchange rate?*

Your corporation's valuation must be at least 100e12 to bribe. Exchange rate: 1e9/reputation.

- *What's the maximum number of divisions?*

In BN3, it's 20.

- *I follow “Quick start” but the offer is much lower than your number, not even reaches 80% your number.*

Check these things:

- Make sure that you use the right numbers and you are in BN3.
- If you use custom Smart Supply script, make sure that your script works properly.
- *Why does my division not produce anything?*

Check these things:

- Have at least 1 employee in Operations or Engineer
- Have enough storage space. Warehouse congestion is a common problem.
- *Which industry should I focus on?*

Check this [section](#).

- *Should I create more divisions for the same industry? For example: multiple Agriculture.*

No, focus your funds on one division for each industry.

- *Which “feature” (Export, Smart Supply, etc.) should I unlock?*

Check this [part](#).

- *Which upgrade should I buy?*

Check this [part](#).

- *Which research should I buy?*

Check this [part](#).

- *I unlocked “Smart Supply”, but it does not work or its behavior is weird.*

Check these things:

- You have to enable it if you unlock it after purchasing a warehouse.
- Always choose “Use leftovers”. “Use imported” is only useful in special cases.
- *How do I implement a custom Smart Supply script?*

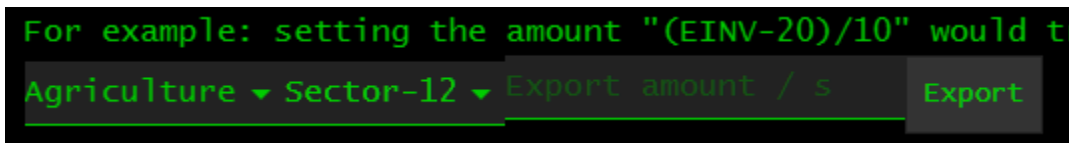
Check this [section](#).

- *How do I setup the quantity of exported materials?*

Specify an export string.

- *What is export string?*

- It's the string that you put into this:



- The optimal export string is $(IPROD+IINV/10)*(-1)$. Check this [part](#) for details.
- *Is research's benefit shared between different divisions?*

It's shared if those divisions are in the same industry. However, the RP pool is not shared.

- *Should I expand to all 6 cities?*

Yes. In fact, you must do that for maximum efficiency. Check this [part](#).

- *What are boost materials?*

They are the materials that boost [division production multiplier](#). There are 4 boost materials: AI Cores, Hardware, Real Estate, and Robots.

- *How many boost materials that I should buy?*

Check this [optimizer](#).

- *Why does [General advice](#) section tell me to use API to upgrade office size?*

API (upgradeOfficeSize) gives you granular control over office size. You cannot do that through UI.

- *Why does setAutoJobAssignment not take effect immediately?*

It only takes effect in next cycle's START state.

- *Why does energy and morale matter?*

They are used for calculating employeeProductionByJob, then that property is used for calculating other things: RP, material's quality, product's stats, division raw production and material/product's MaxSalesVolume.

- *What do Interns do?*

Don't bother with that job. Its only purpose is to maintain energy and morale. A tea/party script can do it for you, and it is very simple to implement.

- *Everybody tells me to use 1/9 as Intern ratio, but when I use it, energy and morale still drop.*

You can only use that ratio when your corporation works fine (funds > 0 or profit > 0). If it does not, use 1/6.

- *Are there any other ways to maintain these 2 stats?*

There are researches for that. However, you should never buy them, it's always better spending your RP elsewhere or just stock up on RP.

- *Buying tea and throwing parties cost me too much money. Why are they so expensive?*

Tea and parties are cheap. If your budget is so low that they cost you too much money, it means you wasted too much of your funds.

- *How much money should I spend to throw parties? How often should I buy tea / throw party?*

Check this [part](#).

- *What is optimal ratio for office's employees?*

Check this [part](#) and this [part](#).

- *How do I know if the qualities of my input materials are too low and need to be improved?*

Check this [part](#).

- *What are Awareness and Popularity?*

Check this [part](#).

- *Should I buy Dream Sense?*

No. Check this [part](#) for the reason.

- *Is Wilson retroactive?*

No.

- *Does that mean I should buy Wilson as soon as possible? If yes, then why don't you buy any Wilson in round 1 and 2?*

You should buy Wilson as soon as possible, but not too soon. Round 1 and 2 are those cases. Check this [part](#) for details.

- *What are Demand and Competition?*

Check this [part](#).

- *How much should I spend for "Design investment" and "Marketing investment" when I create new product? How do they affect the product?*

They are not too important. It's fine to spend 1% of your current funds for them. Check this [section](#) for details.

- *Should I buy Market-TA1?*

No, wait for Market-TA2. Market-TA1 is useless on its own.

- *When should I buy Market-TA2?*

As soon as possible, it greatly increases your profit because it can find the optimal price. However, that research is expensive, it costs a total of 75e3 RP (Hi-Tech R&D Laboratory + Market-TA1 + Market-TA2). Depleting the entire RP pool extremely degrades your product rating, so I recommend saving up 150e3 RP before buying it. Depleting half of the RP pool is acceptable, considering the positive effect of Market-TA2.

- *What is the difference between Market-TA1 and Market-TA2?*

Market price:

```
Tobacco-000: 125.014 (-0.019/s)
Effective rating: 2.098m
Est. Production Cost: $3.056k
Est. Market Price: $15.280k
Discontinue Sell (12.520/0.000) @$15.280k
```

Market-TA1: set a price that ensures that you can sell all produced goods in storage.

```
Tobacco-000: 125.527 (-0.019/s)
Effective rating: 2.098m
Est. Production Cost: $3.057k
Est. Market Price: $15.284k
Discontinue Sell (12.571/0.000) @$290.283m
```

Market-TA2: set the highest possible price that ensures that you can sell all produced goods in storage.

```
Tobacco-000: 125.338 (-0.019/s)
Effective rating: 2.098m
Est. Production Cost: $3.026k
Est. Market Price: $15.131k
Discontinue Sell (12.552/0.000) @$46.743b
```

- *I bought Market-TA2, but it does not set optimal price for me.*

You have to enable it.

```
the 'MP' variable to set a dynamically changing price that depends on
you set it to 'MP*5' then it will always be sold at five times the est
Confirm ☒ Set for all cities ☐ Market-TA.I ☒ Market-TA.II
```

- *Is there a workaround for Market-TA2? Waiting for RP takes too long.*

Yes, you can reimplement Market-TA2. Implementing a custom Market-TA2 script is the best optimization in round 3+. Check this [section](#) to see how to do it.

- *How do I discard materials/products?*

Set selling price to 0.

- *What is a dummy division?*

Check this [part](#).

- *Can I skip Chemicals in round 2 and invest all funds in Agriculture?*

No. Without a Chemical division, the quality of your Agriculture's output materials will be too low, and you cannot sell all those low-quality materials at good price.

- *How do I split funds between things like office, warehouse, advert, boost materials, upgrades, etc.*

It's hard to give a short answer. You should read other sections for analyses of multiple different aspects of Corporation. Different phases require different considerations. Check this [section](#) for details.

- *How should I split funds between Smart Storage and warehouse upgrade to optimize storage space? How should I split funds between X and Y to optimize Z?*

Use the optimizer. Theoretically, you can employ fancy mathematical analysis to find an optimal solution. We do this with the [boost materials optimizer](#). However, most problems are solvable by either brute force (check [corporationOptimizer.ts](#)) or approximation (check this [part](#)).

- Do I need to implement all optimizers? Are they mandatory?

No. As long as you follow the most important advices, you don't need the optimizer(s).

- Do I have to strictly follow all the numbers in the [standard strategy](#)?

No. It is only a recommended strategy, you can devise your own strategy by following these [general advices](#).

3. Industry – Supply chain

3.1 Basic term

- Industry consumes input materials and produces output materials/products. Most industries produce either materials or products, some industries produce both of them.
 - Material industries are simple to bootstrap. We use them as starting/support industries.
 - Product industries are hard to bootstrap, but they generate massive profits.
- Each industry has different values of "factor":
 - AICoreFactor, HardwareFactor, RealEstateFactor, RobotFactor: boost material's coefficient. They are used for calculating [division production multiplier](#).
 - ScienceFactor:
 - ❖ With material industry: affect the output material's [quality](#).
 - ❖ With product industry: affect the output product's [markup and rating](#).
 - AdvertisingFactor: affect the number of units that we can sell ([AdvertFactor](#)).
- "Support industry" is the industry that provides high-quality input materials for other industries.
- "Endgame" is the very late phase after the division passes the exponential growth. In this phase, the new product is only marginally better than the old products, and the profit increases slowly. For example, the endgame of Tobacco is when the profit reaches $\sim 1e98/s$.

3.2 Criterion

In order to choose good industries for our supply chain, we must consider many criteria:

- Good synergy between industries. Industries must be able to support each other with the "Export" feature. After the reworking in 2.3, the quality of input material becomes crucial, and importing high-quality materials from other industries is mandatory.
- Must have an easy-to-start material industry for early rounds. This industry must have good combinations of boost materials' coefficients and boost materials' sizes. This ensures a high division product multiplier that is the most crucial factor in early rounds. In 4 boost materials, Real Estate (material, not industry) should be noticed because of its tiny size.
- High ScienceFactor is a bonus for the support industry because its output material's quality can be boosted without much investment in early rounds.
- Must have a highly profitable product industry for late phases.
 - It's preferred to require material from the starting industry rather than to expand into another support industry.
 - High ScienceFactor. This is necessary for a high product's rating.
 - High AdvertisingFactor. This means our product's selling price can be boosted greatly by Wilson and Advert. This is the most important factor for the endgame. AdvertisingFactor

scales tremendously well when you reach high advert bonuses (Awareness and Popularity). These bonuses are capped at $\sim 1.7977e308$.

- Any worthy industry must have at least 1 input material. Spring Water is the only industry that does not need any input material. It may look convenient at first glance, but it's a trap for newbies. Having no input material means the quality/rating of the output material/product will always be capped at the square root of its maximum value.

3.3 Agriculture + Chemical + Tobacco

- Agriculture is inarguably the best starting industry. Its `realEstateFactor` is highest among all industries (0.72), and real estate's size is tiny (0.005). It means we can stock up on a huge number of real estate and increase the division product multiplier to an extremely high value in early rounds.
- Chemical is the best support industry if we choose Agriculture as the starting industry. It has the highest `ScienceFactor` among all material industries (0.75). As you can see in round 2's strategy, we only need to invest a minimal budget in the Chemical division and it can still adequately boost the Agriculture division (after waiting for some RP).
- Agriculture and Chemical have great synergy. Agriculture needs `Chemicals` from Chemical and produces `Plants`. Chemical needs `Plants` from Agriculture and produces `Chemicals`.
- Tobacco is an excellent product industry:
 - It requires only `Plants`, and `Plants` come from Agriculture. Therefore, the quality of input material is not a problem with this industry if we choose Agriculture as the starting industry.
 - High `ScienceFactor` (0.75), only below Pharmaceutical.
 - High `AdvertisingFactor` (0.2), only below Restaurant and Real Estate.

3.4 Other product industries

All other product industries always have some kind of flaws in them and not be as versatile as Tobacco. For example:

- Pharmaceutical has the highest `ScienceFactor` (0.8) but a low `AdvertisingFactor` (0.16).
 - Its products are potentially better than Tobacco's ones. This sounds good in theory, but not in practice. Pharmaceutical requires `Chemicals`. `Chemicals` can only be produced by the Chemical industry, and that industry has bad production capability. The early products' effective rating will be low because the Chemical division cannot produce enough high-quality material units. In the end, the offers of round 3 and round 4 are much lower than Tobacco's ones.
 - It is worse than Tobacco in the endgame.
- Healthcare has the same `ScienceFactor` as Tobacco (0.75) but the lowest `AdvertisingFactor` (0.11).
 - It requires 4 input materials instead of 1 or 2. Those materials include `Robots` with a coefficient of 10 (the highest value of all industries), and the size of `Robots` is the biggest

among boost materials. This means the required materials take up huge space in the warehouse.

- This industry is the worst one in the endgame due to its extremely low AdvertisingFactor.
- Restaurant has the highest AdvertisingFactor (0.25) but very low ScienceFactor (0.12).
 - Its products are significantly worse than Tobacco's ones. It takes too long to hit the exponential growth (due to worse products) and surpass Tobacco in profit.
 - It's fine to expand into this industry after Tobacco to have a better endgame.
- Real Estate has the same AdvertisingFactor as Restaurant but the lowest ScienceFactor (0.05). It requires 4 input materials instead of 1 or 2. This industry has the same potential and problem as Restaurant with its high AdvertisingFactor and low ScienceFactor. The only difference is that its ScienceFactor is even lower than the Restaurant's one.

3.5 Conclusion

[Agriculture + Chemical + Tobacco is the most balanced supply chain](#). It's very easy to bootstrap in early rounds and reaches high profit in late phases for practical purposes.

4. General advice

Attention:

- In this section, you will see some new terms. They will be explained in later sections. If you see a link, you should follow it and read the relevant section.
- All the optimizers that I mention in this section are optional. I will show you how to write them in later sections. If you think that they are too complicated, you can skip them and spread the funds/employee equally across upgrades/non-R&D jobs.
- Generally, these advices are applicable for all corporation-viable BitNodes. However, there are some BitNodes that apply harsh penalty on [valuation](#). In these cases, you need to adapt these advices to the penalized valuation. Check this [section](#) to see how to do that.

4.1 Round 1

- Create Agriculture division, expand to 6 cities and buy 6 warehouses.
- Use API (upgradeOfficeSize) to upgrade office size from 3 to 4. Set 4 employees to R&D and wait until RP is at least 55. Switch to Operations (1) + Engineer (1) + Business (1) + Management (1) before buying boost materials.
- Make sure that your employees' energy and morale are at maximum value. This is always mandatory, not just in round 1.
- Write a custom "Smart Supply" script. If you skip this step, buy "Smart Supply" feature. "Smart Supply" costs 25b, and 25b is huge in round 1.
- There are not many things else to do in this round. The budget is too low, so you can skip all these things:
 - Expand into other industries.
 - Buy corporation's upgrades (except Smart Storage).
- Use remaining funds to buy these upgrades:
 - Only focus on Smart Storage and warehouse upgrade. Run optimizer to find the best combination of them.
 - Buy 2 Advert levels.
- After that, find the optimal quantities of boost materials and buy them. Do not use "Bulk Purchase", it requires paying upfront. Buying boost materials per second does not need funds because you can go into debt.

4.2 Round 2

- Buy "Export".
- Upgrade Agriculture division:
 - Find a good number for office size. 8 is the optimal size.

- Buy a couple of Advert levels. Advert level 8 is enough for most cases.
- Create Chemical division:
 - Expanding into Chemical industry is mandatory. Without high-quality Chemicals (material) from Chemical division, output materials in Agriculture will be low-quality, and low-quality materials cannot be sold well.
 - Chemical division is a support division, so don't invest much funds on it. Don't waste funds on its Office/Advert upgrades.
 - Chemical industry has low boost materials' coefficients, so you should only buy very small number of warehouse upgrade for it. On the other hand, you should not skip Chemical's warehouse upgrade entirely. You still need Chemical division produces an acceptable amount of high-quality Chemical; otherwise, the quality of Chemical used in PRODUCTION state of Agriculture division is reduced too much due to "dilution" in PURCHASE state. For this purpose, 1 warehouse upgrade is enough.
- Run optimizer to find the best combination of Smart Storage, Smart Factories, warehouse upgrade (Agriculture).
- Waiting for RP is mandatory in this round. It serves 2 purposes:
 - Raising RP in Chemical. High RP means high-quality Chemicals (material).
 - Raising RP in Agriculture. High RP means high-quality Plants.
 - ❖ High-quality Plants can be exported to Chemical to create a loop of quality-enhancing process.
 - ❖ High-quality Plants can be sold easier (higher [MaxSalesVolume](#)). With your limited budget, you can only increase MaxSalesVolume by buying Advert and improving quality of output material. Buying Advert costs money, but waiting for RP is free (except your time).
- Waiting for 700RP/390RP in Agriculture/Chemical division respectively is enough. You can wait for more if you want.

4.3 Round 3+

- Create Tobacco division in round 3 and set up export route for Plants from Agriculture to Tobacco. This is the optimal product division in this phase.
- The basic game loop that you need to do in round 3+ is simple:
 - Buy research.
 - Buy Wilson and Advert.
 - Continuously develop new product.
 - Upgrade product division and buy corporation's upgrades.
 - Upgrade support divisions.
- These are the most important things that you need to focus on:
 - Buy Wilson and Advert.
 - Continuously develop new product.

- Get high-quality input materials from support division(s).
- Stock up on RP.
- Get Market-TA2 as soon as possible.
- Wilson and Advert are extremely important, they are the main factors that make product extremely profitable.
 - Check this [section](#) for details about the mechanism of Wilson and Advert. You should read the parts that I highlight carefully even if you don't want to write the optimizer.
 - After accepting the investment offer, you will have large budget. In this case, run the optimizer to calculate the optimal numbers of Wilson and Advert upgrades that you should buy.
 - When you continuously improve the product division and buy the upgrades with small budget (profit of a few cycles):
 - ❖ Buy Wilson if you can afford it.
 - ❖ After that, use at least 20% of current funds to buy Advert.
 - Stop buying Wilson and Advert when product division's awareness/popularity reaches max value.
- [Product](#) is the center of round 3+.
- Product's rating is limited to product's effective rating by quality of input materials. You must make sure that support divisions produce enough high-quality materials for product division. For material divisions, increasing RP is the best way to improve the quality in early rounds. However, in round 3+, the most important factor is EngineerProduction, so you must prioritize the "Engineer" job over other jobs. It's especially true for Agriculture due to its mediocre ScienceFactor. Check this [section](#) for details. On the other hand, it's fine to invest minimally in support divisions, as long as they produce enough high-quality materials.
- RP is important for product's rating. Do not deplete RP pool, especially right before completing new product.
- Check this [section](#) for more advices about researches.
- Office setup is important to efficiently develop new product. There are multiple setups for different purposes:
 - Raw production: This setup prioritizes production capability. For support divisions, you should "combine" the prioritization of "Engineer" job with this setup.
 - Progress: This setup prioritizes development speed of new product. It's best for product division in round 3 and round 4. The product's development speed is very low in these 2 rounds (especially in round 3). By focusing on development speed and get better product sooner, you can reach the point that you can get an adequate offer faster. That's the reason 1P/2P setup (more about it in the [standard strategy](#)) is the optimal setup in my strategy.
 - Profit. This setup prioritizes profit. It's best at the end of a round, before accepting offer.
 - Profit-Progress. This setup provides good balance between current profit and development speed of new product. It's best after accepting last offer.

- All the setups above can be calculated by the optimizer in this [part](#). However, that optimizer is hard to implement correctly and efficiently. If you are a newbie, you can skip this optimization and spread employees equally across non-R&D jobs. It's not optimal, but you can still reach the endgame after couple hours if you follow other important advices. You can check this [part](#) for the sample ratios. Do not blindly use them.
- There are 2 types of office: main office and support office. Main office is where you develop new product. Support office is where you assign a large number of employees to R&D job to increase RP. The most common setup is 1 main office and 5 support offices. The most important office is the main office, its budget must be much higher than support offices' budget.
- The purpose of investment offer is to get large funds and quickly grow the corporation. Better product brings more profit, and higher profit means higher offer. However, it takes a long time to develop early product(s). Sometimes, spending more time to develop better product before accepting offer can harm your overall growth. You must find a good number of products to develop before accepting offer.
- Miscellaneous advices:
 - Buy tea / throw party every cycle.
 - DesignInvestment and AdvertisingInvestment scale very badly (the exponent is 0.1). It's fine to spend 1% of your current funds for them.
 - Create [dummy divisions](#) to get higher offer.
 - Prioritize Tobacco division over Chemical division when setting up export routes for Plants. [Export](#) route is FIFO.

