Adjust income per room energy mode:

LOW (25% under): -10%
NORMAL (-25% - 10%): 0%
HIGH (+10%): +20%
SUPER [illegible] (+20%): +20%

[illegible] below storage target): -20%

Calculating energy availability:

RoomManager(room) checks owned energy surplus and stores storage which checks only the storage or central container. It then queries every manager and remote room (which queries their managers) for the following creep info:

energy income: mainly miners, report per tick, should be the count of work parts or the mine max energy / regen time

energy active cost: upgraders, builders, repairers, generally report work parts. The controller manager can reduce the sum to 10

energy build cost: all, should be stored in memory: a creep's memory['build']['cost']; double for creeps with claim parts, they have [illegible]

[illegible] 2m for 2x [illegible] cm

[illegible] 2m for 1x 2wcm

[illegible] 1x 2c4m for 1x 2wcm

[illegible] 1x 2c4m, 1x c2m for 1x 2wcm

37-48: 2x 2c4m for 1x 2wcm

49-61: 2x 2c4m, 1x c2m for 1x 2wcm

62-73: 3x 2c4m for 1x 2wcm

74-86: 3x 2c4m, 1x c2m for 1x 2wcm

87-98: 4x 2c4m for 1x 2wcm

99-111: 4x 2c4m, 1x c2m for 1x 2wcm

112-123: 5x 2c4m [illegible] 1x 2wcm

Mining Manager [illegible]
Level 1

miner 1: target: mine 1; build: passive; pri: 100
miner 2: target: mine 1; build: passive; pri: 80
miner 3: target: mine 2; build: passive; pri: 70
miner 4: target: mine 2; build: passive; pri: 70
hauler 1: per hauler need; pri: 90
Upgrader 1: after surplus detected; pri: 60
hauler +: per hauler need; pri: 40

On hauler delivery to upgrader, room checks with controller manager. If it can complete the level with current energy, it clears ~~build queue~~, notifies managers of impending level increase. When mining manager is notified of level 1->2, it [illegible] all miners/builders to active builders. Once their container is built, they become disabled builders and only mine.

Java
computer lang
but incoherent
defining words that
wouldn't need
definition for a
skilled program[mer]

Alternatively no energy levels, just constantly adjust

If storage is below [illegible] [illegible] spawn [illegible] 100 - storage % if over 200 % adjust income by 2 x storage % - 100

Each manager can sum these to return just 3 values. The com should calculate creep build cost / CREEP LIFETIME and add that to creep active cost to get total creep cost. Next the com needs to contact each mining manager (and [illegible] → mining manager) to get the following: distance from spawn and total work parts on miners, and total hauler capacity. Calculate distance * 2 times workparts to get hauler need. Add all hauler need to get hauler need total. Add all hauler capacity to get hauler capacity total. If hauler capacity total < hauler need total, adjust energy income total * hauler capacity total / hauler need total. Energy income total - energy cost total is the energy surplus per tick.

Also energy surplus [illegible] is [illegible]

(w[illegible] mmmm)
WC → CK 7300

D

5: C12, T20, M268, C18, T5, P1, T5, D1, T5, P1
T5, D1, T5, P1, P1, T5, D1

10: C12, T40, M248, C18, T10, P1, T10, D1, T10, [illegible]
T10, D1, T10, P1, P1, T10, D1

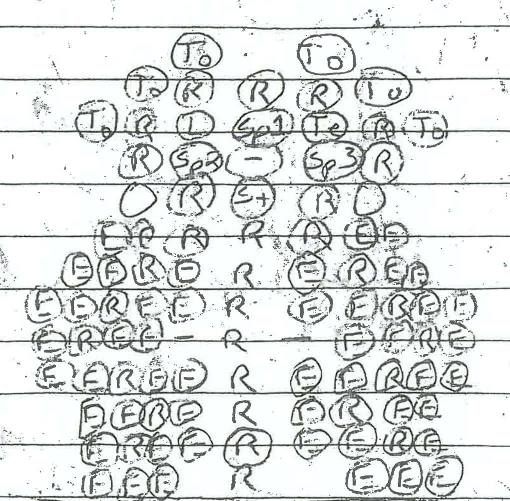
15: C12, T60, M228, C18, T15, P1, T15, D1, T15, P1,
T15, D1, T15, P1, T15 [illegible]