5. Generate useful Corporation data

- There are multiple data that you either cannot access through NS API or have to access it through inconvenient API. You can generate raw data files based on those data and access them for the sake of convenience. Some people may call that cheating, but in my opinion, it's not, they are just constant data.
- Sample code:

[dataGenerator.ts](#)

6. Unlocks – Upgrade – Research

6.1 Unlocks

Name	Price	Description
Export	20e9	Allow exporting material between different divisions. Extremely important. Buy it at the start of round 2.
Smart Supply	25e9	Enable “Smart Supply” feature. Only buy it if you don’t implement your custom Smart Supply script.
Market Research - Demand	5e9	Grant access to Demand data. You need it to implement a custom Market-TA2 script.
Market Data - Competition	5e9	Grant access to Competition data. You need it to implement a custom Market-TA2 script.
VeChain	10e9	View more statistics about Corporation. Useless.
Shady Accounting	500e12	Reduce DividendTax by 0.05
Government Partnership	2e15	Reduce DividendTax by 0.1

6.2 Upgrade

- Each upgrade has different BasePrice, PriceMult, Benefit.
- Most upgrades affect all divisions.
- There are 3 special upgrades. These upgrades only affect its division and have different formulas for cost/benefits.
 - Warehouse. Check this [part](#).
 - Office. Check this [part](#).
 - Advert. Check this [part](#).
- Normal upgrade’s formulas:
 - Upgrade cost:

$$UpgradeCost = BasePrice * PriceMult^{CurrentLevel}$$

- Upgrade cost from level 0 to level n:

$$(UpgradeCost)_{From\ 0\ to\ n} = \sum_{k=0}^{n-1} BasePrice * PriceMult^k$$

≡

$$(UpgradeCost)_{From\ 0\ to\ n} = BasePrice * \left(\frac{1 - PriceMult^n}{1 - PriceMult} \right)$$

≡

$$(UpgradeCost)_{From\ 0\ to\ n} = BasePrice * \left(\frac{PriceMult^n - 1}{PriceMult - 1} \right)$$

- Upgrade cost level a to level b:

$$(UpgradeCost)_{From\ a\ to\ b} = \sum_{k=0}^{b-1} BasePrice * PriceMult^k - \sum_{k=0}^{a-1} BasePrice * PriceMult^k$$

≡

$$(UpgradeCost)_{From\ a\ to\ b} = BasePrice * \left(\frac{PriceMult^b - 1}{PriceMult - 1} \right) - BasePrice * \left(\frac{PriceMult^a - 1}{PriceMult - 1} \right)$$

≡

$$(UpgradeCost)_{From\ a\ to\ b} = BasePrice * \left(\frac{PriceMult^b - PriceMult^a}{PriceMult - 1} \right)$$

- Maximum upgrade level with a given MaxCost:

$$MaxUpgradeLevel = \log_{PriceMult} \left(MaxCost * \frac{PriceMult - 1}{BasePrice} + (PriceMult)^{CurrentLevel} \right)$$

- Benefit:

- ❖ All benefits are multipliers. BaseBenefit is 1.
- ❖ The only exception is DreamSense. Its benefit is raw value, its BaseBenefit is 0.

$$Benefit = BaseBenefit + Benefit * CurrentLevel$$

- Normal upgrades:

Name	Base price	Price multiplier	Benefit	Type
SmartFactories	2e9	1.06	0.03	Production
SmartStorage	2e9	1.06	0.1	Storage
DreamSense	4e9	1.1	0.001	Awareness/Popularity

WilsonAnalytics	4e9	2	0.005	Advert's benefits
NuoptimalNootropicInjectorImplants	1e9	1.06	0.1	Employee's creativity
SpeechProcessorImplants	1e9	1.06	0.1	Employee's charisma
NeuralAccelerators	1e9	1.06	0.1	Employee's intelligence
FocusWires	1e9	1.06	0.1	Employee's efficiency
ABCSalesBots	1e9	1.07	0.01	Sales
ProjectInsight	5e9	1.07	0.05	RP

- Special upgrades:

Name	Base price	Price multiplier	Type
Warehouse	1e9	1.07	Storage
Office	4e9	1.09	Office's size
Advert	1e9	1.06	Awareness/Popularity

- Advices:
 - DreamSense is useless. Never buy it.
 - Round 1:
 - ❖ SmartStorage and Warehouse are the most important upgrades in this round.
 - ❖ Only buy 1 or 2 Advert level(s).
 - Round 2:
 - ❖ SmartFactories, SmartStorage and Warehouse are the most important upgrades in this round.
 - ❖ Only buy 1 Office level and a couple of Advert levels for Agriculture division.
 - ❖ Do not buy Office/Advert for Chemical division.
 - Check this [section](#) for more advices, especially for round 3+.
- Sample code:

[corporationFormulas.ts](#): getUpgradeCost()

[corporationFormulas.ts](#): getMaxAffordableUpgradeLevel()

[corporationFormulas.ts](#): getUpgradeBenefit()

6.3 Research

- Each research has a set of multipliers. For example: sciResearchMult, productionMult, etc.
- Benefit of research type is the product of all research's multiplier of the same type.

Type	Research	Multiplier	Effect
advertisingMult	No research	1	Advert's benefits
employeeChaMult	CPH4 Injections	1.1	Employee's charisma
employeeCreMult	CPH4 Injections	1.1	Employee's creativity
employeeEffMult	CPH4 Injections, Overclock	1.1*1.25	Employee's efficiency
employeeIntMult	CPH4 Injections, Overclock	1.1*1.25	Employee's intelligence
productionMult	Drones - Assembly, Self-Correcting Assemblers	1.2*1.1	Production
productProductionMult	uPgrade: Fulcrum	1.05	Product's production
salesMult	No research	1	Sales
sciResearchMult	Hi-Tech R&D Laboratory	1.1	RP
storageMult	Drones - Transport	1.5	Storage

- Research list:

Name	Cost	Description
Hi-Tech R&D Laboratory	5000	Top priority. Increase RP gain rate. It is the prerequisite of all other researches.
Market-TA.I	20000	Useless. It is the prerequisite of Market-TA.II.
Market-TA.II	50000	Top priority if you don't write custom script. Check this section to see how to write custom script.
Automatic Drug Administration	10000	It is the prerequisite of Go-Juice and CPH4 Injections.
Go-Juice	25000	Useful. Increase maximum energy.

CPH4 Injections	25000	Useful. Increase employee's stats.
Overclock	15000	Useful. Increase employee's stats. It is the prerequisite of Sti.mu.
Sti.mu	30000	Useful. Increase maximum morale.
Drones	5000	It is the prerequisite of Drones - Assembly and Drones - Transport.
Drones - Assembly	25000	Useful. Increase all productions.
Drones - Transport	30000	Useful. Increase warehouse's storage space.
Self-Correcting Assemblers	25000	Useful. Increase all productions.
uPgrade: Fulcrum	10000	Useful. Increase product's production.
uPgrade: Capacity.I	20000	Not useful. The cost is too high for its mediocre benefit. Increase maximum number of products by 1 (from 3 to 4).
uPgrade: Capacity.II	30000	Not useful. The cost is too high for its mediocre benefit. Increase maximum number of products by 1 (from 4 to 5).
uPgrade: Dashboard	5000	Useless.
AutoBrew	12000	Useless.
AutoPartyManager	15000	Useless.
HRBuddy-Recruitment	15000	Useless.
HRBuddy-Training	20000	Useless.

- Advices:
 - Do not deplete entire RP pool to buy research. You should only buy research if it costs less than half of the RP pool. Personally, my conditions for buying researches are:
 - ❖ For energy/morale and employee's stats: if it costs less than 20% of RP pool.
 - ❖ For production: if it costs less than 10% of RP pool.
 - If you don't have a custom Market-TA2 script, you must prioritize Market-TA1 and Market-TA2. Market-TA1 is useless, the only reason to buy it is because it's the prerequisite of Market-TA2. If you buy them, you should stock up on RP and buy them together. However, I

recommend implementing a custom Market-TA2 script as soon as possible. Market-TA1 and Market-TA2 cost 70000 RP, that's a huge number of RP at the start of round 3+. [Implementing a custom Market-TA2 script is the best optimization in round 3+](#).

- After that, you should prioritize researches for higher maximum energy/morale and employee's stats over production. Researches for production are nice to have, but it's much less important than energy/morale/employee's stats.
- My research order for higher maximum energy/morale and employee's stats: Overclock → Sti.mu → Automatic Drug Administration → Go-Juice → CPH4 Injections.
- Do not buy these useless researches:
 - ❖ uPgrade: Dashboard
 - ❖ AutoBrew
 - ❖ AutoPartyManager
 - ❖ HRBuddy-Recruitment
 - ❖ HRBuddy-Training
- In most cases, uPgrade: Capacity.I and uPgrade: Capacity.II are useless. New products are usually much better than the old ones, so there is no point in increasing the maximum number of products. The only exception is when you reach the endgame. In the endgame, new products are only marginally better than the old ones, so having more product slots may be beneficial. However, even in the endgame, those researches may do more harm than good. In the endgame, the warehouse's size and high-quality input materials are serious bottlenecks. Having more product slots means that you need more free space in the warehouse and more units of input materials. In some cases, increasing product slots actually reduces the overall profit. You need to fine-tune it per use case.
- You can exchange hashes for RP if you have SF9. This number of RP is added to all divisions.
- RP gain rate:
 - RP is increased in 4 states: PURCHASE, PRODUCTION, EXPORT and SALE.
 - RP gain per city per state:
$$RnDProduction = office.employeeProductionByJob["Research \& Development"]$$
$$RPGain = 0.004 * (RnDProduction)^{0.5} * UpgradeMultiplier * ResearchMultiplier$$
 - Industry's ScienceFactor does not affect RP gain rate.
 - Sample code:

[corporationUtils.ts](#): `getResearchPointGainRate()`

7. Warehouse

7.1 Formula

- Warehouse starts at level 1 after being bought. The initial price is 5e9.
- BasePrice in the following formulas is the upgrade's base price (1e9), not the initial price above.
- Warehouse upgrade cost: its formula is a bit different from other upgrades (the exponent is (CurrentLevel+1) instead of (CurrentLevel)):

$$UpgradeCost = BasePrice * 1.07^{CurrentLevel+1}$$

- Upgrade cost for buying from level 1 to level n:

$$(UpgradeCost)_{From\ 1\ to\ n} = \sum_{k=2}^n BasePrice * 1.07^k$$

≡

$$(UpgradeCost)_{From\ 1\ to\ n} = BasePrice * \left(\frac{1.07^{n+1} - 1.07^2}{0.07} \right)$$

- Upgrade cost for buying from level a to level b:

$$(UpgradeCost)_{From\ a\ to\ b} = BasePrice * \left(\frac{1.07^{b+1} - 1.07^{a+1}}{0.07} \right)$$

- Maximum level with a given MaxCost:

$$MaxLevel = \left(\log_{1.07} \left(MaxCost * \frac{0.07}{BasePrice} + 1.07^{CurrentLevel+1} \right) \right) - 1$$

- Warehouse size:
 - Upgrade multiplier: multiplier from Smart Storage.
 - Research multiplier: multiplier from researches.

$$WarehouseSize = WarehouseLevel * 100 * UpgradeMultiplier * ResearchMultiplier$$

7.2 Sample code

[corporationFormulas.ts](#): `getUpgradeWarehouseCost()`

[corporationFormulas.ts](#): `getMaxAffordableWarehouseLevel()`

[corporationFormulas.ts](#): `getWarehouseSize()`

8. Boost material

8.1 Division production multiplier

- Each industry has a different set of boost material's coefficients. For example:
 - Agriculture:
 - ❖ AI Cores: 0.3
 - ❖ Hardware: 0.2
 - ❖ Real Estate: 0.72
 - ❖ Robots: 0.3
 - Chemical:
 - ❖ AI Cores: 0.2
 - ❖ Hardware: 0.2
 - ❖ Real Estate: 0.25
 - ❖ Robots: 0.25
 - Tobacco:
 - ❖ AI Cores: 0.15
 - ❖ Hardware: 0.15
 - ❖ Real Estate: 0.15
 - ❖ Robots: 0.25
- The division production multiplier is used for calculating [division raw production](#) in the PRODUCTION state. It's sum of each warehouse's cityMult, and cityMult is calculated by combining the quantity of each boost material with the boost material's coefficient.
- This multiplier is this.productionMult in Division.ts.

```
calculateProductionFactors(): void {
  let multSum = 0;
  for (const warehouse of getRecordValues(this.warehouses)) {
    const materials = warehouse.materials;

    const cityMult =
      Math.pow(0.002 * materials["Real Estate"].stored + 1,
this.realEstateFactor) *
      Math.pow(0.002 * materials.Hardware.stored + 1, this.hardwareFactor) *
      Math.pow(0.002 * materials.Robots.stored + 1, this.robotFactor) *
      Math.pow(0.002 * materials["AI Cores"].stored + 1, this.aiCoreFactor);
    multSum += Math.pow(cityMult, 0.73);
  }

  multSum < 1 ? (this.productionMult = 1) : (this.productionMult = multSum);
}
```

- This is the reason we must expand to all 6 cities. More cities → Higher `this.productionMult` → Higher raw production → More produced materials/products → Higher profit per city → Higher total profit.
- Expanding to 6 cities means `this.productionMult` is multiplied by 6, and we have 6 cities, so effectively production is multiplied by 36. This is not exactly true because there are more things that affect the raw production value of each city, but `x36` can be seen as a rough estimate of benefit, especially in early rounds. In those rounds, division production multiplier is the most important thing.

8.2 Optimizer

- In order to increase `this.productionMult`, we need to buy boost materials. The problem is how much we should buy each material, given a specific constraint of storage space.
- Each boost material has a coefficient (“factor” in source code) and a base size (size for 1 unit in storage).
- Let’s define:
 - 4 coefficients: c_1, c_2, c_3, c_4
 - 4 base sizes: s_1, s_2, s_3, s_4
 - Quantities of each boost materials: x, y, z, w
- Assuming same warehouse setup in all cities, the division production multiplier is:

$$F(x, y, z, w) = \sum_{i=1}^6 ((1 + 0.002 * x)^{c_1} * (1 + 0.002 * y)^{c_2} * (1 + 0.002 * z)^{c_3} * (1 + 0.002 * w)^{c_4})^{0.73}$$

- In order to find the maximum of the function above, we can find the maximum of this function:

$$F(x, y, z, w) = (1 + 0.002 * x)^{c_1} * (1 + 0.002 * y)^{c_2} * (1 + 0.002 * z)^{c_3} * (1 + 0.002 * w)^{c_4}$$

- Constraint function (S is storage space):

$$G(x, y, z, w) = s_1 * x + s_2 * y + s_3 * z + s_4 * w = S$$

- Problem: Find the maximum of $F(x, y, z, w)$ with constraint $G(x, y, z, w)$.

8.3 Solution

8.3.1 Non-linear constrained optimization library

- Use [ALGLIB](#). Using this library for solving our problem is not recommended because there is another optimal solution. This library also has 2 disadvantages:
 - Only return approximation.
 - Run much slower than our optimal solution below.
- I will provide sample code, but do not use them in your script without understanding the drawbacks.

- Sample code:

```

async function calculateOptimalBoostMaterialQuantitiesWithAlglibJs(
  matCoefficients: number[],
  matSizes: number[],
  spaceConstraint: number): Promise<number[]> {
  let result: number[] = [];
  const f1 = function ([x, y, z, w]: number[]) {
    return Math.pow(1 + 0.002 * x, matCoefficients[0])
      * Math.pow(1 + 0.002 * y, matCoefficients[1])
      * Math.pow(1 + 0.002 * z, matCoefficients[2])
      * Math.pow(1 + 0.002 * w, matCoefficients[3]);
  };
  const constrainedFunction = function ([x, y, z, w]: number[]) {
    return matSizes[0] * x + matSizes[1] * y + matSizes[2] * z +
matSizes[3] * w - spaceConstraint;
  };
  const solverAlglib = new Alglib();
  await solverAlglib.promise.then(function () {
    solverAlglib.add_function(f1);

solverAlglib.add_less_than_or_equal_to_constraint(constrainedFunction);
    const guess = [0, 0, 0, 0];
    const success = solverAlglib.solve("max", guess);
    if (!success) {
      throw new Error("Cannot find solution");
    }
    result = <number[]>solverAlglib.get_results();
    solverAlglib.remove();
  });
  return result;
}

```

8.3.2 Lagrange multiplier method

Disclaimer: This is based on discussion between @Jesus and @yichizhng on Discord. All credit goes to them.

- By using the [Lagrange multiplier](#) method, we have this system:

$$\left\{ \begin{array}{l} \frac{\partial F}{\partial x} = \lambda \frac{\partial G}{\partial x} \\ \frac{\partial F}{\partial y} = \lambda \frac{\partial G}{\partial y} \\ \frac{\partial F}{\partial z} = \lambda \frac{\partial G}{\partial z} \\ \frac{\partial F}{\partial w} = \lambda \frac{\partial G}{\partial w} \\ G(x, y, z, w) = S \end{array} \right.$$

- In order to solve this system, we have 2 choices:

- Solve that system with [Ceres Solver](#).
- Do the hard work with basic calculus and algebra. This is the optimal way in both accuracy and performance, so we will focus on it. In the following sections, I will show the proof and sample code for this solution.

$$x * s_1 = \frac{S - 500 * \left(\frac{s_1}{c_1} * (c_2 + c_3 + c_4) - (s_2 + s_3 + s_4) \right)}{\frac{c_1 + c_2 + c_3 + c_4}{c_1}}$$

$$y * s_2 = \frac{S - 500 * \left(\frac{s_2}{c_2} * (c_1 + c_3 + c_4) - (s_1 + s_3 + s_4) \right)}{\frac{c_1 + c_2 + c_3 + c_4}{c_2}}$$

$$z * s_3 = \frac{S - 500 * \left(\frac{s_3}{c_3} * (c_1 + c_2 + c_4) - (s_1 + s_2 + s_4) \right)}{\frac{c_1 + c_2 + c_3 + c_4}{c_3}}$$

$$w * s_4 = \frac{S - 500 * \left(\frac{s_4}{c_4} * (c_1 + c_2 + c_3) - (s_1 + s_2 + s_3) \right)}{\frac{c_1 + c_2 + c_3 + c_4}{c_4}}$$

8.4 Proof

- @Jesus came up with idea of using Lagrange multiplier and did the initial work, but @Jesus used wrong formula for $F(x, y, z, w)$. @Jesus used: $x^{c_1} * y^{c_2} * z^{c_3} * w^{c_4}$ instead of $(1 + 0.002 * x)^{c_1} * (1 + 0.002 * y)^{c_2} * (1 + 0.002 * z)^{c_3} * (1 + 0.002 * w)^{c_4}$. After that, @yichizhng completed the work with proof and sample code (my sample code is based on @yichizhng's code). However, there are some typos in @yichizhng's proof:

```
set these equal to each other; the third and fourth parts obviously cancel,
and collecting the variables:
0.72 * 0.5 * (1+(0.002/0.5)b) = 0.3 * 0.005 * (1+(0.002/0.005)a)
      0.4                0.004

solving for a,c,d in terms of b and simplifying:
a = 597.5 + 2.4*b <- I only check until this line, so I'm not sure about the numbers in the following parts
c = 200 + b
d = 410/3 + (2/3)*b

plug those into the space equation and solve for S:
S = a + b + c + d = (5605/6) + (76/15)*b
b = (15/76)*S - 1475/8

plug b back in to get a,c,d in terms of S and then divide by the size to get x,y,z,w back:
x = (1800/19)*S + 31000
y = (15/38)*S - 1475/4
z = (450/228)*S + 625/4
w = (250/114)*S + 1375/6
```

```

* (m2*w2) = (500 * (p2*w1 - p1*w2)) / p1 + (p2/p1)*(m1*w1)) ← Correct
* with the others being analogous. summing them up:
* S = m1*w1 + m2*w2 + m3*w3 + m4*w4
* = m1*w1 +
*   (500 * (p2*w1 - p1*w2)) / p1 + (p1/p2)*(m1*w1) + ← Typo
*   (500 * (p3*w1 - p1*w3)) / p1 + (p1/p3)*(m1*w1) +
*   (500 * (p4*w1 - p1*w4)) / p1 + (p1/p4)*(m1*w1)
*
* We can now solve this for m1/w1 in terms of S:
* m1*w1 = (S - 500 * ((w1/p1)*(p2+p3+p4) - (w2+w3+w4))) / (1 + p1/p2 + p1/p3 + p1/p4)
* and the other values follow by analogy.
*/

```

- That's why I rewrite the proof here in my own way.