20: C12, T80, M208, C18, T20, P1, T20, D1, T20, P1,
T20, D1

30: C12, T120, M168, C18, T30, P1, T30, D1, T30, P1
T30, D1

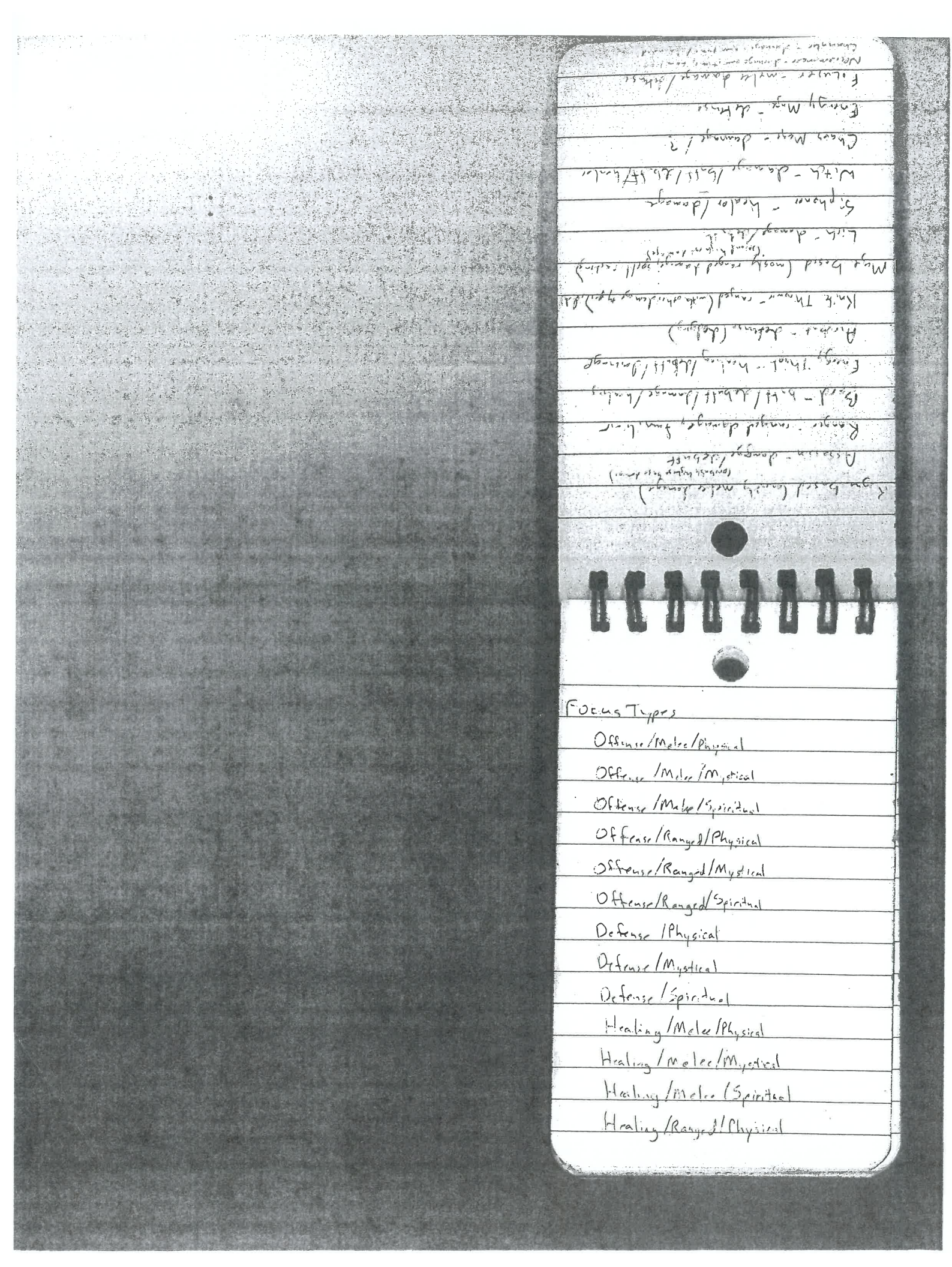
40: C12, T160, M128, C18, T40, P1, T40, D1, T40,
P1, T40, D1