- Define: $k = 0.002$

$$\begin{cases} \frac{\partial F}{\partial x} = (k * c_1 * (1 + k * x)^{c_1-1}) * (1 + k * y)^{c_2} * (1 + k * z)^{c_3} * (1 + k * w)^{c_4} = \lambda * s_1 \\ \frac{\partial F}{\partial y} = (1 + k * x)^{c_1} * (k * c_1 * (1 + k * y)^{c_2-1}) * (1 + k * z)^{c_3} * (1 + k * w)^{c_4} = \lambda * s_2 \end{cases}$$

≡

$$k * c_1 * (1 + k * x)^{-1} * s_2 = k * c_2 * (1 + k * y)^{-1} * s_1$$

≡

$$c_1 * s_2 * (1 + k * y) = c_2 * s_1 * (1 + k * x)$$

≡

$$1 + k * y = \frac{c_2 * s_1}{c_1 * s_2} * (1 + k * x)$$

≡

$$y = \frac{c_2 * s_1 + k * x * c_2 * s_1 - c_1 * s_2}{k * c_1 * s_2}$$

≡

$$y * s_2 = \frac{c_2 * s_1 * s_2 + k * x * c_2 * s_1 * s_2 - c_1 * s_2 * s_2}{k * c_1 * s_2}$$

≡

$$y * s_2 = \frac{c_2 * s_1}{k * c_1} + \frac{x * c_2 * s_1}{c_1} - \frac{s_2}{k}$$

≡

$$y * s_2 = \frac{c_2}{c_1} * x * s_1 + \frac{1}{k} * \frac{c_2 * s_1 - c_1 * s_2}{c_1}$$

≡

$$y * s_2 = \frac{c_2}{c_1} * x * s_1 + 500 * \frac{c_2 * s_1 - c_1 * s_2}{c_1}$$

- Repeating above steps, we have:

$$z * s_3 = \frac{c_3}{c_1} * x * s_1 + 500 * \frac{c_3 * s_1 - c_1 * s_3}{c_1}$$

$$w * s_4 = \frac{c_4}{c_1} * x * s_1 + 500 * \frac{c_4 * s_1 - c_1 * s_4}{c_1}$$

- Substituting into the constraint function:

$$x * s_1 + y * s_2 + z * s_3 + w * s_4 = S$$

≡

$$x * s_1 + \frac{c_2}{c_1} * x * s_1 + 500 * \frac{c_2 * s_1 - c_1 * s_2}{c_1} + \frac{c_3}{c_1} * x * s_1 + 500 * \frac{c_3 * s_1 - c_1 * s_3}{c_1} + \frac{c_4}{c_1} * x * s_1 + 500 * \frac{c_4 * s_1 - c_1 * s_4}{c_1} = S$$

≡

$$\frac{x * s_1 * (c_1 + c_2 + c_3 + c_4)}{c_1} + \frac{500}{c_1} * (c_2 * s_1 - c_1 * s_2 + c_3 * s_1 - c_1 * s_3 + c_4 * s_1 - c_1 * s_4) = S$$

≡

$$\frac{x * s_1 * (c_1 + c_2 + c_3 + c_4)}{c_1} + \frac{500}{c_1} * (s_1 * (c_2 + c_3 + c_4) - c_1 * (s_2 + s_3 + s_4)) = S$$

≡

$$x * s_1 * \frac{c_1 + c_2 + c_3 + c_4}{c_1} + \frac{500}{c_1} * (s_1 * (c_2 + c_3 + c_4) - c_1 * (s_2 + s_3 + s_4)) = S$$

≡

$$x * s_1 = \frac{S - 500 * \left(\frac{s_1}{c_1} * (c_2 + c_3 + c_4) - (s_2 + s_3 + s_4) \right)}{\frac{c_1 + c_2 + c_3 + c_4}{c_1}}$$

- We can do the same steps for y,z,w.

8.5 Handle low storage space

With small S, any variable (x,y,z,w) can be negative. In that case, we remove the variable that ends up being negative and then redo the steps above. When implementing this solution, we can use a recursive function to handle those cases.

8.6 Sample code

This is based on @yichizhng's and @CaldZoëll's code on Discord. I only rename variables.

[corporationUtils.ts](#): `getOptimalBoostMaterialQuantities()`

9. Division raw production

9.1 Definition

- Each industry requires different input materials. Each required material has its own coefficient. This value is not the same as the boost material's coefficient; they are different things. For example:
 - Agriculture: { Water: 0.5, Chemicals: 0.2 }
 - Chemical: { Plants: 1, Water: 0.5 }
 - Tobacco: { Plants: 1 }
- Each division has a number that I call "Division raw production". This raw value is the division's production capability. Let's call it RawProduction. It's used for:
 - Calculating how much input material that we need. It's multiplied by the input material's coefficient to find the required quantity of that input material.
 - Calculating how much material/product that division can produce. It's multiplied by ProducibleFrac. ProducibleFrac starts at 1 and is reduced if there are not enough input materials.
- For example, with Agriculture, if RawProduction is 1000, we need 500 units of Water and 200 units of Chemicals. With these input materials, we can produce 1000 units of Plants and 1000 units of Food.

9.1.1 Formula

- RawProduction is the product of 4 multipliers:
 - Office multiplier:
 - ❖ Employee production in 3 jobs (Operations, Engineer, Management) and their sum:

$$\text{OperationsProd} = \text{office.employeeProductionByJob.Operations}$$

$$\text{EngineerProd} = \text{office.employeeProductionByJob.Engineer}$$

$$\text{ManagementProd} = \text{office.employeeProductionByJob.Management}$$

$$\text{TotalEmployeesProd} = \text{OperationsProd} + \text{EngineerProd} + \text{ManagementProd}$$

- Management factor:

$$\text{ManagementFactor} = 1 + \frac{\text{ManagementProd}}{1.2 * \text{TotalEmployeesProd}}$$

- Employee production multiplier:

$$\text{EmployeeProductionMultiplier} = ((\text{OperationsProd})^{0.4} + (\text{EngineerProd})^{0.3}) * \text{ManagementFactor}$$

- Balancing multiplier:

$$\text{BalancingMultiplier} = 0.05$$

- ❖ If output is material:

$$\text{OfficeMultiplier} = \text{BalancingMultiplier} * \text{EmployeeProductionMultiplier}$$

- ❖ If output is product:

$$\text{OfficeMultiplier} = 0.5 * \text{BalancingMultiplier} * \text{EmployeeProductionMultiplier}$$

- Division production multiplier: see previous [part](#).
- Upgrade multiplier: multiplier from Smart Factories.
- Research multiplier: multiplier from researches.

9.1.2 Sample code

[corporationFormulas.ts](#): `getDivisionRawProduction()`

9.2 Optimizer

- There are situations that we need to find the best combination of Smart Storage, Smart Factories and warehouse upgrade to maximize RawProduction. In these situations, multipliers of offices and researches can be seen as constants.
- In order to do this, we create an optimizer that check all combinations of Smart Storage, Smart Factories and warehouse upgrade. With each combination, we calculate relevant data: cost, production, costPerProduction, boostMaterials, boostMaterialMultiplier.
- When we calculate production, we only need to multiply DivisionProductionMultiplier with UpgradeMultiplier. OfficeMultiplier and ResearchMultiplier are constants in these cases, so we can skip them.
- Sample data:

```
(storage:25, warehouse:15, factory:16, totalCost:322.402e9, warehouseSize:5.250e3, production:1.190e3, costPerProduction:270.886e6, boostMaterialMultiplier:804.175, boostMaterials:8446.9440,428895,1289)
(storage:21, warehouse:16, factory:19, totalCost:326.547e9, warehouseSize:4.960e3, production:1.191e3, costPerProduction:274.148e6, boostMaterialMultiplier:758.683, boostMaterials:7988.8931,406916,1198)
(storage:25, warehouse:16, factory:13, totalCost:326.535e9, warehouseSize:5.600e3, production:1.195e3, costPerProduction:273.313e6, boostMaterialMultiplier:859.516, boostMaterials:8998.10054,455421,1400)
(storage:21, warehouse:17, factory:16, totalCost:329.325e9, warehouseSize:5.270e3, production:1.195e3, costPerProduction:275.637e6, boostMaterialMultiplier:807.282, boostMaterials:8477.9475,430411,1295)
(storage:22, warehouse:15, factory:21, totalCost:328.098e9, warehouseSize:4.800e3, production:1.196e3, costPerProduction:274.379e6, boostMaterialMultiplier:733.612, boostMaterials:7735.8650,394789,1147)
(storage:23, warehouse:17, factory:13, totalCost:329.750e9, warehouseSize:5.610e3, production:1.197e3, costPerProduction:275.502e6, boostMaterialMultiplier:861.084, boostMaterials:9014.10071,456179,1403)
(storage:23, warehouse:16, factory:16, totalCost:324.378e9, warehouseSize:5.280e3, production:1.197e3, costPerProduction:270.941e6, boostMaterialMultiplier:808.938, boostMaterials:8493.9492,431168,1299)
(storage:27, warehouse:14, factory:17, totalCost:328.611e9, warehouseSize:5.180e3, production:1.198e3, costPerProduction:274.377e6, boostMaterialMultiplier:793.156, boostMaterials:8335.9317,423589,1267)
(storage:25, warehouse:14, factory:20, totalCost:328.074e9, warehouseSize:4.900e3, production:1.199e3, costPerProduction:273.656e6, boostMaterialMultiplier:749.285, boostMaterials:7893.8826,402368,1179)
(storage:26, warehouse:15, factory:15, totalCost:326.193e9, warehouseSize:5.400e3, production:1.200e3, costPerProduction:271.726e6, boostMaterialMultiplier:827.897, boostMaterials:8683.9703,440263,1337)
(storage:24, warehouse:15, factory:18, totalCost:324.771e9, warehouseSize:5.100e3, production:1.202e3, costPerProduction:270.157e6, boostMaterialMultiplier:780.619, boostMaterials:8209.9177,417526,1242)
(storage:22, warehouse:16, factory:18, totalCost:327.637e9, warehouseSize:5.120e3, production:1.207e3, costPerProduction:271.464e6, boostMaterialMultiplier:783.718, boostMaterials:8240.9212,419042,1248)
(storage:24, warehouse:16, factory:15, totalCost:327.224e9, warehouseSize:5.440e3, production:1.210e3, costPerProduction:270.541e6, boostMaterialMultiplier:834.150, boostMaterials:8746.9773,443935,1349)
(storage:23, warehouse:15, factory:20, totalCost:328.881e9, warehouseSize:4.950e3, production:1.211e3, costPerProduction:271.533e6, boostMaterialMultiplier:757.025, boostMaterials:7972.8913,406158,1194)
(storage:25, warehouse:15, factory:17, totalCost:327.483e9, warehouseSize:5.250e3, production:1.214e3, costPerProduction:269.688e6, boostMaterialMultiplier:804.175, boostMaterials:8446.9440,428895,1289)
(storage:23, warehouse:16, factory:17, totalCost:329.459e9, warehouseSize:5.280e3, production:1.221e3, costPerProduction:269.717e6, boostMaterialMultiplier:808.938, boostMaterials:8493.9492,431168,1299)
```

- The top line is what I use in round 2. If you start round 2 with more than 431b, you can invest more in Smart Storage, Smart Factories and warehouse upgrade. In that case, run this optimizer and find out your best combination.

9.3 Sample code

[corporationOptimizer.ts](#): `optimizeStorageAndFactory()`

10. Office

10.1 Basic information

- Employee's stats are kept track as average value. There are 6 average values: AvgEnergy, AvgMorale, AvgIntelligence, AvgCharisma, AvgCreativity, AvgEfficiency. Every time you hire a new employee, these average values are recalculated, they are modified by a randomized number from 50 to 100:

```
this.avgMorale = (this.avgMorale * this.numEmployees +  
getRandomInt(averageStat, averageStat)) / (this.numEmployees + 1);
```

- Job assignment:
 - Each office has 2 records: employeeJobs and employeeNextJobs. Data in employeeJobs (number of employees in each job) is used for calculations of other relevant data like EmployeeProductionByJob, AvgEnergy, AvgMorale, TotalExperience. When you call setAutoJobAssignment, its parameter is calculated and assigned to employeeNextJobs. In next cycle's START state, data in employeeNextJobs will be copied to employeeJobs.
 - Behavior of setAutoJobAssignment may be confused at first glance. Let's say you call it like this:

ns.corporation.setAutoJobAssignment("Agriculture", "Sector-12", "Operations", 5)

- ❖ If you have 5 "Operations" employees, it does nothing.
 - ❖ If you have 7 "Operations" employees, it reduces number of "Operations" employees to 5, and set "Unassigned" employees to 2.
 - ❖ If you have 2 "Operations" employees, it checks if you have at least 3 "Unassigned" employees. If yes, it changes "Operations" employees to 5 and reduces "Unassigned" employees by 3. If not, it throws error. Essentially, it tries to move employees from "Unassigned" to "Operations".
- This means the proper way to use setAutoJobAssignment is:
 - ❖ Use it to set all jobs to 0.
 - ❖ Use it to set all jobs to your requirements.
- Total experience is increased in these cases:
 - Hire new employee. Each new employee increases total experience by getRandomInt(50, 100).
 - In START state. Gain per cycle:

$$TotalExperienceGain = 0.0015 * (TotalEmployees - UnassignedEmployees + InternEmployees * 9)$$

- If an office has 100 employees and all employees are assigned to non-intern positions, it gains 0.15 experience/cycle. It's 54 experience/hour without bonus time.
- Salary per cycle:

$$Salary = 3 * TotalEmployees * \left(AvgIntelligence + AvgCharisma + AvgCreativity + AvgEfficiency + \frac{TotalExperience}{TotalEmployees} \right)$$

10.2 Upgrade

- Upgrade cost:

$$UpgradeCost = BasePrice * \left(\frac{\sqrt[3]{1.09} - 1}{0.09} \right) * 1.09^{\frac{CurrentSize}{3}}$$

- Upgrade cost from size 3 to size n:

$$(UpgradeCost)_{From\ 3\ to\ n} = \sum_{k=3}^{n-1} BasePrice * \left(\frac{\sqrt[3]{1.09} - 1}{0.09} \right) * 1.09^{\frac{k}{3}}$$

≡

$$(UpgradeCost)_{From\ 3\ to\ n} = \sum_{k=3}^{n-1} BasePrice * \left(\frac{\sqrt[3]{1.09} - 1}{0.09} \right) * (\sqrt[3]{1.09})^k$$

≡

$$(UpgradeCost)_{From\ 3\ to\ n} = BasePrice * \left(\frac{\sqrt[3]{1.09} - 1}{0.09} \right) * \left(\frac{(\sqrt[3]{1.09})^n - 1.09}{\sqrt[3]{1.09} - 1} \right)$$

≡

$$(UpgradeCost)_{From\ 3\ to\ n} = BasePrice * \left(\frac{1.09^{\frac{n}{3}} - 1.09}{0.09} \right)$$

- Upgrade cost size a to size b:

$$(UpgradeCost)_{From\ a\ to\ b} = BasePrice * \left(\frac{1.09^{\frac{b}{3}} - 1.09^{\frac{a}{3}}}{0.09} \right)$$

- Maximum size with a given MaxCost:

$$MaxSize = 3 * \log_{1.09} \left(MaxCost * \frac{0.09}{BasePrice} + 1.09^{\frac{CurrentSize}{3}} \right)$$

10.3 Energy and morale

- They are calculated in START state.
- They start dropping when your office's number of employees is greater than or equal to 9. The minimum value is 10.
- PerfMult is a multiplier that increases/decreases energy/morale.

$$InternMultiplier = 0.002 * \min \left(\frac{1}{9}, \frac{InternEmployees}{TotalEmployees} - \frac{1}{9} \right) * 9$$

$$PenaltyMultiplier = \begin{cases} 0, & (CorpFunds > 0) \vee (DivisionLastCycleRevenue > DivisionLastCycleExpenses) \\ 0.001, & (CorpFunds < 0) \wedge (DivisionLastCycleRevenue < DivisionLastCycleExpenses) \end{cases}$$

$$PerfMult = \begin{cases} 1.002, & TotalEmployees < 9 \\ 1 + InternMultiplier - PenaltyMultiplier, & TotalEmployees \geq 9 \end{cases}$$

- Buying tea gives a flat +2 to energy. It costs 500e3 per employee.
- When throwing party, PartyMult is calculated. It's used in the calculation of morale in the next cycle.

$$PartyMult = 1 + \frac{PartyCostPerEmployee}{10^7}$$

- PartyMult is not affected by number of employees. Therefore, you can throw a “big party” (high PartyCostPerEmployee) when you have 1 employee at low total cost (due to having only 1 employee), then hire the rest of employees later.
- There is a flat randomized reduction of energy/morale per cycle.
 - It's capped at 0.002 per cycle. This is tiny so it's not a problem.
 - The comment in source code is outdated, it states that the cap is 0.001. However, it's 0.002 now, this was changed in commit 1f98eeeb57fd65d86d11f30b96ffeb429539df96, maybe Mughur forgot to update the comment.
- There is a flat increase of morale if PartyMult is greater than 1. PartyMult is based on PartyCostPerEmployee, so this increase is based on PartyCostPerEmployee.

$$IncreaseOfMorale = (PartyMult - 1) * 10$$

≡

$$IncreaseOfMorale = \frac{PartyCostPerEmployee}{10^6}$$

```
const reduction = 0.002 * marketCycles;
const increase = this.partyMult > 1 ? (this.partyMult - 1) * 10 : 0;
this.avgEnergy = (this.avgEnergy - reduction * Math.random()) * perfMult +
(this.teaPending ? 2 : 0);
this.avgMorale = ((this.avgMorale - reduction * Math.random()) * perfMult +
increase) * this.partyMult;
```

- There are 3 ways to counter the drop of energy/morale:
 - Buy tea and throw party. You should always use this option. Writing script to automate them is very easy.
 - Assign Intern. Many people throw around the ratio 1/9 as the way to counter the drop of energy/morale. You can only use that ratio when your corporation/division works fine. If it does not, there is a penalty multiplier (0.001). In this case, you need to use 1/6.
 - Buy 2 researches: AutoBrew and AutoPartyManager. They keep energy/morale at maximum value. However, you should never buy them. It's always better to spend your RP on other useful researches or just stock up on RP.
- AvgEnergy and AvgMorale are increased by a randomized amount when we hire new employee.

```

this.avgMorale = (this.avgMorale * this.numEmployees + getRandomInt(50, 100))
/ (this.numEmployees + 1);
this.avgEnergy = (this.avgEnergy * this.numEmployees + getRandomInt(50, 100))
/ (this.numEmployees + 1);

```

- Optimal PartyCostPerEmployee:

- The flat randomized reduction is tiny, so we can ignore it.
- We want to increase AvgMorale from CurrentMorale to MaxMorale:

$$\left(\text{CurrentMorale} * \text{PerfMult} + \frac{\text{PartyCostPerEmployee}}{10^6} \right) * \left(1 + \frac{\text{PartyCostPerEmployee}}{10^7} \right) = \text{MaxMorale}$$

- Define:

$$a = \text{CurrentMorale}$$

$$b = \text{MaxMorale}$$

$$k = \text{PerfMult}$$

$$x = \text{PartyCostPerEmployee}$$

- We have equation:

$$\left(a * k + \frac{x}{10^6} \right) * \left(1 + \frac{x}{10^7} \right) = b$$

≡

$$x_1 = -500000 * \left(\sqrt{(a * k - 10)^2 + 40 * b} + a * k + 10 \right)$$

$$x_2 = 500000 * \left(\sqrt{(a * k - 10)^2 + 40 * b} - a * k - 10 \right)$$

- x_1 is always negative. Therefore, x_2 is the only solution.
- One big party is less cost-effective than multiple small parties. For example: throwing 1 big party for 70→100 morale costs more than throwing 3 small parties: 70→80, 80→90, 90→100.
- Don't be a cheapskate when it comes to tea/party. Energy and morale are vital to efficient office. Check the next part for the formulas.
 - It's fine to spend 500e3 per employee when throwing party. You can spend more if you want.
 - Try to maintain maximum energy/morale at all times. Personally, I always buy tea / throw party when energy/morale decreases to 99.5 (109.5, if I bought the relevant researches).
 - In round 1 and 2, the office's size is small, it's usually smaller than 9, so energy/morale is not a problem. In round 3+, you should buy tea / throw party every cycle.