50: C12, T200, M88, C18, T50, P1, T50, D1, T50,
P1, T50



[illegible] C9 T5 [illegible] D1 [illegible] P1

M5 M1 M5 M1 M5 M1 M5 M1
T5 D1 T5 P1 T5 D1 T5 P1



Focus Types

Offense/Melee/Physical

Offense/Melee/Mystical

Offense/Melee/Spiritual

Offense/Ranged/Physical

Offense/Ranged/Mystical

Offense/Ranged/Spiritual

Defense/Physical

Defense/Mystical

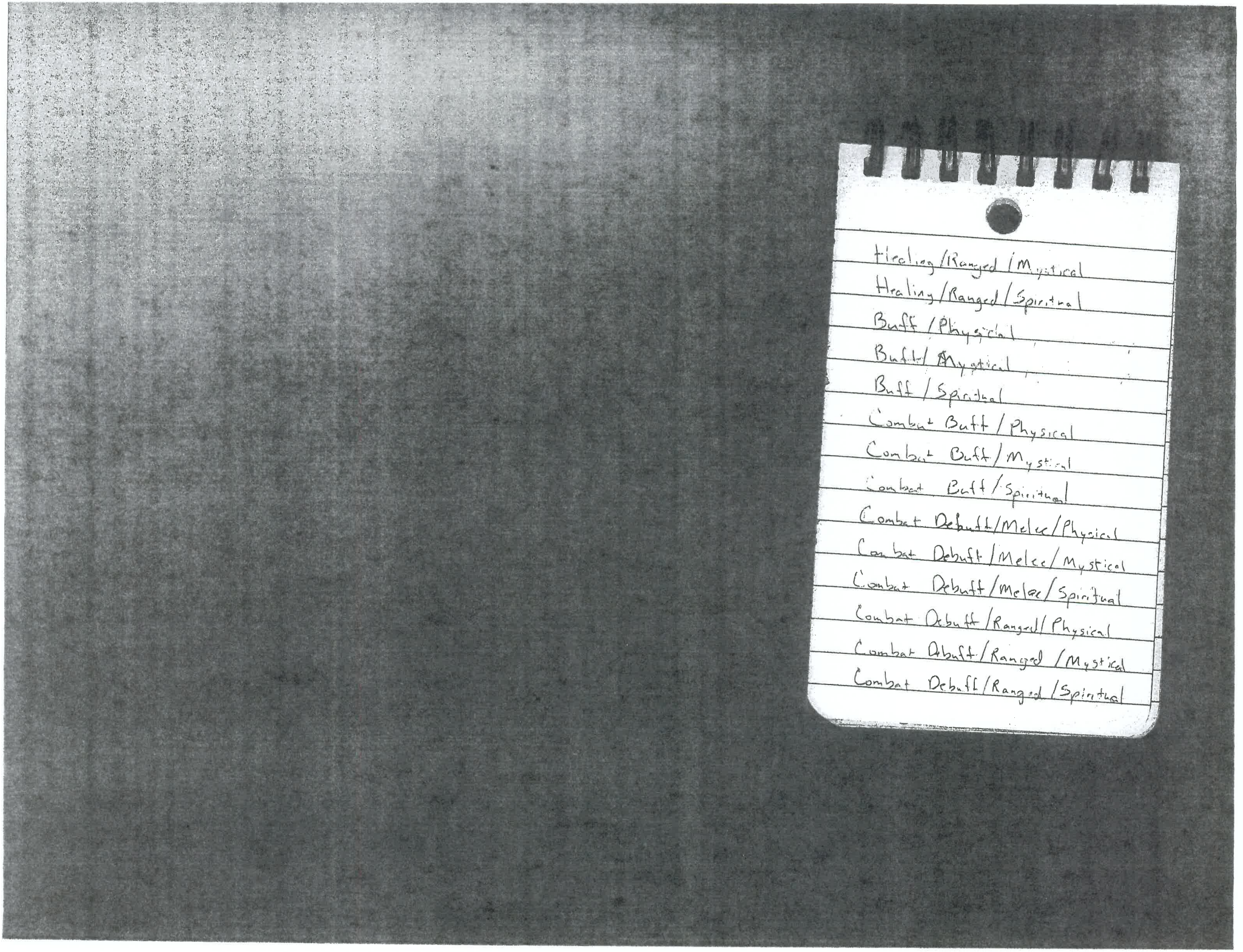
Defense/Spiritual

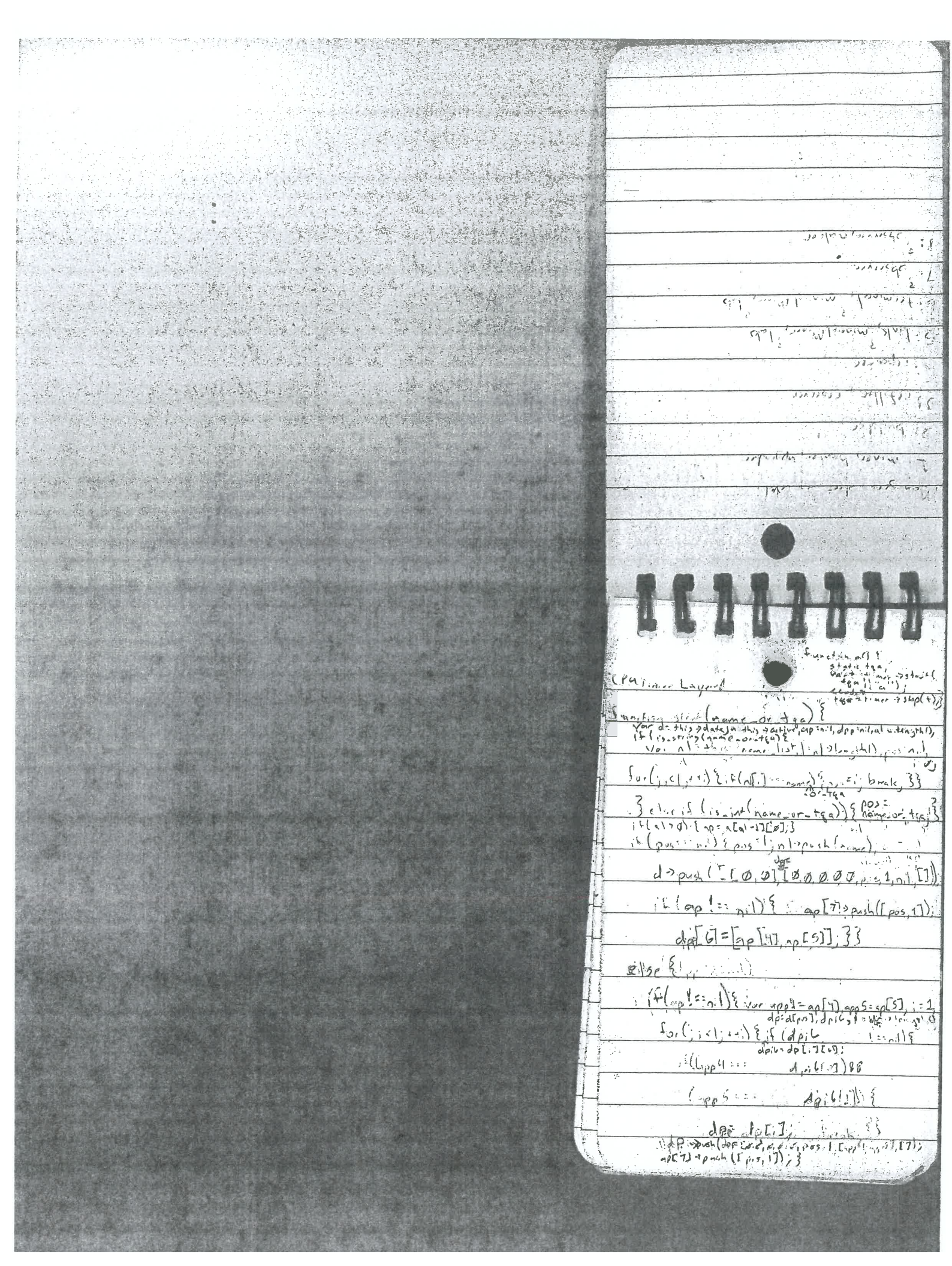
Healing/Melee/Physical