10.4 Employee production by job

- In each cycle's START state, all stats are used for calculating "production" values, these values are saved in office.employeeProductionByJob. These values can be used later for calculating:
 - RP.

- Material's quality.
- Product's stats.
- Division raw production.
- Material/product's MaxSalesVolume.
- Formulas:
 - Calculate multipliers of Intelligence, Charisma, Creativity, and Efficiency. They are the product of average value, upgrade benefit and research benefit.
 - Production base:

$$ProductionBase = AvgMorale * AvgEnergy * 10^{-4}$$
 - Experience:

$$Exp = \frac{TotalExperience}{TotalEmployees}$$
 - Production multiplier:
 - ❖ Operations:

$$ProductionMultiplier = 0.6 * IntelligenceMult + 0.1 * CharismaMult + Exp + 0.5 * CreativityMult + EfficiencyMult$$
 - ❖ Engineer:

$$ProductionMultiplier = IntelligenceMult + 0.1 * CharismaMult + 1.5 * Exp + EfficiencyMult$$
 - ❖ Business:

$$ProductionMultiplier = 0.4 * IntelligenceMult + CharismaMult + 0.5 * Exp$$
 - ❖ Management:

$$ProductionMultiplier = 2 * CharismaMult + Exp + 0.2 * CreativityMult + 0.7 * EfficiencyMult$$
 - ❖ Research and Development:

$$ProductionMultiplier = 1.5 * IntelligenceMult + 0.8 * Exp + CreativityMult + 0.5 * EfficiencyMult$$
 - $EmployeesJobCount = office.employeeJobs[JobName]$
 - Employee production by job:

$$EmployeeProductionByJob = EmployeesJobCount * ProductionMultiplier * ProductionBase$$

10.5 Calculate employee's stat

- 4 stats AvgIntelligence, AvgCharisma, AvgCreativity, AvgEfficiency are inaccessible through NS API.
- We can calculate them by using the formulas from previous part and [Ceres Solver](#). In 5 jobs Operations, Engineer, Business, Management, Research & Development, we need at least 4 jobs having 1 employee at the minimum to use this solution.
- Sample code:

[corporationFormulas.ts](#): `calculateEmployeeStats()`

10.6 Optimizer

[corporationOptimizer.ts](#): optimizeOffice()

[corporationOptimizerTools.ts](#): optimizeOffice()

11. Quality

11.1 Basic term

- Let's define some terms:
 - AvgInputQuality: average quality of input materials.
 - MaxOutputQuality: maximum value of output material's quality.
 - OutputQuality: final value of output material's quality. This value is always less than or equal to MaxOutputQuality.
 - MaxOutputRating: maximum value of output product's rating.
 - OutputRating: final value of output product's rating. This value is always less than or equal to MaxOutputRating.
- Each industry has a set of input materials and their coefficients, for example, Agriculture needs Water and Chemicals, their coefficients are [0.5, 0.2] respectively. These coefficients do not affect AvgInputQuality. AvgInputQuality is the mean value of qualities of input materials in warehouse. For example, if Agriculture division's warehouse has Water (quality 1) and Chemicals (quality 11), AvgInputQuality is $(1+11)/2$.
- Purchased material is low-quality. Its quality is always 1.
- When you import/export your materials between different divisions, you can see quality of some input materials constantly change. Quality is high after EXPORT state, but it reduces after PURCHASE state in the next cycle. Qualities of the materials in warehouse are recalculated in these 2 states.
- In PURCHASE state, material's quality is "diluted" by low-quality purchased material (quality 1).

$$Quality = \frac{Quality * CurrentQuantity + BuyAmount}{CurrentQuantity + BuyAmount}$$

- In PRODUCTION state, the "diluted" quality value is used for calculating AvgInputQuality.
- In EXPORT state:

$$Quality = \frac{Quality * CurrentQuantity + ImportQuality * ImportAmount}{CurrentQuantity + ImportAmount}$$

- The production capability of support division should be balanced. The ImportAmount (the number of material units that the support division exports) does not need to equal the required number of input material units, but it should also not be too small.

11.2 Material

- MaxOutputQuality is sum of 3 values:
 - Engineer summand:

$$EngineerProduction = office.employeeProductionByJob["Engineer"]$$

$$EngineerSummand = \frac{EngineerProduction}{90}$$

- Research point summand:

$$ResearchPointSummand = (RP)^{IndustryScienceFactor}$$

- AI Cores summand: if there is AI Cores in the warehouse:

$$AICoresSummand = \frac{(AICoresQuantity)^{IndustryAICoreFactor}}{1000}$$

- Output quality:

$$OutputQuality = \sqrt{MaxOutputQuality} * AvgInputQuality$$

- With formulas above, we have these conclusions:

- In early rounds, increasing RP is the best way to improve MaxOutputQuality. This is especially true for industry with high science factor like Chemical.
- In late rounds (round 3+), we have large funds to upgrade offices. In this case, the most important factor of MaxOutputQuality is EngineerProduction. “Engineer” is more significant than “Research & Development”.
- OutputQuality starts at square root of MaxOutputQuality. AvgInputQuality increases it until it reaches MaxOutputQuality.
- This is a simple strategy for checking if we need to increase AvgInputQuality:
 - ❖ If square of AvgInputQuality is greater than or equal to current output quality, it’s fine.
 - ❖ If not, you need to increase input material’s quality. It usually means you need to improve the support division.

11.3 Product

- MaxOutputRating is product.rating.
- Game UI shows OutputRating as “Effective rating”
- Output rating:

$$OutputRating = \sqrt{MaxOutputRating} * AvgInputQuality$$

- Use same strategy as material for checking AvgInputQuality.

12. Smart Supply

12.1 Logic

- Create a function called `getLimitedRawProduction`:
 - Calculate `RawProduction`.
 - Multiply `RawProduction` by 10.
 - Calculate `RequiredStorageSpaceOfEachOutputUnit`. It is the net change in warehouse's storage space when producing an output unit.
 - If `RequiredStorageSpaceOfEachOutputUnit` is greater than 0:
 - ❖ Calculate `MaxNumberOfOutputUnits`. It is the maximum number of units that we can produce and store in warehouse's free space.
 - ❖ Limit `RawProduction` to `MaxNumberOfOutputUnits` if needed.
- Create a map called `SmartSupplyData`. Its key is ``${divisionName}|${city}``. Its value is the `TotalRawProduction` that will be calculated later.
- After `PURCHASE` state:
 - Initialize `TotalRawProduction` with 0.
 - If division produces materials: call `getLimitedRawProduction`, then add the returned value to `TotalRawProduction`.
 - If division produces products:
 - ❖ Loop through all products of the division.
 - ❖ If the product is finished, call `getLimitedRawProduction`, then add the returned value to `TotalRawProduction`.
 - Set `TotalRawProduction` to `SmartSupplyData`.
- Before `PURCHASE` state:
 - Check if warehouse is congested. If it is, alert player and try to mitigate the situation.
 - Find required quantity of each input material to produce material/product:
 - ❖ Get `TotalRawProduction` from `SmartSupplyData`.
 - ❖ Multiply `TotalRawProduction` with input material's coefficient.
 - Find which input material creates the least number of output units.
 - Align all the input materials to the smallest amount.
 - Calculate the total size of all input materials we are trying to buy.
 - If there is not enough free space, we apply a multiplier to required quantity to not overfill warehouse.
 - Deduct the number of stored input material units from the required quantity.
 - Buy input materials.

12.2 Detect warehouse congestion

- Warehouse can be congested due to multiple reasons. One common case is when the logic of calculating required quantity of input materials does not take into account of free space and stored input materials units in warehouse. When it happens, warehouse is filled with excessive input materials and the production process is halted completely due to no free space for produced units.
- For each case, we need to find way(s) to detect congestion and mitigate it. In the case above, we can use this simple heuristic:
 - Create a map called WarehouseCongestionData. Its key is `\${divisionName}|\${city}`. Its value is the number of times that we suspect the warehouse is congested.
 - In each cycle, check material.productionAmount and product.productionAmount of output material/product.
 - ❖ If productionAmount is 0, increase the entry's value of this warehouse in the map by 1. If not, set the entry's value to 0.
 - ❖ If the entry's value is greater than 5, the warehouse is very likely congested.
 - This heuristic is based on the observation: when warehouse is filled with excessive input materials, the production process is halted completely, this means productionAmount is 0. We wait for 5 times to reduce false positives.
 - When we start our Smart Supply script, productionAmount of output material/product may be 0, because nothing controls the production process in previous cycles.
- When there are excessive input materials, discarding all of them is the simplest mitigation measure. It's inefficient, but it's the fastest way to make our production line restart.

12.3 Sample code

[corporationUtils.ts](#): getLimitedRawProduction()

[corporationUtils.ts](#): setSmartSupplyData()

[corporationUtils.ts](#): detectWarehouseCongestion()

[corporationUtils.ts](#): buyOptimalAmountOfInputMaterials()

13. Wilson Analytics – Advert

13.1 Awareness and popularity

- Wilson and Advert increase 2 important stats: awareness and popularity. These 2 stats affect initial demand and maximum sellable number of product units. I will show formulas for them in later sections.
- Awareness and popularity are capped at `Number.MAX_VALUE` (~1.7976931348623157E+308).
- Raw values of those stats are crucial, but their ratio is also important. We want to have high ratio of popularity/awareness, check this [part](#) for formulas.
- DreamSense only increases popularity by 0.001/cycle/level and awareness by 0.004/cycle/level. Those benefits are minuscule. However, that's not its biggest problem, the biggest one is the ratio of popularity/awareness. We want that ratio to be as high as possible, but DreamSense constantly lowers it.
- Popularity decreases by 0.0001 per cycle.

13.2 Advert

- Advert is a special upgrade. It affects only the division that buys it.
- Cost: use the formulas in this [part](#) with `BasePrice = 109` and `PriceMult = 1.06`.
- Benefit:

$$AdvertMultiplier = WilsonUpgradeBenefit * ResearchAdvertisingMultiplier$$

$$Awareness = (Awareness + 3 * AdvertMultiplier) * (1.005 * AdvertMultiplier)$$

$$Popularity = (Popularity + AdvertMultiplier) * (1 + \frac{Random(1,3)}{200}) * AdvertMultiplier$$

13.3 Wilson Analytics

- Wilson is a multiplier that is applied on Advert's benefits when we buy Advert, so it's not retroactive. Therefore, we need to buy it as soon as possible. However, there are cases that Wilson is too expensive and it does not bring much benefits, round 1 and 2 are those cases.
- Wilson has `priceMult` of 2, so its price is doubled every time we buy it. The exponentiation of Advert's cost function uses base of 1.06, so its price increases much slower.

13.4 Optimizer

- In order to see how Wilson affects Advert, we create an optimizer that checks combinations of Wilson and Advert, calculate relevant data: `totalCost`, `popularity`, `awareness`, `ratio (popularity/awareness)`, [advertisingFactor](#), `costPerAdvertisingFactor`.
 - First, we run test cases with extremely low budget:
 - ❖ `Budget < 10e9`:

```
{wilson:0, advert:1, totalCost:1.000e9, advertisingFactor:0.424, popularity:1.010, awareness:3.015, ratio:0.335, costPerAdvertisingFactor:2.358e9}
{wilson:1, advert:1, totalCost:5.000e9, advertisingFactor:0.424, popularity:1.020, awareness:3.045, ratio:0.335, costPerAdvertisingFactor:11.784e9}
{wilson:0, advert:2, totalCost:2.060e9, advertisingFactor:0.439, popularity:2.030, awareness:6.045, ratio:0.336, costPerAdvertisingFactor:4.691e9}
{wilson:1, advert:2, totalCost:6.060e9, advertisingFactor:0.439, popularity:2.056, awareness:6.121, ratio:0.336, costPerAdvertisingFactor:13.792e9}
{wilson:0, advert:3, totalCost:3.184e9, advertisingFactor:0.450, popularity:3.060, awareness:9.090, ratio:0.337, costPerAdvertisingFactor:7.077e9}
{wilson:1, advert:3, totalCost:7.184e9, advertisingFactor:0.450, popularity:3.107, awareness:9.228, ratio:0.337, costPerAdvertisingFactor:15.955e9}
{wilson:0, advert:4, totalCost:4.375e9, advertisingFactor:0.458, popularity:4.101, awareness:12.151, ratio:0.338, costPerAdvertisingFactor:9.543e9}
{wilson:1, advert:4, totalCost:8.375e9, advertisingFactor:0.459, popularity:4.174, awareness:12.365, ratio:0.338, costPerAdvertisingFactor:18.250e9}
{wilson:0, advert:5, totalCost:5.637e9, advertisingFactor:0.466, popularity:5.152, awareness:15.227, ratio:0.338, costPerAdvertisingFactor:12.107e9}
{wilson:1, advert:5, totalCost:9.637e9, advertisingFactor:0.466, popularity:5.256, awareness:15.534, ratio:0.338, costPerAdvertisingFactor:20.672e9}
{wilson:0, advert:6, totalCost:6.975e9, advertisingFactor:0.472, popularity:6.214, awareness:18.318, ratio:0.339, costPerAdvertisingFactor:14.781e9}
{wilson:1, advert:6, totalCost:10.975e9, advertisingFactor:0.472, popularity:6.214, awareness:18.318, ratio:0.339, costPerAdvertisingFactor:23.267e9}
{wilson:0, advert:7, totalCost:8.394e9, advertisingFactor:0.478, popularity:7.286, awareness:21.424, ratio:0.340, costPerAdvertisingFactor:17.577e9}
{wilson:1, advert:7, totalCost:12.394e9, advertisingFactor:0.478, popularity:7.286, awareness:21.424, ratio:0.340, costPerAdvertisingFactor:25.064e9}
{wilson:0, advert:8, totalCost:9.897e9, advertisingFactor:0.483, popularity:8.369, awareness:24.546, ratio:0.341, costPerAdvertisingFactor:20.504e9}
{wilson:1, advert:8, totalCost:13.897e9, advertisingFactor:0.483, popularity:8.369, awareness:24.546, ratio:0.341, costPerAdvertisingFactor:28.981e9}
```

In this case, Advert level 8 without Wilson has highest AdvertFactor.

❖ Budget < 20e9:

```
{wilson:0, advert:8, totalCost:9.897e9, advertisingFactor:0.483, popularity:8.369, awareness:24.546, ratio:0.341, costPerAdvertisingFactor:20.504e9}
{wilson:1, advert:8, totalCost:13.897e9, advertisingFactor:0.484, popularity:8.604, awareness:25.234, ratio:0.341, costPerAdvertisingFactor:28.737e9}
{wilson:0, advert:9, totalCost:11.491e9, advertisingFactor:0.487, popularity:9.462, awareness:27.684, ratio:0.342, costPerAdvertisingFactor:23.572e9}
{wilson:1, advert:9, totalCost:15.491e9, advertisingFactor:0.488, popularity:9.754, awareness:28.532, ratio:0.342, costPerAdvertisingFactor:31.712e9}
{wilson:0, advert:10, totalCost:13.181e9, advertisingFactor:0.492, popularity:10.567, awareness:30.837, ratio:0.343, costPerAdvertisingFactor:26.793e9}
{wilson:1, advert:10, totalCost:17.181e9, advertisingFactor:0.493, popularity:10.921, awareness:31.863, ratio:0.343, costPerAdvertisingFactor:34.845e9}
{wilson:0, advert:11, totalCost:14.972e9, advertisingFactor:0.496, popularity:11.683, awareness:34.007, ratio:0.344, costPerAdvertisingFactor:30.175e9}
{wilson:1, advert:11, totalCost:18.972e9, advertisingFactor:0.497, popularity:12.105, awareness:35.228, ratio:0.344, costPerAdvertisingFactor:38.142e9}
```

In this case, Advert level 11 with Wilson level 1 has highest AdvertFactor. However, the second-best combination's AdvertFactor is almost equal the best's one while having much better costPerAdvertisingFactor.

❖ These cases show that [buying Advert without Wilson is a valid strategy in extremely low-budget situations like round 1 and 2.](#)

- Second, we check cases with much larger budget:

❖ Budget < 1e25:

```
{wilson:50, advert:568, totalCost:8.444e24, advertisingFactor:36.795e3, popularity:18.986e57, awareness:3.465e57, ratio:5.480, costPerAdvertisingFactor:229.501e18}
{wilson:49, advert:576, totalCost:8.533e24, advertisingFactor:36.893e3, popularity:12.320e57, awareness:2.162e57, ratio:5.700, costPerAdvertisingFactor:231.283e18}
{wilson:50, advert:569, totalCost:8.681e24, advertisingFactor:37.534e3, popularity:23.970e57, awareness:4.352e57, ratio:5.507, costPerAdvertisingFactor:231.278e18}
{wilson:49, advert:577, totalCost:8.910e24, advertisingFactor:37.625e3, popularity:15.492e57, awareness:2.705e57, ratio:5.728, costPerAdvertisingFactor:236.804e18}
{wilson:50, advert:570, totalCost:8.931e24, advertisingFactor:38.288e3, popularity:30.262e57, awareness:5.468e57, ratio:5.535, costPerAdvertisingFactor:233.267e18}
{wilson:49, advert:578, totalCost:9.309e24, advertisingFactor:38.370e3, popularity:19.481e57, awareness:3.384e57, ratio:5.757, costPerAdvertisingFactor:242.612e18}
{wilson:50, advert:571, totalCost:9.197e24, advertisingFactor:39.058e3, popularity:38.206e57, awareness:6.869e57, ratio:5.562, costPerAdvertisingFactor:235.473e18}
{wilson:49, advert:579, totalCost:9.733e24, advertisingFactor:39.131e3, popularity:24.496e57, awareness:4.234e57, ratio:5.785, costPerAdvertisingFactor:248.718e18}
{wilson:50, advert:572, totalCost:9.479e24, advertisingFactor:39.843e3, popularity:48.235e57, awareness:8.629e57, ratio:5.590, costPerAdvertisingFactor:237.901e18}
{wilson:50, advert:573, totalCost:9.777e24, advertisingFactor:40.644e3, popularity:60.897e57, awareness:10.840e57, ratio:5.618, costPerAdvertisingFactor:240.558e18}
```

❖ Budget < 1e25, but we limit Wilson at level 25:

```
{wilson:25, advert:574, totalCost:5.590e24, advertisingFactor:689.244, popularity:652.088e30, awareness:117.607e30, ratio:5.545, costPerAdvertisingFactor:8.110e21}
{wilson:25, advert:575, totalCost:5.925e24, advertisingFactor:698.078, popularity:740.935e30, awareness:132.969e30, ratio:5.572, costPerAdvertisingFactor:8.488e21}
{wilson:25, advert:576, totalCost:6.281e24, advertisingFactor:707.026, popularity:841.888e30, awareness:150.339e30, ratio:5.600, costPerAdvertisingFactor:8.884e21}
{wilson:25, advert:577, totalCost:6.658e24, advertisingFactor:716.087, popularity:956.595e30, awareness:169.977e30, ratio:5.628, costPerAdvertisingFactor:9.298e21}
{wilson:25, advert:578, totalCost:7.057e24, advertisingFactor:725.265, popularity:1.087e33, awareness:192.180e30, ratio:5.656, costPerAdvertisingFactor:9.731e21}
{wilson:25, advert:579, totalCost:7.481e24, advertisingFactor:734.561, popularity:1.235e33, awareness:217.283e30, ratio:5.684, costPerAdvertisingFactor:10.184e21}
{wilson:25, advert:580, totalCost:7.930e24, advertisingFactor:743.976, popularity:1.403e33, awareness:245.666e30, ratio:5.712, costPerAdvertisingFactor:10.658e21}
{wilson:25, advert:581, totalCost:8.405e24, advertisingFactor:753.511, popularity:1.594e33, awareness:277.756e30, ratio:5.741, costPerAdvertisingFactor:11.155e21}
{wilson:25, advert:582, totalCost:8.910e24, advertisingFactor:763.169, popularity:1.812e33, awareness:314.038e30, ratio:5.769, costPerAdvertisingFactor:11.675e21}
{wilson:25, advert:583, totalCost:9.444e24, advertisingFactor:772.951, popularity:2.059e33, awareness:355.059e30, ratio:5.798, costPerAdvertisingFactor:12.218e21}
```

❖ Budget < 1e35:

```
{wilson:83, advert:964, totalCost:80.057e33, advertisingFactor:326.135e9, popularity:1.473e150, awareness:37.385e147, ratio:39.404, costPerAdvertisingFactor:245.473e21}
{wilson:82, advert:973, totalCost:89.239e33, advertisingFactor:333.049e9, popularity:1.175e150, awareness:28.531e147, ratio:41.198, costPerAdvertisingFactor:267.946e21}
{wilson:83, advert:965, totalCost:82.540e33, advertisingFactor:335.507e9, popularity:2.105e150, awareness:53.165e147, ratio:39.600, costPerAdvertisingFactor:246.015e21}
{wilson:82, advert:974, totalCost:93.433e33, advertisingFactor:342.537e9, popularity:1.674e150, awareness:40.429e147, ratio:41.403, costPerAdvertisingFactor:272.768e21}
{wilson:83, advert:966, totalCost:85.171e33, advertisingFactor:345.148e9, popularity:3.009e150, awareness:75.604e147, ratio:39.797, costPerAdvertisingFactor:246.766e21}
{wilson:82, advert:975, totalCost:97.879e33, advertisingFactor:352.296e9, popularity:2.384e150, awareness:57.290e147, ratio:41.609, costPerAdvertisingFactor:277.831e21}
{wilson:83, advert:967, totalCost:87.960e33, advertisingFactor:355.066e9, popularity:4.300e150, awareness:107.515e147, ratio:39.995, costPerAdvertisingFactor:247.728e21}
{wilson:83, advert:968, totalCost:90.916e33, advertisingFactor:365.269e9, popularity:6.145e150, awareness:152.895e147, ratio:40.194, costPerAdvertisingFactor:248.902e21}
{wilson:83, advert:969, totalCost:94.050e33, advertisingFactor:375.766e9, popularity:8.783e150, awareness:217.428e147, ratio:40.394, costPerAdvertisingFactor:250.290e21}
{wilson:83, advert:970, totalCost:97.372e33, advertisingFactor:386.564e9, popularity:12.552e150, awareness:309.198e147, ratio:40.595, costPerAdvertisingFactor:251.891e21}
```

❖ Budget < 1e35, but we limit Wilson at level 50:

```
(wilson:50, advert:970, totalCost:58.686e33, advertisingFactor:109.662e6, popularity:940.713e96, awareness:23.347e96, ratio:40.292, costPerAdvertisingFactor:535.158e24)
(wilson:50, advert:971, totalCost:62.208e33, advertisingFactor:111.866e6, popularity:1.188e99, awareness:29.330e96, ratio:40.493, costPerAdvertisingFactor:556.091e24)
(wilson:50, advert:972, totalCost:65.940e33, advertisingFactor:114.114e6, popularity:1.499e99, awareness:36.846e96, ratio:40.694, costPerAdvertisingFactor:577.842e24)
(wilson:50, advert:973, totalCost:69.897e33, advertisingFactor:116.408e6, popularity:1.893e99, awareness:46.288e96, ratio:40.897, costPerAdvertisingFactor:600.444e24)
(wilson:50, advert:974, totalCost:74.090e33, advertisingFactor:118.748e6, popularity:2.390e99, awareness:58.149e96, ratio:41.100, costPerAdvertisingFactor:623.930e24)
(wilson:50, advert:975, totalCost:78.536e33, advertisingFactor:121.135e6, popularity:3.017e99, awareness:73.049e96, ratio:41.305, costPerAdvertisingFactor:648.335e24)
(wilson:50, advert:976, totalCost:83.248e33, advertisingFactor:123.569e6, popularity:3.809e99, awareness:91.768e96, ratio:41.510, costPerAdvertisingFactor:673.694e24)
(wilson:50, advert:977, totalCost:88.243e33, advertisingFactor:126.053e6, popularity:4.809e99, awareness:115.284e96, ratio:41.717, costPerAdvertisingFactor:700.045e24)
(wilson:50, advert:978, totalCost:93.573e33, advertisingFactor:128.587e6, popularity:6.072e99, awareness:144.825e96, ratio:41.924, costPerAdvertisingFactor:727.427e24)
(wilson:50, advert:979, totalCost:99.150e33, advertisingFactor:131.171e6, popularity:7.665e99, awareness:181.937e96, ratio:42.133, costPerAdvertisingFactor:755.880e24)
```

❖ Budget < 1e45:

```
(wilson:115, advert:1370, totalCost:943.983e42, advertisingFactor:693.494e18, popularity:6.639e276, awareness:22.395e273, ratio:296.472, costPerAdvertisingFactor:1.361e24)
(wilson:116, advert:1362, totalCost:820.327e42, advertisingFactor:699.442e18, popularity:12.112e276, awareness:42.504e273, ratio:284.952, costPerAdvertisingFactor:1.173e24)
(wilson:117, advert:1354, totalCost:970.804e42, advertisingFactor:702.365e18, popularity:20.720e276, awareness:75.652e273, ratio:273.880, costPerAdvertisingFactor:1.382e24)
(wilson:115, advert:1371, totalCost:990.652e42, advertisingFactor:718.639e18, popularity:10.562e276, awareness:35.448e273, ratio:297.947, costPerAdvertisingFactor:1.379e24)
(wilson:116, advert:1363, totalCost:849.608e42, advertisingFactor:724.958e18, popularity:19.328e276, awareness:67.492e273, ratio:286.370, costPerAdvertisingFactor:1.172e24)
(wilson:117, advert:1355, totalCost:989.175e42, advertisingFactor:728.145e18, popularity:33.169e276, awareness:120.508e273, ratio:275.243, costPerAdvertisingFactor:1.358e24)
(wilson:116, advert:1364, totalCost:880.646e42, advertisingFactor:751.405e18, popularity:30.843e276, awareness:107.170e273, ratio:287.795, costPerAdvertisingFactor:1.172e24)
(wilson:116, advert:1365, totalCost:913.546e42, advertisingFactor:778.817e18, popularity:49.219e276, awareness:170.175e273, ratio:289.226, costPerAdvertisingFactor:1.173e24)
(wilson:116, advert:1366, totalCost:948.420e42, advertisingFactor:807.229e18, popularity:78.544e276, awareness:270.221e273, ratio:290.665, costPerAdvertisingFactor:1.175e24)
(wilson:116, advert:1367, totalCost:985.387e42, advertisingFactor:836.677e18, popularity:125.340e276, awareness:429.084e273, ratio:292.111, costPerAdvertisingFactor:1.178e24)
```

❖ Budget < 1e45, but we limit Wilson at level 50:

```
(wilson:50, advert:1365, totalCost:581.239e42, advertisingFactor:284.335e9, popularity:9.11e138, awareness:31.863e135, ratio:286.134, costPerAdvertisingFactor:2.044e33)
(wilson:50, advert:1366, totalCost:616.113e42, advertisingFactor:290.050e9, popularity:11.510e138, awareness:40.028e135, ratio:287.558, costPerAdvertisingFactor:2.124e33)
(wilson:50, advert:1367, totalCost:653.080e42, advertisingFactor:295.880e9, popularity:14.532e138, awareness:50.285e135, ratio:288.988, costPerAdvertisingFactor:2.207e33)
(wilson:50, advert:1368, totalCost:692.265e42, advertisingFactor:301.827e9, popularity:18.346e138, awareness:63.170e135, ratio:290.426, costPerAdvertisingFactor:2.294e33)
(wilson:50, advert:1369, totalCost:733.801e42, advertisingFactor:307.894e9, popularity:23.162e138, awareness:79.358e135, ratio:291.871, costPerAdvertisingFactor:2.383e33)
(wilson:50, advert:1370, totalCost:777.829e42, advertisingFactor:314.082e9, popularity:29.242e138, awareness:99.693e135, ratio:293.233, costPerAdvertisingFactor:2.477e33)
(wilson:50, advert:1371, totalCost:824.499e42, advertisingFactor:320.395e9, popularity:36.918e138, awareness:125.240e135, ratio:294.782, costPerAdvertisingFactor:2.573e33)
(wilson:50, advert:1372, totalCost:873.969e42, advertisingFactor:326.835e9, popularity:46.610e138, awareness:157.332e135, ratio:296.249, costPerAdvertisingFactor:2.674e33)
(wilson:50, advert:1373, totalCost:926.407e42, advertisingFactor:333.404e9, popularity:58.845e138, awareness:197.649e135, ratio:297.723, costPerAdvertisingFactor:2.779e33)
(wilson:50, advert:1374, totalCost:981.991e42, advertisingFactor:340.105e9, popularity:74.291e138, awareness:248.296e135, ratio:299.204, costPerAdvertisingFactor:2.887e33)
```

❖ Budget < 1e48:

```
(wilson:121, advert:1465, totalCost:207.849e45, advertisingFactor:133.841e21, popularity:94.164e306, awareness:198.138e303, ratio:475.245, costPerAdvertisingFactor:1.553e24)
(wilson:118, advert:1491, totalCost:898.539e45, advertisingFactor:134.862e21, popularity:22.439e306, awareness:41.508e303, ratio:540.590, costPerAdvertisingFactor:6.663e24)
(wilson:120, advert:1474, totalCost:338.509e45, advertisingFactor:136.467e21, popularity:73.374e306, awareness:147.657e303, ratio:496.920, costPerAdvertisingFactor:2.481e24)
(wilson:119, advert:1483, totalCost:565.579e45, advertisingFactor:138.477e21, popularity:53.276e306, awareness:102.536e303, ratio:519.584, costPerAdvertisingFactor:4.084e24)
(wilson:120, advert:1476, totalCost:379.691e45, advertisingFactor:138.803e21, popularity:179.769e306, awareness:381.791e303, ratio:470.858, costPerAdvertisingFactor:2.735e24)
(wilson:121, advert:1466, totalCost:219.682e45, advertisingFactor:138.872e21, popularity:152.645e306, awareness:319.602e303, ratio:477.610, costPerAdvertisingFactor:1.582e24)
(wilson:118, advert:1492, totalCost:952.371e45, advertisingFactor:139.841e21, popularity:36.034e306, awareness:66.327e303, ratio:543.280, costPerAdvertisingFactor:6.810e24)
(wilson:120, advert:1475, totalCost:358.500e45, advertisingFactor:141.566e21, popularity:118.572e306, awareness:237.432e303, ratio:499.393, costPerAdvertisingFactor:2.532e24)
(wilson:119, advert:1484, totalCost:599.354e45, advertisingFactor:143.621e21, popularity:85.825e306, awareness:164.363e303, ratio:522.169, costPerAdvertisingFactor:4.173e24)
(wilson:119, advert:1485, totalCost:635.156e45, advertisingFactor:148.957e21, popularity:138.260e306, awareness:263.470e303, ratio:524.766, costPerAdvertisingFactor:4.264e24)
```

❖ Budget < 1e48, but we limit Wilson at level 50:

```
(wilson:50, advert:1483, totalCost:562.920e45, advertisingFactor:2.976e12, popularity:8.038e150, awareness:15.641e147, ratio:513.921, costPerAdvertisingFactor:189.149e33)
(wilson:50, advert:1484, totalCost:596.696e45, advertisingFactor:3.036e12, popularity:10.148e150, awareness:19.649e147, ratio:516.478, costPerAdvertisingFactor:196.548e33)
(wilson:50, advert:1485, totalCost:632.497e45, advertisingFactor:3.097e12, popularity:12.812e150, awareness:24.684e147, ratio:519.047, costPerAdvertisingFactor:204.236e33)
(wilson:50, advert:1486, totalCost:670.447e45, advertisingFactor:3.159e12, popularity:16.175e150, awareness:31.009e147, ratio:521.630, costPerAdvertisingFactor:212.224e33)
(wilson:50, advert:1487, totalCost:710.674e45, advertisingFactor:3.223e12, popularity:20.421e150, awareness:38.955e147, ratio:524.225, costPerAdvertisingFactor:220.525e33)
(wilson:50, advert:1488, totalCost:753.314e45, advertisingFactor:3.287e12, popularity:25.782e150, awareness:48.938e147, ratio:526.833, costPerAdvertisingFactor:229.151e33)
(wilson:50, advert:1489, totalCost:798.513e45, advertisingFactor:3.353e12, popularity:32.550e150, awareness:61.478e147, ratio:529.454, costPerAdvertisingFactor:238.114e33)
(wilson:50, advert:1490, totalCost:846.424e45, advertisingFactor:3.421e12, popularity:41.094e150, awareness:77.232e147, ratio:532.088, costPerAdvertisingFactor:247.428e33)
(wilson:50, advert:1491, totalCost:897.209e45, advertisingFactor:3.490e12, popularity:51.881e150, awareness:97.022e147, ratio:534.735, costPerAdvertisingFactor:257.106e33)
(wilson:50, advert:1492, totalCost:951.042e45, advertisingFactor:3.560e12, popularity:65.500e150, awareness:121.884e147, ratio:537.396, costPerAdvertisingFactor:267.162e33)
```

❖ These cases show that **Wilson becomes extremely important in big-budget situations.**
Wilson is usually good investment in late phases.

- This optimizer works best in the situations that we have large budget, like after accepting investment offer.
- In round 3+, we usually improve the divisions continuously with relatively small budget (profit of a few cycles). In this case, we should buy Wilson as soon as we can afford it.

13.5 Sample code

[corporationOptimizer.ts](#): `optimizeWilsonAndAdvert()`

14. Demand – Competition

14.1 Usage

- They are used for calculating MaxSalesVolume of materials and products.
- “Market Research – Demand” grants access to Demand data.
- “Market Data – Competition” grants access to Competition data.

14.2 Material

- Each material has its own demandBase, demandRange, competitionBase, competitionRange, maxVolatility. Both Demand and competition start at their bases and are always in their respective ranges.
- This is non-intuitive: demand and competition are NOT used for calculating marketPrice.
- During the START state, the game calculates 6 variables:

```
const priceVolatility: number = (Math.random() * this.maxVolatility) / 300;  
const priceChange: number = 1 + priceVolatility;  
const compVolatility: number = (Math.random() * this.maxVolatility) / 100;  
const compChange: number = 1 + compVolatility;  
const dmdVolatility: number = (Math.random() * this.maxVolatility) / 100;  
const dmdChange: number = 1 + dmdVolatility;
```

- priceChange, compChange and dmdChange are the amount of marketPrice, competition and demand changed in the next steps.
- After that, it randomizes twice:
 - First: Math.random() < 0.5. If yes, increase competition and marketPrice. If not, decrease them.
 - Second: Math.random() < 0.5. If yes, increase demand and marketPrice. If not, decrease them.

14.3 Product

- Initial values are set when the product is finished. Check the next [section](#) for the formulas.
- During the START state, the game decreases demand and increases competition of the product.
 - Amount of change:
$$\text{AmountOfChange} = \text{Random}(0,3) * 0.0004$$
 - This amount is multiplied by 3 if the industry is Pharmaceutical, Software or Robotics.
- Demand's minimum value is 0.001. Competition's maximum value is 99.99.

15. Product

15.1 Overview

- The product industry is much better than the material industry in the late phases because we can sell products at ridiculously high prices.

```
Tobacco-00006-1.38553e+72: 185.808e6 (0.134/s)
Effective rating: 44.354e12
Est. Production Cost: $2.964e3
Est. Market Price: $14.818e3
Discontinue Sell (18.581e6/MAX) @$35.010e81
```

- Market-TA2 automatically sets the optimal prices for products. Check this [section](#) to see how to implement a custom Market-TA2 script. [Implementing a custom Market-TA2 script is the best optimization in round 3+.](#)
- Products need to be developed before being produced. We can put multiple products in the development queue, but only one product can be developed at a time.
- There is a limit on how many products a division can have. The default limit is 3. There are 2 researches that increase that limit. However, those researches are nearly useless because, in most cases, there is no point in increasing the maximum number of products. Check this [part](#) for details. When we reach the limit, we need to discontinue a product before developing a new one.
- We need to continuously develop new products. New products are almost always better than the old ones and generate much more profit.

```
Tobacco-00000-1.00000e+9: 2.193e6 (0.007/s)
Effective rating: 29.592e3
Est. Production Cost: $2.912e3
Est. Market Price: $14.562e3
Discontinue Sell (219.317e3/MAX) @$1.699e15
```

```
Tobacco-00001-1.94371e+9: 2.193e6 (0.007/s)
Effective rating: 496.655e3
Est. Production Cost: $2.912e3
Est. Market Price: $14.562e3
Discontinue Sell (219.317e3/MAX) @$745.877e15
```

```
Tobacco-00002-5.67940e+12: 2.193e6 (0.007/s)
Effective rating: 3.758e6
Est. Production Cost: $2.912e3
Est. Market Price: $14.562e3
Discontinue Sell (219.317e3/MAX) @$50.684e18
```

```

Tobacco-00003-3.12975e+17: 2.193e6 (0.007/s)
Effective rating: 12.411e6
Est. Production Cost: $2.912e3
Est. Market Price: $14.562e3
Discontinue Sell (219.317e3/MAX) @$630.864e18

Tobacco-00004-3.41195e+21: 2.193e6 (0.007/s)
Effective rating: 52.978e6
Est. Production Cost: $2.912e3
Est. Market Price: $14.562e3
Discontinue Sell (219.317e3/MAX) @$10.934e21

```

- The product's markup and effective rating are extremely important because they are part of [MaxSalesVolume](#)'s calculation.
- The product's effective rating is based on the product's rating and the input material's quality. Check this [section](#) to see how input material's quality affects the product's rating and effective rating. This is why we need a support division that produces high-quality material for the product division.
- The product's markup and rating are based on:
 - `CreationJobFactors[JobName]`. More about this in the next part.
 - RP. This is why we should stock up on RP.
 - `ResearchFactor`. It is the industry's `scienceFactor`.
 - Design investment and advertising investment. Game UI shows them as "Design investment" and "Marketing investment". These two investments are not too important because their exponents in the formulas are very low. It's fine to use only 1% of current funds for them.
- Office's upgrade and employee stats' upgrades are very important for products because they increase employee production. High employee production means high `CreationJobFactors` and RP. Those upgrades and products create a powerful loop: More upgrades → Better product → Higher profit → More upgrades.
- Office setup is important to efficiently develop new products. Check this [part](#) for advice on how to set up the office.