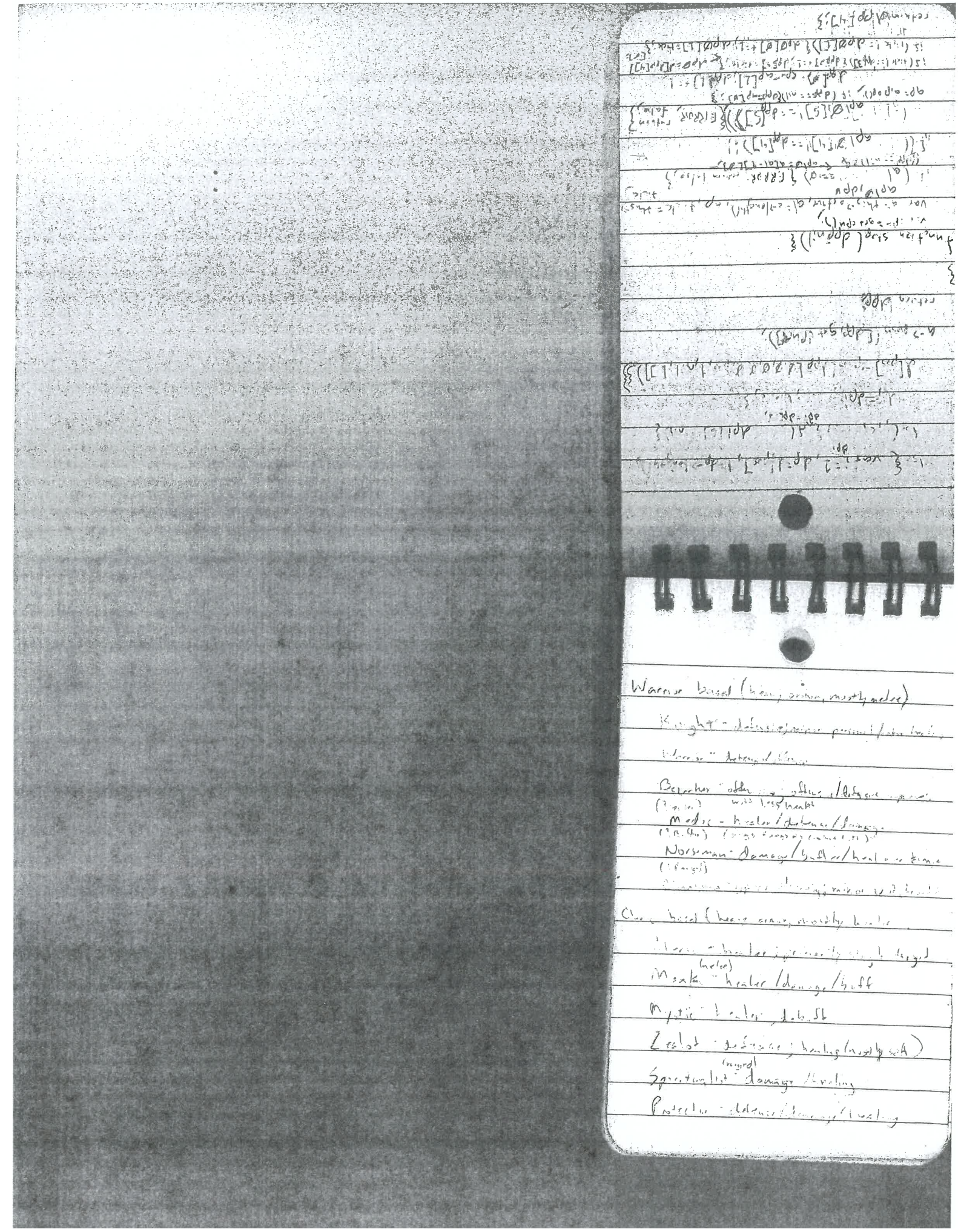
Healing/Melee/Mystical

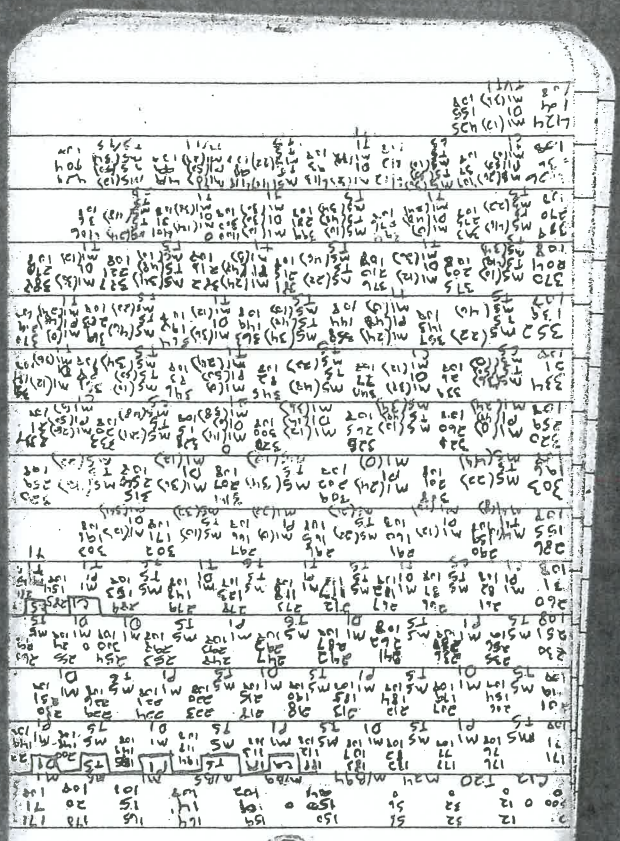
Healing/Melee/Spiritual

Healing/Ranged/Physical


Healing/Ranged/Mystical
Healing/Ranged/Spiritual
Buff/Physical
Buff/Mystical
Buff/Spiritual
Combat Buff/Physical
Combat Buff/Mystical
Combat Buff/Spiritual
Combat Debuff/Melee/Physical
Combat Debuff/Melee/Mystical
Combat Debuff/Melee/Spiritual
Combat Debuff/Ranged/Physical
Combat Debuff/Ranged/Mystical
Combat Debuff/Ranged/Spiritual