15.2 Formula

- `CreationJobFactors[JobName]` are values accumulated over the time that product was developed. `DevelopmentProgress` starts at 0. In each cycle:
 - Total employee production:

$$TotalEmployeeProd = OperationsProd + EngineerProd + ManagementProd$$

- Management factor:

$$ManagementFactor = 1 + \frac{ManagementProd}{1.2 * TotalEmployeeProd}$$

- Product development multiplier:

$$ProductDevelopmentMultiplier = ((EngineerProd)^{0.34} + (OperationsProd)^{0.2}) * ManagementFactor$$

- Progress:

$$Progress = 0.01 * ProductDevelopmentMultiplier$$

- Development progress:

$$DevelopmentProgress = DevelopmentProgress + Progress$$

- CreationJobFactors[JobName]:

$$CreationJobFactors[JobName] = CreationJobFactors[JobName] + \frac{\{EmployeeJob\}Prod * Progress}{100}$$

- When DevelopmentProgress reaches 100, product is finished.

- Define:

$$A = CreationJobFactors[Engineer]$$

$$B = CreationJobFactors[Management]$$

$$C = CreationJobFactors[RnD]$$

$$D = CreationJobFactors[Operations]$$

$$E = CreationJobFactors[Business]$$

$$TotalCreationJobFactors = A + B + C + D + E$$

- {JobName}Ratio:

$$EngineerRatio = \frac{A}{TotalCreationJobFactors}$$

$$ManagementRatio = \frac{B}{TotalCreationJobFactors}$$

$$RnDRatio = \frac{C}{TotalCreationJobFactors}$$

$$OperationsRatio = \frac{D}{TotalCreationJobFactors}$$

$$BusinessRatio = \frac{E}{TotalCreationJobFactors}$$

- Design investment multiplier:

$$DesignInvestMult = 1 + \frac{(DesignInvestment)^{0.1}}{100}$$

- Science multiplier:

$$ScienceMult = 1 + \frac{(RP)^{ResearchFactor}}{800}$$

- Balance multiplier:

$$BalanceMult = 1.2 * EngineerRatio + 0.9 * ManagementRatio + 1.3 * RnDRatio + 1.5 * OperationsRatio + BusinessRatio$$

- Total multiplier:

$$TotalMult = BalanceMult * DesignInvestMult * ScienceMult$$

- Product's quality:

$$TotalMult * (0.1 * A + 0.05 * B + 0.05 * C + 0.02 * D + 0.02 * E)$$

- Product's performance:

$$TotalMult * (0.15 * A + 0.02 * B + 0.02 * C + 0.02 * D + 0.02 * E)$$

- Product's durability:

$$TotalMult * (0.05 * A + 0.02 * B + 0.08 * C + 0.05 * D + 0.05 * E)$$

- Product's reliability:

$$TotalMult * (0.02 * A + 0.08 * B + 0.02 * C + 0.05 * D + 0.08 * E)$$

- Product's aesthetics:

$$TotalMult * (0.08 * B + 0.05 * C + 0.02 * D + 0.1 * E)$$

- Product's features:

$$TotalMult * (0.08 * A + 0.05 * B + 0.02 * C + 0.05 * D + 0.05 * E)$$

- Product's rating:

❖ If an industry produces product, it has its own `RatingWeights` for its product. `RatingWeights` contains coefficients of 6 stats: quality, performance, durability, reliability, aesthetics, features. For example: Tobacco's `RatingWeights`:

- Quality's coefficient: 0.7.
- Durability's coefficient: 0.1.
- Aesthetics' coefficient: 0.2.

❖ `RatingWeights` is `industryData.product.ratingWeights`.

❖ Formula:

$$ProductRating = \sum_{i=1}^6 ProductStat_i * StatCoefficient_i$$

- Advertising investment multiplier:

$$AdvertInvestMult = 1 + \frac{(AdvertisingInvestment)^{0.1}}{100}$$

- Business-Management ratio:

$$BusinessManagementRatio = Max \left(BusinessRatio + ManagementRatio, \left(\frac{1}{TotalCreationJobFactors} \right) \right)$$

- Product's markup:

$$ProductMarkup = \frac{100}{AdvertInvestMult * (ProductQuality + 0.001)^{0.65} * BusinessManagementRatio}$$

- Product's demand:

$$Demand = \begin{cases} \text{Min}(100, \text{AdvertInvestMult} * (100 * (\text{Popularity}/\text{Awareness}))), & \text{Awareness} \neq 0 \\ 20, & \text{Awareness} = 0 \end{cases}$$

- Product's competition:

$$Competition = \text{Random}(0,70)$$

- Product's size:

❖ It's `product.size`.

❖ Formula:

$$ProductSize = \sum_{i=1}^{NumberOfInputMaterials} InputMaterialSize_i * InputMaterialCoefficient_i$$

15.3 Approximation value of product markup

In order to calculate product markup, we need:

- `CreationJobFactors[JobName]`
- `RP`
- `ResearchFactor`
- `DesignInvestment`
- `AdvertisingInvestment`

Product markup is calculated when product is finished. At that time, there is one thing that we cannot get: `CreationJobFactors[JobName]`, because there is not any NS API to query it. There are 2 approaches for this problem:

- Manually record them. This means that we simulate `product.creationJobFactors`. This approach is simple, but it has a big problem: if we miss any cycle, the data is invalid.
- Calculate them directly. Product's stats are public data, so with above formulas, we have a system of 6 functions with only 5 variables. We can use [Ceres Solver](#) to find its solution.

15.4 Sample code

[corporationUtils.ts](#): `getProductMarkup()`

16. Optimal selling price – Market-TA2

16.1 Market price and markup limit

- Market price:
 - Material: `material.marketPrice`.
 - Product: `product.productionCost`. This value is based on `ProductMarketPriceMult`, input materials' `MarketPrice` and `Coefficient`.

$n = \text{Number of input materials}$

$\text{ProductMarketPriceMult} = 5$

$$\text{ProductMarketPrice} = \text{ProductMarketPriceMult} * \sum_{i=1}^n \text{MaterialMarketPrice}_i * \text{MaterialCoefficient}_i$$

- Markup limit:

- Material:

$$\text{MaterialMarkupLimit} = \frac{\text{MaterialQuality}}{\text{MaterialMarkup}}$$

- Product:

$$\text{ProductMarkupLimit} = \frac{\text{Max}(\text{ProductEffectiveRating}, 0.001)}{\text{ProductMarkup}}$$

16.2 Maximize sales volume

- Define:

$$\text{ExpectedSalesVolume} = \frac{\text{ProducedUnits}}{10}$$

- This means we want to sell all produced units.
- In cycle's SALE state, game calculates `MaxSalesVolume` of material/product. If we set price too high, `MaxSalesVolume` is penalized. In order to maximize profit, we have to set the highest possible price while `MaxSalesVolume` is still equal to `ExpectedSalesVolume`. This is what Market-TA2 does.
- Calculation of material and product is pretty similar, so I will call them “item” and use 1 formula.
- `MaxSalesVolume` is the product of 7 multipliers:
 - Item multiplier:

- ❖ Material:

$$\text{ItemMultiplier} = \text{MaterialQuality} + 0.001$$

- ❖ Product:

$$\text{ItemMultiplier} = 0.5 * (\text{ProductEffectiveRating})^{0.65}$$

- Business factor:

$$BusinessProduction = 1 + office.employeeProductionByJob["Business"]$$

$$BusinessFactor = (BusinessProduction)^{0.26} + \left(\frac{BusinessProduction}{1000} \right)$$

- Advert factor:

$$AwarenessFactor = (Awareness + 1)^{IndustryAdvertisingFactor}$$

$$PopularityFactor = (Popularity + 1)^{IndustryAdvertisingFactor}$$

$$RatioFactor = \begin{cases} \text{Max} \left(0.01, \frac{Popularity + 0.001}{Awareness} \right), & Awareness \neq 0 \\ 0.01, & Awareness = 0 \end{cases}$$

$$AdvertFactor = (AwarenessFactor * PopularityFactor * RatioFactor)^{0.85}$$

- Market factor

$$MarketFactor = \text{Max} \left(0.1, \frac{Demand * (100 - Competition)}{100} \right)$$

- Corporation's upgrade bonus: SalesBots bonus.
- Division's research bonus: this is always 1. Currently there is not any research that increases sales bonus.
- MarkupMultiplier: initialize with 1.
 - ❖ SellingPrice is the selling price that you set.
 - ❖ If we set SellingPrice to 0 or a negative number, MarkupMultiplier is 10^{12} (check the formula below). With an extremely high MarkupMultiplier, we can sell all units, regardless of other factors. This is the fastest way to discard stored units.
 - ❖ If (SellingPrice > MarketPrice):
 - If (SellingPrice > MarketPrice + MarkupLimit):

$$MarkupMultiplier = \left(\frac{MarkupLimit}{SellingPrice - MarketPrice} \right)^2$$

- ❖ If (SellingPrice <= MarketPrice):

$$MarkupMultiplier = \begin{cases} \frac{MarketPrice}{SellingPrice}, & SellingPrice > 0 \\ 10^{12}, & SellingPrice \leq 0 \end{cases}$$

16.3 Optimal selling price

- As we can see with previous part, MarkupMultiplier is basically a penalty modifier if we set SellingPrice greater than (MarketPrice + MarkupLimit), and we will always do this. This means we need to find out highest possible SellingPrice while MaxSalesVolume is still equal to ExpectedSalesVolume.
- This is the reason why we should not bother with Market-TA1. It simply sets SellingPrice = MarketPrice + MarkupLimit. This means Market-TA1 sets a "safe" SellingPrice for us, it

guarantees that we won't be penalized due to too high price. However, this "safe" SellingPrice is too low, and we can find a much higher SellingPrice.

- Formula:

- Define:

$$M = \text{ItemMultiplier} * \text{BusinessFactor} * \text{AdvertFactor} * \text{MarketFactor} * \text{SaleBotsBonus} * \text{ResearchBonus}$$

- We want MaxSalesVolume to equal ExpectedSalesVolume:

$$\text{MaxSalesVolume} = \text{ExpectedSalesVolume}$$

≡

$$M * \left(\frac{\text{MarkupLimit}}{\text{SellingPrice} - \text{MarketPrice}} \right)^2 = \text{ExpectedSalesVolume}$$

≡

$$\frac{\text{MarkupLimit}}{\text{SellingPrice} - \text{MarketPrice}} = \sqrt{\frac{\text{ExpectedSalesVolume}}{M}}$$

≡

$$\text{SellingPrice} = \frac{\text{MarkupLimit} * \sqrt{M}}{\sqrt{\text{ExpectedSalesVolume}}} + \text{MarketPrice}$$

- In order to use this formula, we need MarkupLimit. With product, we need ProductMarkup to calculate MarkupLimit, but ProductMarkup is inaccessible via NS API. We have two solutions:

- Calculate approximation value. Check previous section to see how to do this. The sample code uses this solution.

- Calculate MarkupLimit directly:

- ❖ Set SellingPrice to a very high value, it must be so high that we cannot sell all produced units ($\text{MaxSalesVolume} < \text{ExpectedSalesVolume}$). This forces the game applies the penalty modifier that contains MarkupLimit.

- ❖ Wait for 1 cycle to get ActualSalesVolume. It's product.actualSellAmount and material.actualSellAmount.

- ❖ Use ActualSalesVolume in place of ExpectedSalesVolume in previous formula:

$$\text{MarkupLimit} = (\text{SellingPrice} - \text{MarketPrice}) * \sqrt{\frac{\text{ActualSalesVolume}}{M}}$$

- ❖ Calculate ProductMarkup from MarkupLimit, save ProductMarkup to reuse later. ProductMarkup never changes.

16.4 Sample code

[corporationUtils.ts](#): `getOptimalSellingPrice()`

17. Financial statement

17.1 Total assets

- TotalAssets is the sum of:
 - Funds.
 - With each division:
 - ❖ Division's RecoupableValue. It's half of the sum of:
 - Industry's starting cost.
 - With each city that division has expanded to (exclude Sector-12):
 - Office's initial cost.
 - Warehouse's initial cost.
 - ❖ Output material: `material.stored * material.averagePrice`.
 - ❖ Product: `product.stored * product.productionCost`.
- This value is kept track by TotalAssets and PreviousTotalAssets.
- Funds is increased/decreased by gainFunds/loseFunds for each "action" (buying tea, throwing party, buying upgrade, etc.). Each action is either "long-term" (FundsSourceLongTerm) or "short-term" (FundsSourceShortTerm). If an action is "long-term", it modifies totalAssets.

```
if (LongTermFundsSources.has(source)) {  
    this.totalAssets += amt;  
}  
this.funds += amt;
```

- FundsSourceLongTerm and FundsSourceShortTerm are in FundsSource.ts.

17.2 Valuation

- Cycle's valuation:
 - AssetDelta:
$$AssetDelta = \frac{TotalAssets - PreviousTotalAssets}{10}$$
 - Pre-IPO:
 - ❖ If AssetDelta is greater than 0, it's used for calculating valuation.
 - ❖ Formula:

$$Valuation = \left(10^{10} + \frac{Funds}{3} + AssetDelta * 315000\right) * \left(\sqrt[12]{1.1}\right)^{NumberOfOfficesAndWarehouses}$$

- ❖ Valuation is rounded down to nearest million.
- Post-IPO:
 - ❖ AssetDelta is affected by DividendRate.

$$AssetDelta = AssetDelta * (1 - DividendRate)$$

❖ Formula:

$$Valuation = (Funds + AssetDelta * 85000) * \left(\sqrt[12]{1.1}\right)^{NumberOfOfficesAndWarehouses}$$

- Minimum value of valuation is 10^{10} .
- Valuation is multiplied by CorporationValuation. Many BitNodes cripple Corporation via this multiplier.
- Corporation's valuation is the mean of last 10 cycles' valuations.
- Bribing faction for reputation is unlocked when corporation's valuation is greater than or equal to $100e12$. Exchange rate: $1e9/reputation$.

17.3 Investment offer

- There are 4 investment rounds.
- Each round has its own FundingRoundShares and FundingRoundMultiplier.

$$FundingRoundShares = [0.1, 0.35, 0.25, 0.2]$$

$$FundingRoundMultiplier = [3, 2, 2, 1.5]$$

- Formula:

$$Offer = CorporationValuation * FundingRoundShares * FundingRoundMultiplier$$

- Analyses:
 - Offer depends on Funds, AssetDelta and NumberOfOfficesAndWarehouses.
 - ❖ Funds are usually depleted to improve divisions.
 - ❖ NumberOfOfficesAndWarehouses is the exponent of the multiplier, it can be increased by creating [dummy division](#). It is an easy way to get higher offer in round 3+, when we have enough funds to do that.
 - ❖ AssetDelta is multiplied by 315000, so it is the main source of offer.
 - Assuming that we can sell all produced units and not buy more boost materials, AssetDelta is the delta of funds, and the delta of funds is the profit. This is why we try our best to improve profit.

17.4 Dividend

- DividendTax depends on CorporationSoftcap. In BN3, CorporationSoftcap is 1.

$$DividendTax = 1 - CorporationSoftcap + 0.15$$

- ShadyAccounting reduces DividendTax by 0.05.
- GovernmentPartnership reduces DividendTax by 0.1.
- Formula:

$$TotalDividends = DividendRate * (Revenue - Expenses) * 10$$

$$Dividend = \left(OwnedShares * \frac{TotalDividends}{TotalShares} \right)^{1-DividendTax}$$

- Retained earning:

$$RetainedEarning = (1 - DividendRate) * (Revenue - Expenses) * 10$$

- Dividend is added to player's money. Retained earning is added to corporation's funds. This means if we increase DividendRate, corporation's valuation is dwindled.

17.5 Shares

- Self-fund:
 - Cost 150b.
 - Total shares: 1b.
 - Initial owned shares: 1b.
- Use seed money:
 - Does not cost money.
 - Total shares: 1.5b.
 - Initial owned shares: 1b.
- In each investment round, investors take a percentage of initial owned shares. The percentage of each round is in FundingRoundShares.
- If your corporation is self-funded and you sell CEO position, you only need 50b to create next corporation.
- TargetSharePrice:

$$TargetSharePrice = \frac{CorporationValuation * \left(0.5 + \sqrt{\frac{OwnedShares}{TotalShares}} \right)}{TotalShares}$$

- When corporation goes public, the initial share price is TargetSharePrice.
- Share price is updated in START state.

$$SharePrice = \begin{cases} SharePrice * (1 + Math.random() * 0.01), & SharePrice \leq TargetSharePrice \\ SharePrice * (1 - Math.random() * 0.01), & SharePrice > TargetSharePrice \end{cases}$$

- Minimum share price is 0.01.
- Issue new shares:
 - Maximum number of new shares is 20% of total shares.
 - The number of new shares issued must be a multiple of 10 million.
 - New share price:

$$NewSharePrice = \frac{CorporationValuation * \left(0.5 + \sqrt{\frac{OwnedShares}{TotalShares + NewShares}} \right)}{TotalShares}$$

- Profit:

$$Profit = \frac{NewShares * (SharePrice + NewSharePrice)}{2}$$

- Profit is added to corporation's funds.
- DefaultCooldown is 4 hours.
- Cooldown:

$$Cooldown = DefaultCooldown * \frac{TotalShares}{10^9}$$

- Part of the new shares are added to InvestorShares. The remaining ones are added to IssuedShares.
- ❖ MaxPrivateShares:

$$MaxPrivateShares = \frac{NewShares}{2} * \frac{InvestorShares}{TotalShares}$$

- ❖ PrivateShares is randomized between 0 and MaxPrivateShares, rounded to nearest 10 million.
- ❖ InvestorShares:

$$InvestorShares = InvestorShares + PrivateShares$$

- ❖ IssuedShares:

$$IssuedShares = IssuedShares + NewShares - PrivateShares$$

- Sell shares:
 - We cannot sell all our shares.
 - We cannot sell more than 10^{14} shares at a time.
 - Cooldown is 1 hour.
 - Sold shares are added to IssuedShares.
- Buy back shares:
 - We can only buy back shares that were issued. The shares that were owned by government (when we use seed money) and investors cannot be bought back.
 - Shares must be bought back at a 10% premium over the market price.
 - We cannot use corporation's funds to buy back shares. They must be bought with our money.
 - We cannot buy back more than 10^{14} shares at a time.
- Sold/bought back shares are processed in multiple "iterations".
 - Number of shares processed each iteration is shareSalesUntilPriceUpdate. Default value is 10^6 .
 - Share price is recalculated each iteration.

$$TargetSharePrice = \frac{CorporationValuation * \left(0.5 + \sqrt{\frac{OwnedShares - ProcessedShares}{TotalShares}}\right)}{TotalShares}$$

$$SharePrice = \begin{cases} SharePrice * 1.005, & SharePrice \leq TargetSharePrice \\ SharePrice * 0.995, & SharePrice > TargetSharePrice \end{cases}$$

18. Miscellany

18.1 Corporation's state

- Corporation continuously transitions between 5 states: START → PURCHASE → PRODUCTION → EXPORT → SALE → START. 1 cycle of these transitions takes 10 seconds. If you have enough bonus time, it takes 1 second.
- START:
 - Division:
 - ❖ Office:
 - Calculate: energy, morale, total experience, salary, employee production by jobs.
 - Set employees' jobs: copy data from `employeeNextJobs` to `employeeJobs`.
 - ❖ Material market: update demand, competition, `marketPrice`.
 - ❖ Product market: decrease demand and increase competition.
 - Calculate corporation's financial statements: revenue, expenses, profit, dividend, total assets, valuation, share price.
- PURCHASE:
 - Buy input materials.
 - If we have unlocked "Smart Supply", it's used for calculating optimal quantity of input material units.
- PRODUCTION:
 - Produce output materials and products.
 - If new product's `developmentProgress` is greater than or equal to 100, it's finished. New finished product's stats are calculated in this state.
- EXPORT: export output materials.
- SALE:
 - Sell output materials and product.
 - If we have unlocked "Market-TA1" / "Market-TA2", it's used for calculating selling price.
- Special cases: RP and salary expense are increased in 4 states: PURCHASE, PRODUCTION, EXPORT, SALE.

18.2 Import and export

- Importing/exporting material is done in EXPORT state.
- EXPORT state is before SALE state. It means you sell the material units remained after being exported.
- Export string can use "MAX", "EINV", "IINV", "EPROD" and "IPROD". Read the description in export dialog for the meaning of these values.

- Optimal value for export string: $(IPROD + IINV/10) * (-1)$. For example: export “Chemicals” from Chemical division to Agriculture division:
 - Agriculture division needs 100 Chemicals/s and has 700 Chemicals in warehouse.
 - ❖ $IPROD = -100$ (“Consumption is negative production”)
 - ❖ $IINV = 700$
 - “Export” is expressed by number of units per second, so we want to export:

$$\left(100 - \frac{700}{10}\right) = \left(-100 + \frac{700}{10}\right) * (-1) = \left(IPROD + \frac{IINV}{10}\right) * (-1)$$
- Export route is FIFO. You can remove an export route by using `cancelExportMaterial` NS API.
- Export data is in `material.exports`. Sample code for getting all export data:

[corporationUtils.ts](#): `getExportRoutes()`

18.3 Use mathematical library

- I use the JavaScript ports of these libraries. They don’t support JavaScript.
- I use the default setting without any tuning. If you tune their parameters properly, their accuracy and performance will be improved.

18.3.1 Ceres Solver

- Quote from <http://ceres-solver.org/>

Ceres Solver is an open source C++ library for modeling and solving large, complicated optimization problems. It can be used to solve Non-linear Least Squares problems with bounds constraints and general unconstrained optimization problems.

- JavaScript port: <https://github.com/Pterodactylus/Ceres.js>
- We can use it to solve the non-linear systems in these cases:
 - Case 1: Find the optimal quantities of boost materials.
 - Case 2: Find `CreationJobFactors[JobName]` when calculating product markup.
- The accuracy and performance are acceptable, so we employ it case 2. We don’t use it in case 1 because there is another optimal solution, it’s better in both accuracy and performance.
- Quick test for case 2 shows that the accuracy is pretty good:
 - `creationJobFactors`:
 - ❖ Business: 420.103620358641
 - ❖ Engineer: 29666.47672073447
 - ❖ Management: 40466.091598191015
 - ❖ Operations: 25760.399443536793
 - Solver’s result:
 - ❖ Business: 420.08121628008024

- ❖ Engineer: 29664.894610028987
- ❖ Management: 40463.933544920386
- ❖ Operations: 25759.025643594476

18.3.2 ALGLIB

- Quote from <https://www.alglib.net/>

ALGLIB is a cross-platform numerical analysis and data processing library. It supports five programming languages (C++, C#, Java, Python, Delphi) and several operating systems (Windows and POSIX, including Linux). ALGLIB features include:

- *Data analysis (classification/regression, statistics)*
- *Optimization and nonlinear solvers*
- *Interpolation and linear/nonlinear least-squares fitting*
- *Linear algebra (direct algorithms, EVD/SVD), direct and iterative linear solvers*
- *Fast Fourier Transform and many other algorithms*
- JavaScript port: <https://github.com/Pterodactylus/Alglib.js>
- We can use it to find the approximation of optimal quantities of boost materials.
- The solution that we get when using this library is not really good. It has lower accuracy and run much slower than Ceres Solver.
- Run them with $S = 5250$:
 - Optimal solution:
 - ❖ [10518.092105263155, 11742.32456140351, 528368.4210526316, 1703.6184210526312]
 - Ceres Solver:
 - ❖ [10518.092105294003, 11742.324561437803, 528368.4210509873, 1703.6184210587894]
 - ALGLIB:
 - ❖ [10517.862063548288, 11740.824106170172, 528402.7972848249, 1703.500721701672]
- Run them 200 times:
 - Optimal solution: 0.3-0.5 ms
 - Ceres Solver: 380-400 ms
 - ALGLIB: 32000-34000 ms

18.4 Noodles trick

There is a place called “Noodle Bar” in New Tokyo. After going there, there is a button that says “Eat noodles”. Eating noodles gives you multiple benefits. Each benefit is miniscule, but you can press that button programmatically. With Corporation, the benefit is:

```
Player.corporation.gainFunds(Player.corporation.revenue * 0.000001, "glitch in reality");
```

In previous versions, the revenue is multiplied by 0.01. If you do it fast enough, it will raise the investment offer by a considerable amount. Nowadays, this trick is useless.

18.5 "sudo.Assist" research

This research exists in `ResearchMap.ts`, but it's unused in `BaseResearchTree.ts`. If you use it in NS API, it will throw error.

```
"sudo.Assist": new Research({
  name: "sudo.Assist",
  cost: 15e3,
  desc: "Develop a virtual assistant AI to handle and manage administrative issues for your corporation.",
})
```

18.6 Dummy division

- Dummy division is the division that you create only to increase the [valuation](#) and the [investment offer](#).
- Use Restaurant industry for dummy division. Its starting cost is only 10e9. Spring Water's starting cost is also 10e9, but it's a newbie trap, so it's best not to touch it.
- For dummy division, you only need to expand to 6 cities and buy 6 warehouses. Don't invest in other things (Warehouse/Office/Advert upgrades).
- If you start round 3 with large budget, you can create dummy divisions immediately to simplify the script's logic. It's not optimal but still acceptable. However, if the budget is tight, you should not create them at the start of round 3. After completing first product, profit will be high enough to get enough funds to create them without any problem.

19. Standard strategy

19.1 How to use this section

- This section shows you a good strategy to help you create a profitable corporation. This strategy is specialized for BN3. You can devise your own strategy by following the [general advices](#).
- Always expand to 6 cities and maintain your employees' energy and morale by buying tea and throwing party. I won't repeat this part.
- "Buying tea + Throwing party" script is mandatory. Doing them manually through UI is too tedious. Sample code:

[corporationUtils.ts](#): `buyTeaAndThrowParty()`

- Warehouse starts at level 1, so when I say "Warehouse level 6", it means upgrading warehouse 5 times. All other things start at level 0, "X level 9" means upgrading it 9 times.
- **All boost materials' numbers are the total quantities of material units after buying.** For example: when I say "AI Cores: 8556", that means you need 8556 AI Cores in the warehouse after buying, not buy more 8556 AI Cores.
- Do not use "Bulk Purchase", it requires paying upfront. Buying boost materials per second does not need funds, you can go into debt. In round 1 and 2, you need to use the funds to upgrade / expand your divisions and go into debt at the end of the round after buying boost materials.
- "Custom Smart Supply" means that you need to write script that simulates "Smart Supply" feature. You should write it as soon as possible. "Smart Supply" costs 25b, and 25b is huge in round 1.
- Use API (`upgradeOfficeSize`) to upgrade office. API gives you granular control over office size. You cannot do that through UI.
- The strategies for round 1 and round 2 have been tested at least 200 times in [headless mode](#). I will show you their means and medians, but don't be surprised if your number is different from mean/median, there are always [randomized factors](#). You should expect the number in the form "Offer: xxx-yyy".
 - Round 1 (Custom Smart Supply): Offer: 540-560b
 - ❖ Mean: 551.164b
 - ❖ Median: 551.934b
 - Round 1 (Without custom Smart Supply): Offer: 335-346b
 - ❖ Mean: 340.413b
 - ❖ Median: 340.112b
 - Round 2: Offer: 14.145-14.871t
 - ❖ Mean: 14.521t
 - ❖ Median: 14.53t

19.2 Round 1

- Create Agriculture division.
- Upgrade office size from 3 to 4. Set 4 employees to R&D and wait until RP is at least 55.
- Switch to Operations (1) + Engineer (1) + Business (1) + Management (1) before buying boost materials.

19.2.1 Custom Smart Supply

- Upgrade:
 - Smart Storage level 6
 - Warehouse level 6
 - Advert level 2
- Warehouse size: 960
- Boost materials
 - AI Cores: 1733
 - Hardware: 1981
 - Real Estate: 106686
 - Robots: 0

19.2.2 Without custom Smart Supply

- Upgrade:
 - Smart Storage level 3
 - Warehouse level 4
 - Advert level 2
- Warehouse size: 520
- Boost materials:
 - AI Cores: 777
 - Hardware: 919
 - Real Estate: 60794
 - Robots: 0

19.3 Round 2

You must have at least 490b for this strategy. It's not a problem if you follow the "Custom Smart Supply" strategy above. If you don't have enough funds, you can customize the strategy easily by following this [part](#). Do not waste time on waiting for more funds.

19.3.1 Phase 1

- Buy “Export”.
- Upgrade Agriculture division:
 - Upgrade office size from 4 to 8. Set all employees to R&D.
 - Warehouse level 16.
 - Advert level 8.
- Create Chemical division:
 - Do not upgrade office, initial size (3) is fine. Set 3 employees to R&D.
 - Warehouse level 2.
 - Do not buy Advert.
- Setup export routes between Agriculture division and Chemical division with the optimal [export string](#):
 - Plants from Agriculture to Chemical.
 - Chemicals from Chemical to Agriculture.
- Upgrade:
 - Smart Storage level 25.
 - ❖ Agriculture: warehouse size: 5600
 - ❖ Chemical: warehouse size: 700
 - Smart Factories level 20.
 - Do not buy Wilson in this round.
- **Wait for RP:** Wait until Agriculture and Chemical have at least ~700RP and ~390RP respectively. They will have those amounts of RP at approximately the same time. You can wait for more if you want, but it's not mandatory.

19.3.2 Phase 2

- After you have enough RP, set employees:
 - Agriculture:
 - ❖ Operations: 3
 - ❖ Engineer 1
 - ❖ Business: 2
 - ❖ Management: 2
 - Chemical
 - ❖ Operations: 1
 - ❖ Engineer 1
 - ❖ Business: 1

- Buy boost materials:
 - Agriculture:
 - ❖ AI Cores: 8556
 - ❖ Hardware: 9563
 - ❖ Real Estate: 434200
 - ❖ Robots: 1311
 - Chemical
 - ❖ AI Cores: 1717
 - ❖ Hardware: 3194
 - ❖ Real Estate: 54917
 - ❖ Robots: 54

19.4 Round 3+

19.4.1 Overview

- Round 3+ is much more complicated than previous rounds. You must read this [part](#) and make sure that you understand everything in it before using the advices below.
- Do not blindly apply the numbers in the advices. Round 3+ is more “freestyle” than previous rounds. I only give you those numbers as recommendation. Make sure to understand the reasoning behind them.
- Do not blindly mix and match any number in the advices with your custom strategy. They are tuned to work together. Only do that if you understand the mechanism.
- The strategy is optimized to reach profit of $1e90/s$ as soon as possible.
- If you have $11t+$ after round 2, it takes ~ 370 cycles to hit $1e90/s$ after creating Tobacco division. It's ~ 1 hour and 1.667 minutes.

19.4.2 Advice

- For Wilson and Advert:
 - At the start of each cycle, buy Wilson if you can afford it with last cycle's profit.
 - In this strategy, when profit reaches $\sim 1e18-1e19/s$, Advert's benefits outweigh the cost. Let's call it `thresholdOfFocusingOnAdvert`.
 - ❖ Before profit reaches this threshold, budget for Wilson and Advert is part of budget for product division and corporation's upgrades. Run the optimizer to calculate the optimal numbers of Wilson and Advert upgrades that you should buy.
 - ❖ After profit reaches this threshold, you must focus on Advert. At the start of each cycle, after buying Wilson (if needed), use at least 20% of current funds to buy Advert. Personally, I use up to 60% of current funds.

- After buying Wilson and Advert (if needed), invest most funds in product division and corporation's upgrades ($\geq 90\%$). Support divisions do not need much investment, as long as they can provide enough high-quality materials for product division.
- At the start of round 3, you should spend only a small amount of funds for support divisions. 500b/100b for Agriculture/Chemical is enough. Personally, I spend only 150b/30b.
- For product division and corporation's upgrades:
 - Split the funds into:
 - ❖ rawProduction: SmartFactories, SmartStorage and cities' warehouse upgrades.
 - ❖ wilsonAdvert: WilsonAnalytics and Advert.
 - ❖ office: main office and support offices.
 - ❖ employeeStatUpgrades
 - ❖ salesBot
 - ❖ projectInsight
 - Before thresholdOfFocusingOnAdvert:
 - ❖ rawProduction: 1 / 23
 - ❖ wilsonAdvert: 4 / 23
 - ❖ office: 8 / 23
 - ❖ employeeStatUpgrades: 8 / 23
 - ❖ salesBot: 1 / 23
 - ❖ projectInsight: 1 / 23
 - After thresholdOfFocusingOnAdvert:
 - ❖ rawProduction: 1 / 19
 - ❖ wilsonAdvert: 0
 - ❖ office: 8 / 19
 - ❖ employeeStatUpgrades: 8 / 19
 - ❖ salesBot: 1 / 19
 - ❖ projectInsight: 1 / 19
- For support division, funds ratio of warehouse to offices: 10-90.
- Funds ratio of main office to support offices:
 - Round 3: 75-25.
 - Round 4+: 50-50.
- Office setup for support divisions:
 - Ratio of R&D job to non-R&D jobs is 20-80. In round 3+, EngineerProduction is much more important than RP, so you should set only 20% of employees to R&D job.
 - With non-R&D employees, assign ~65% of them to "Engineer". "Business" is useless, do not assign any employee to that job. Use "raw production" setup to assign the remaining employees. Sample ratio for non-R&D jobs:

- ❖ Operations: 0.22
- ❖ Engineer: 0.632
- ❖ Business: 0
- ❖ Management: 0.148
- Office setup for product division:
 - Main office:
 - ❖ In round 3: use “progress” setup. Sample ratio:
 - Operations: 0.037
 - Engineer: 0.513
 - Business: 0.011
 - Management: 0.44
 - ❖ In round 4: use “progress” setup. Sample ratio:
 - Operations: 0.03
 - Engineer: 0.531
 - Business: 0.003
 - Management: 0.436
 - ❖ After round 4: use “profit-progress” setup.
 - ❖ Sample ratio for: after round 4 and before profit reaches $1e30/s$:
 - Operations: 0.032
 - Engineer: 0.462
 - Business: 0.067
 - Management: 0.439
 - ❖ Sample ratio for: after round 4 and after profit reaches $1e30/s$:
 - Operations: 0.064
 - Engineer: 0.317
 - Business: 0.298
 - Management: 0.321
 - Special case for main office:
 - ❖ Right before accepting offer, switch to “profit” setup to maximize the offer.
 - ❖ Sample ratio for “profit” setup of round 3:
 - Operations: 49 / 138
 - Engineer: 5 / 138
 - Business: 51 / 138
 - Management: 33 / 138
 - ❖ Sample ratio for “profit” setup of round 4:
 - Operations: 68 / 369

- Engineer: 12 / 369
- Business: 244 / 369
- Management: 45 / 369
- Support offices:
 - ❖ In round 3 (before completing first product): set all employees to R&D.
 - ❖ In round 3 (after completing first product) and round 4:
 - Set 1 employee to Operations, Engineer, Business, Management.
 - Set remaining employees to R&D.
 - ❖ After round 4:
 - Set 50% of employees to R&D.
 - With remaining employees, use “profit” setup. Sample ratio: reuse the “profit” setup of round 4.
- Research:
 - RP gain rate in round 3 is too low, so don’t buy any research in that round.
 - Buy “Hi-Tech R&D Laboratory” as soon as possible in round 4. This research is very cost-effective, so it’s fine to use all RP to buy this research.
- Number of products before accepting investment offer:
 - Let’s define a term: X-P/Y-P. This means that you need to develop X products before accepting round 3’s offer, then develop more (Y-X) products before accepting round 4’s offer. For example:
 - ❖ 1P/2P: develop 1 product, accept round 3’s offer, then develop 1 more product before accepting round 4’s offer.
 - ❖ 1P/3P: develop 1 product, accept round 3’s offer, then develop 2 more products before accepting round 4’s offer.
 - I recommend 1P/2P setup. This is the optimal setup if:
 - ❖ Start round 3 with large funds (11t+).
 - ❖ Have a custom Market-TA2 script.
 - ❖ Split funds properly between different divisions/upgrades.
 - ❖ Setup office properly.
 - If you follow my strategy, 1P/2P is the optimal setup, otherwise you need to use different setups (1P/3P, 2P/4P, etc.).
- Efficient strategy should at least reach these milestones:
 - Offer for round 3: 1e16+
 - Offer for round 4: 1e20+
 - Profit of 1e20/s after 4th/5th product
 - Profit of 1e90/s after 7th/8th product
 - Profit of 1e90/s in ~400 cycles
- This is sample log of my strategy. The first number is the cycle number.

0: newest product: Tobacco-00000-1.00000e+9, RP: 0
129: newest product: Tobacco-00001-1.94725e+9, RP: 7.581k
135: profit: 13.804b
145: round: 3, offer: 1.457e16
146: profit: 154.909b
209: newest product: Tobacco-00002-5.96180e+12, RP: 53.614k
211: profit: 37.176t
212: profit: 122.843t
214: profit: 225.114t
225: round: 4, offer: 4.758e20
226: profit: 5.152e16
227: profit: 1.072e17
228: profit: 2.304e17
264: newest product: Tobacco-00003-4.46167e+17, RP: 135.073k
265: profit: 3.366e19
266: profit: 6.052e19
268: profit: 1.33e20
278: profit: 1.083e21
299: profit: 1.022e22
301: newest product: Tobacco-00004-9.85834e+20, RP: 185.437k
302: profit: 1.77e23
307: profit: 1.453e24
314: profit: 1.167e25
323: profit: 1e26
331: newest product: Tobacco-00005-3.82355e+25, RP: 308.598k
332: profit: 1.101e28
333: profit: 3.927e28
334: profit: 1.305e29
336: profit: 1.483e30
338: profit: 1.684e31
340: profit: 2.128e32
342: profit: 3.379e33
343: profit: 1.534e34
345: profit: 4.773e35
349: profit: 1.613e40
353: profit: 1.681e53
354: profit: 8.88e61
355: profit: 4.035e72
355: newest product: Tobacco-00006-1.00000e+72, RP: 868.681k
356: profit: 1.125e75
357: profit: 2.03e75
368: newest product: Tobacco-00007-7.12176e+74, RP: 2.744m
369: profit: 1.592e90

20. Advanced strategies

WIP

21. Strategies for other BitNodes

WIP

22. How to test your strategy efficiently and reliably

- Manual testing through UI is tedious because there is too much clicking and waiting. This section shows you how to:
 - Patch the game to expose useful objects and disable annoying pop-ups.
 - Develop support tools to assist testing process.
 - Remove RNG as much as possible to guarantee consistent test result.
- This is only used for manual testing through UI, if you want to maximize the efficiency, you need to patch the game heavily to run it in headless mode and control everything programmatically. Check the next section for that.

22.1 Patch the game

- Expose Player object. This is crucial for our testing process. It allows modifying corporation's data and exposing multiple useful internal functions.
- Expose saveObject object.

```
Player.lastUpdate = Engine._lastUpdate;
Engine.start(); // Run main game loop and Scripts loop

// @ts-ignore
globalThis.Player = Player;
// @ts-ignore
globalThis.saveObject = saveObject;
```

- Disable offline rewards pop-up: comment out the code right below the above part. They start with these lines:

```
const timeOfflineString = convertTimeMsToTimeElapsedString(time);
setTimeout(
```

- Disable new update / beta update pop-up: in SaveObject.ts, comment out body of 2 functions: createNewUpdateText and createBetaUpdateText.
- Exclude home server's scripts in save file when exporting save.
 - In SaveObject.ts, find getSaveString function, add new parameter (forceExcludeScripts = false), then modify it like this:

```
if (Player.gang) this.AllGangsSave = JSON.stringify(AllGangs);

if (forceExcludeScripts) {
  const AllServersObject = JSON.parse(this.AllServersSave);
  AllServersObject["home"].data.scripts = new JSONMap<ScriptFilePath,
Script>();
  AllServersObject["home"].data.runningScripts = [];
  this.AllServersSave = JSON.stringify(AllServersObject);
}
```



```
const saveString = btoa(unescape(encodeURIComponent(JSON.stringify(this))));
return saveString;
```

- Patch exportGame function:

```
exportGame(): void {
  const saveString = this.getSaveString(false, true);
  const filename = this.getSaveFileName();
  download(filename, saveString);
}
```

You can exclude scripts in all servers if you want. When I test my strategies, there are only scripts on home server, so it works fine for me.

- Expose saveObject in SaveObject.ts. We can use this object to get save string.

22.2 Expose Player object

- You can expose the Player object by using webpackChunkbitburner without patching the game.
- Sample code:

[exploits.ts](#): exposePlayerObject()

22.3 Support tools

- UI tools:

CorpMaintain	UnlimitedBonusTime	RemoveBonusTime	CorpRound	CorpTest	ImportSave	DelScripts	Reload	Exit
Input:		Funds	SmartFactories	SmartStorage	▼	Import	Export	Delete
Division:	Agriculture ▼	RP	Office	WarehouseLevel	BoostMats	ClearBoostMats	ClearInputMats	ClearStorage

- CorpMaintain: run maintenance script. This script takes care of:
 - ❖ Tea/Party
 - ❖ Smart Supply
 - ❖ Market-TA2
- UnlimitedBonusTime: set unlimited bonus time. Bonus time is saved at globalThis.Player.corporation.storedCycles.
- RemoveBonusTime: remove unlimited bonus time.
- CorpRound: run testing script.
- CorpTest: run testing script.
- ImportSave: open file dialog to load save file.
- DelScripts: delete all scripts on home server.
- Reload: reload tools
- Exit: close tools.
- Funds: set corporation's funds.
- SmartFactories: set Smart Factories level.

- SmartStorage: set Smart Storage level.
- Import, Export, Delete: allow quickly save/load game data. Data is saved in IndexedDB.
- RP: set Research Point.
- Office: set employees' jobs
- WarehouseLevel: set warehouse level.
- BoostMats: set boost materials' quantities.
- ClearBoostMats: clear boost materials.
- ClearInputMats: clear input materials.
- ClearStorage: clear all materials.
- Save file editor: [saveFileEditor.ts](#)

22.4 Electron

- All modern browsers have many mechanisms to throttle inactive tabs. This slows down the testing process a lot.
- We can use Electron as a workaround. If you play Bitburner on Steam, you don't need to worry about this problem. It has already been an Election application.
- Sample code:

[electron/main.mjs](#)

22.5 Remove RNG

- Many parts of Corporation use RNG:
 - Issuing new shares.

```
const privateShares = Math.round(getRandomInt(0, maxPrivateShares) / 10e6) * 10e6;
```

- Buying Advert.

```
const popularity = (this.popularity + 1 * advMult) * ((1 + getRandomInt(1, 3) / 200) * advMult);
```

- Decreasing demand and increasing competition of product each cycle.

```
let change = getRandomInt(0, 3) * 0.0004;
```

- Changing total experience and average stats of employee in office when you hire new employee.

```
this.totalExperience += getRandomInt(50, 100);

this.avgMorale = (this.avgMorale * this.numEmployees + getRandomInt(50, 100)) / (this.numEmployees + 1);
this.avgEnergy = (this.avgEnergy * this.numEmployees + getRandomInt(50, 100)) / (this.numEmployees + 1);

this.avgIntelligence = (this.avgIntelligence * this.numEmployees +
```

```
getRandomInt(50, 100)) / (this.numEmployees + 1);
this.avgCharisma = (this.avgCharisma * this.numEmployees + getRandomInt(50,
100)) / (this.numEmployees + 1);
this.avgCreativity = (this.avgCreativity * this.numEmployees +
getRandomInt(50, 100)) / (this.numEmployees + 1);
this.avgEfficiency = (this.avgEfficiency * this.numEmployees +
getRandomInt(50, 100)) / (this.numEmployees + 1);
```

- Setting initial competition of product when it's finished.

```
this.competition = getRandomInt(0, 70);
```

- Updating share price.

```
if (this.sharePrice <= targetPrice) {
  this.sharePrice *= 1 + Math.random() * 0.01;
} else {
  this.sharePrice *= 1 - Math.random() * 0.01;
}
```

- Changing demand, competition, market price of material: check processMarket function in Material.ts.
- Changing average energy and morale of office.

```
this.avgEnergy = (this.avgEnergy - reduction * Math.random()) * perfMult +
(this.teaPending ? 2 : 0);
this.avgMorale = ((this.avgMorale - reduction * Math.random()) * perfMult +
increase) * this.partyMult;
```

- We should try to remove RNG in some parts if we want to ensure consistent test results. With these patches, we reduce the number of outliers.
- Recommended patch locations:
 - Buying Advert.
 - Decreasing demand and increasing competition of product each cycle.
 - Changing total experience and average stats of employee in office when you hire new employee.
 - Setting initial competition of product when it's finished.

22.6 Reliability

- Let's see how removing RNG affects our test result. With old strategy of round 1 (Office size 3):
 - Employee stats getRandomInt(50, 50):
 - ❖ Mean: 428.297b
 - ❖ Median: 428.080b
 - Employee stats getRandomInt(75, 75):
 - ❖ Mean: 496.097b
 - ❖ Median: 496.169b
 - Employee stats getRandomInt(90, 90):

- ❖ Mean: 515.231b
- ❖ Median: 515.012b
- Employee stats `getRandomInt(100, 100)`:
 - ❖ Mean: 482.031b
 - ❖ Median: 482.235b
- One interesting thing to see here: when we run test cases with highest stats, the results do not show highest offers. This is because when employee stats are too high, there are cases that employees produce more materials than what they can sell. The warehouse efficiency is reduced in these cases. However, there is no point in customizing strategy for these cases, 50-90-100 cases are there for observing what happens in extreme cases. All strategies should be devised while using average employee's stats.

22.7 Measure the performance of strategy

- We use the mean and median of the offers to measure the performance of strategy.
- Collect data of the offer:
 - Wait until the offer reaches 50% of “target”. Targets for round 1 and 2:
 - ❖ Round 1: 420b
 - ❖ Round 2: 10t
 - Collect samples of the offer. Each sample is collected in START state. Number of samples:
 - ❖ Round 1: 20
 - ❖ Round 2: 30
 - Get the maximum value of all samples. This is the highest offer that we can get in this test run.
- Repeat the steps above 200 times.
- Calculate the mean and median.

23. Bitburner in headless mode

In order to streamline testing process and maximize efficiency, we must patch Bitburner to allow it running in headless mode. This is an ambitious project, so I dedicate an entire section for it.

In this section, you will find multiple problems and the solutions that I came up with. There is one thing to note here: I write this section as a journal, not as a guide. Occasionally, you will find some problems and solutions that I spend time to investigate, find libraries and tools, write patch, etc., and then I conclude that it does not matter or can be sidestepped entirely.

23.1 JavaScript runtime

- I use Node.js and Deno. Originally, I only use Node.js, but there is a feature that Node.js does not support (importing Blob URL). When I'm trying to find a workaround for that, I find out that Deno supports it, so I add Deno to my supported runtime list.
- Deno has some problems too, but I have found all workarounds for them. Personally, when I run my benchmark project, Deno uses less memory than Node.js, it may be because of the workaround of importing Blob URL. I will show you all of the problems and workarounds in later parts.

23.2 esbuild

- Bitburner's source code is built with Webpack, and Webpack is slow. It's fine when we build it successfully in dev mode, the incremental build speed is acceptable. However, when patching it to run in headless mode, failed build is normal. Waiting 20 seconds every time trying to build is unacceptable. We should use other bundlers like esbuild to speed up build procedure. When I build my benchmark project, these are my numbers:
 - esbuild: 2-3 seconds
 - Webpack: 18-20 seconds
- We don't need to change Bitburner's build procedure, we only need to setup esbuild for our project.
- This is the basic code that we try to build:

```
function init(saveFilePath: string) {
  const saveDataString = readFileSync(saveFilePath).toString();
  if (!saveDataString) {
    console.error("Invalid save data");
    return;
  }
  Engine.load(saveDataString);
  Engine.start();
  newRemoteFileApiConnection();
}
```

23.3 Write esbuild script

- Polyfill `require`, `__filename`, `__dirname`.

Many libraries use `require`, `__filename`, `__dirname` in their implementations and esbuild has problem when bundling them. Check this issue:

<https://github.com/evanw/esbuild/issues/1921>.

We use this workaround:

<https://github.com/evanw/esbuild/issues/1921#issuecomment-1623640043>

- Update `js-sha256`.

Version of `js-sha256` that Bitburner uses is outdated. We should update to latest version to fix this:

<https://github.com/emn178/js-sha256/issues/40>

After updating to latest version, we can see that it changes from `eval("require('crypto')")` to `require('crypto')`, from `eval("require('buffer').Buffer")` to `require('buffer').Buffer`.

This is one of many things that breaks our build without the workaround above.

- Fix problem with global objects in Deno.

Some libraries misidentify runtime environment, e.g., `scheduler`, `file-saver`, `popper.js`. Reasons are:

- Deno has global `window` object.
- Deno does not have `global` object.
- Write `raw-loader` plugin for esbuild.

Bitburner uses `raw-loader`, so we need to write a plugin for esbuild to deal with import declarations that start with `!!raw-loader!`

This project shows how to do it:

<https://github.com/hannoeru/esbuild-plugin-raw>

- Mark `canvas` as external module.

23.4 Sample esbuild script

[customTools.esbuild.mjs](#)

23.5 Non-UI code imports UI code

The first problem is the invalid access to `document`, `window`, `navigator`, etc. Why do we need to access these things when we don't render anything and only access `Engine` object? It is because non-UI code imports UI code. Checking the `import` part of `engine.tsx` shows that we import many

things from `"/ui/*"`. This problem is everywhere, so fixing all of them is impossible. We need to find a way to mitigate it.

23.6 jsdom

- The first solution that comes to my mind is “polyfill all those global objects”. Let’s see how to that.
- Viable solution must at least pass this test:

```
console.log(window !== undefined);
console.log(document !== undefined);
console.log(navigator !== undefined);
console.log(location !== undefined);
console.log(window.getComputedStyle !== undefined);
console.log(window.setTimeout !== undefined);
console.log(window.addEventListener !== undefined);
console.log(document.body.style.background !== undefined);
console.log(navigator.userAgent !== undefined);
console.log(location.reload !== undefined);
console.log(window.location.href !== undefined);
```

- Libraries:
 - jsdom: massive library that emulates many browser’s features. We use this because it’s the only viable solution for both Node.js and Deno.
 - linkedom: very fast, but does not implement as many things as jsdom. It does not pass above test case.
 - deno-dom: only works with Deno. I don’t try it because we need a solution that works for both Node.js and Deno.
- In order to bundle jsdom with esbuild, we need to write a plugin for esbuild. Check this:

<https://github.com/evanw/esbuild/issues/1311#issuecomment-1314553926>

If we naïvely use it like this, it won’t work:

```
import { JSDOM } from "jsdom";
const dom = new JSDOM("<!doctype html><html lang='en'></html>");
// @ts-ignore
globalThis.window = dom.window;
globalThis.document = dom.window.document;
globalThis.navigator = dom.window.navigator;
globalThis.location = dom.window.location;
// Call functions in Engine object
```

The reason: many modules have side effects. When loading those modules, they try to access many global objects to check for environments, features and other things. We need to insert the polyfill into the bundle before loading all other libraries. This involves many steps:

- Build a polyfill stub. We can use the code above for the stub.

- Remove boiler code of esbuild in the stub.
- Insert the stub into the bundle, between the boiler code of bundle and the loader of other libraries.
- After doing all those steps, it crashes weirdly. In `tough-cookie/lib/validators.js`, `isObject` tries to access `toString` and `toString` is undefined. It looks like this:

```
function isObject(data) {
  return toString.call(data) === "[object Object]";
}
```

The original source code of that function and its dependency on GitHub:

```
export function isObject(data: unknown): boolean {
  return objectToString(data) === '[object Object]'
}
export const objectToString = (obj: unknown) =>
  Object.prototype.toString.call(obj)
```

When `tough-cookie` transpiles to JavaScript, it simplifies `Object.prototype.toString` to `toString`, this is usually not a problem because `(toString === Object.prototype.toString)` is true. However, in our situation, `toString` somehow becomes undefined. Manually patching `toString` back to `Object.prototype.toString` solves the problem.

After rerunning it, it works. However, there are still 2 warnings:

This browser doesn't support `requestAnimationFrame`. Make sure that you load a polyfill in older browsers. <https://reactjs.org/link/react-polyfills>

This browser doesn't support `cancelAnimationFrame`. Make sure that you load a polyfill in older browsers. <https://reactjs.org/link/react-polyfills>

- We have to add the option `pretendToBeVisual`.

<https://github.com/jsdom/jsdom#pretending-to-be-a-visual-browser>

Our polyfill becomes:

```
import { JSDOM } from "jsdom";
const dom = new JSDOM("<!doctype html><html lang='en'></html>", {
  pretendToBeVisual: true });
// @ts-ignore
globalThis.window = dom.window;
globalThis.document = dom.window.document;
globalThis.navigator = dom.window.navigator;
globalThis.location = dom.window.location;
```

- Redo all the steps above and rerun. Now it works without any more problems. However, is this really what we should do?
 - First, `jsdom` is a massive library. When we spawn hundred threads to test multiple strategies, we want to use as little memory as possible.

- Second, this involves too many error-prone steps, and many patches are just hacky workarounds without solving the real problem (for example: patching `toString` back to `Object.prototype.toString`). I will not be surprised if we suddenly have more troubles with jsdom.
- At this point, jsdom stops being an attractive solution, using it means we add significant amount of technical debt. Manually patching all the invalid accesses looks like a better approach.

23.7 Manually patch invalid access

- In `GameRoot.tsx`, remove access to `ScriptEditorRoot`. `ScriptEditorRoot` imports `monaco-editor`, and that library makes too much invalid access.
- Patch:

```
export let Router: IRouter = {
  isInitialized: false,
  page: () => {
    if (!process.env.HEADLESS_MODE) {
      throw new Error("Router called before initialization");
    }
    return Page.Options;
  },
};
```

- Patch invalid access in:
 - `Achievements/Achievements.ts`
 - `Exploits/loops.ts`
 - `Exploits/Unclickable.tsx`
 - `PersonObjects/Player/PlayerObject.ts`
 - `Themes/ui/Theme.tsx`
 - `engine.tsx`
 - `NetscriptWorker.ts`
 - `SaveObject.ts`
 - `UncaughtPromiseHandler.ts`
 - `utils/Protections.ts`
- How to patch
 - Use `globalThis` and optional chaining operator (`?.`) to check if global objects are defined.
 - `window` → `globalThis`

23.8 Import Blob URL

- This is the first problem that I have after manually patching invalid access to global objects. I switch to Deno to sidestep it before coming back later to find a workaround for Node.js.
- In `NetscriptJSEvaluator.ts`

```
const url = URL.createObjectURL(makeScriptBlob(adjustedCode)) as ScriptURL;
```

Node.js does not support importing Blob URL:

<https://github.com/nodejs/node/issues/47573>

- There is a workaround in `RunScript.test.ts`. The basic idea is that instead of generating Blob URL, we generate Data URL. Workaround in `RunScript.test.ts` works fine with simple script, but with scripts that contain many special characters, we need to encode them with base64.
- Full patch:

```
let url;
if (process.env.RUNTIME_NODE) {
  url = "data:text/javascript;base64," +
  Buffer.from(adjustedCode).toString("base64") as ScriptURL;
} else {
  url = URL.createObjectURL(makeScriptBlob(adjustedCode)) as ScriptURL;
}
```

- Blob URL is revoked immediately by calling `URL.revokeObjectURL` after being imported. We can patch it like this: `URL.revokeObjectURL = () => { }`. We should patch it in our project, not in Bitburner code.

23.9 WebSocket

- Initializing connection like this does not work in Node.js, we need to use library like `ws` to establish web socket connection.

```
this.connection = new WebSocket(address);
```

- We should check for connection state before sending data and wrap the `websocket.send` in a try-catch. This prevents exception occurs when we send data on a closed connection.
- Full patch:

[Remote.ts](#)

23.10 Worker thread

- We use `node:worker_threads`. I don't want to support both `node:worker_threads` and Deno's implementation of Web Worker API.
- Deno's polyfill has problem with dependency of `jszip` in `download.ts`. Check this:

<https://github.com/denoland/deno/discussions/15205>

- `setImmediate` call `global.postMessage("", "*")`, It makes Deno crashes at runtime. It's a dependency of `jszip` in `download.ts`, so we need to patch `download.ts` to fix it.
- Full patch:

[download.ts](#): `exportScripts()`

- Deno sets `workerData`, `threadId`, `environmentData` asynchronously, so we should be careful if we want to use them.

https://github.com/denoland/deno/blob/eab755501394ef3cce120955223de5c6447bd87d/ext/node/polyfills/worker_threads.ts#L233

23.11 Load scripts directly

- Bitburner uses WebSocket to transfer scripts between our real machines and Bitburner's "servers" through its Remote API. After the game is loaded, it tries to establish a connection with the Web Socket Server (WSS) that we set up. WSS can push file to Bitburner's "servers" through this connection. Currently, Remote API only allows pushing a single file per request. If we have 30 scripts, it has to make 30 requests to push all those scripts.

Data	Length	Time
↓ {"jsonrpc":"2.0","method":"pushFile","params":{"server":"home","filename":,"content":		
↑ {"jsonrpc":"2.0","result":"OK","id"		
↓ {"jsonrpc":"2.0","method":"pushFile","params":{"server":"home","filename":,"content":		
↑ {"jsonrpc":"2.0","result":"OK","id"		

- When I try to spawn a large number of threads to test a strategy, the WSS's memory usage increases too much. I tried these mitigation measures:
 - Close connection as soon as possible. Each thread is an instance of Bitburner, and we terminate it after running the script. There is no point in maintaining a connection after getting all the scripts, so I close it immediately.
 - Detect and close broken connections

<https://github.com/websockets/ws#how-to-detect-and-close-broken-connections>

- These measures can mitigate the problem to a limited extent. With a medium number of threads (about 20-30 threads), they are fine. However, with a large number of threads (> 50), they are still ineffective.
- We run Bitburner in headless mode and directly use things like Engine object. This means we can load scripts from our machine and add them to Bitburner's "servers". Sample code:

```
interface Script {
  fileName: string;
  content: string;
}

const home = Player.getHomeComputer();
const scripts: Script[] = loadScripts();
scripts.forEach(script => {
  const filePath = resolveFilePath(script.fileName!);
  if (hasScriptExtension(filePath)) {
    home.writeToFile(filePath, script.content);
  }
});
```

- resolveFilePath is in FilePath.ts.
- hasScriptExtension is in ScriptFilePath.ts.

23.12 Patch Ceres.js

Search for “// Custom patch” in Ceres.js.

[Ceres.js](#)