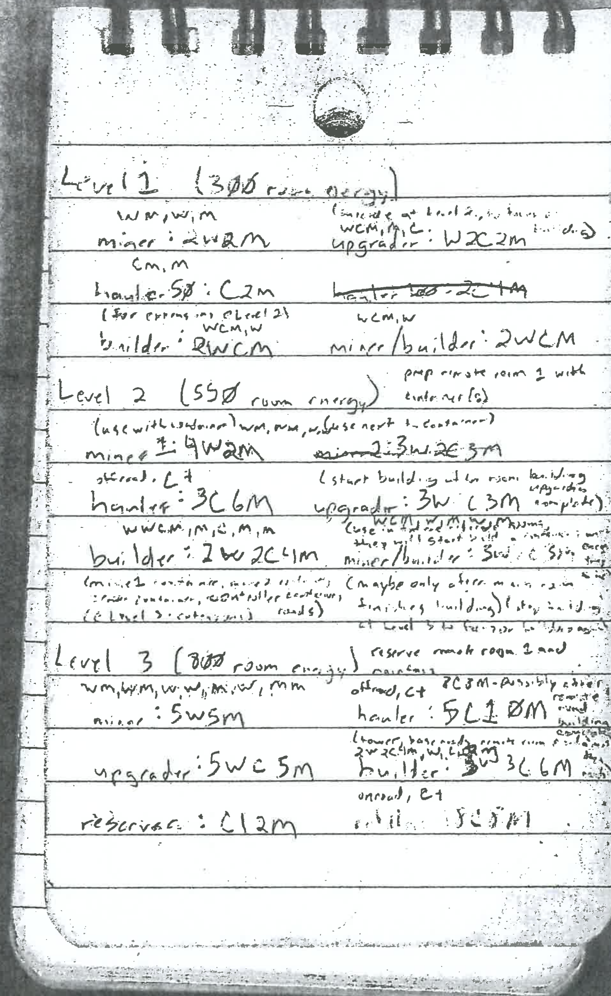




special logic to get around a negative surplus value. Once container is built, change status in miner/builder so it mines only (the one not on top of the container may need a different status so it knows when to transfer to container). Once new statuses are active, change energy action cost to 0. This should cause a hauler to be created, then an upgrader, then more haulers or pure miners as needed. If container building energy cost is 1000, only 4 miner/builders should be required.

(priority 95) first (priority 92) first (priority 48) (priority 50)

delivery focus: upgrade, refill



Level 1 (300 room energy)

WM,W,M
miner: 2W2M

(upgrader at level 2, ... building)
WCM,M,C
upgrader: W2C2M

CM,M
hauler 50: C2M

~~hauler 100: 2C 4M~~

(for expansions at level 2)
WCM,W
builder: 2WCM

WCM,W
miner/builder: 2WCM

Level 2 (550 room energy)

prep remote room 1 with container(s)

(use with container) WM,MM,W,(use next to container)
miner 1: 4W2M

~~miner 2: 3W 2C 3M~~

offroad, C+
hauler: 3C6M

(start building at ... building upgrades complete)
upgrader: 3W C3M

WWCM,M,C,M,M
builder: 2W2C4M

WCM,W,M,W,M
miner/builder: 3W C 3M

(maybe only after main room finishes building)

Level 3 (800 room energy)

reserve remote room 1 and containers

WM,WM,W,W,M,W,MM
miner: 5W5M

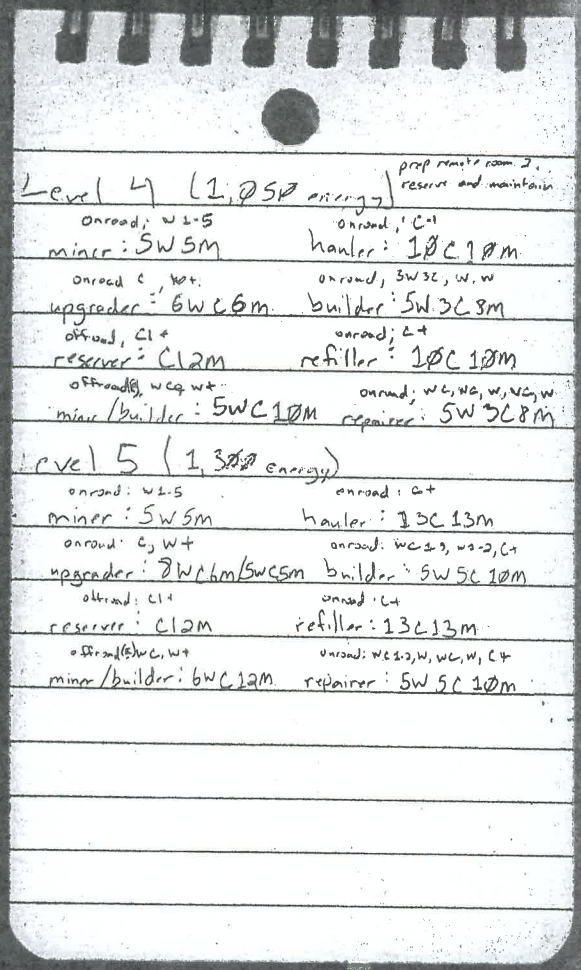
offroad, C+
2C8M - possibly after remote road building complete
hauler: 5C10M

upgrader: 5WC5M

2W2C4M,W,C,2M
builder: 3C6M

onroad, C+

reserver: C12M



Level 4 (1,050 energy) prep remote room 2, reserve and maintain

onroad: W1-5
miner: 5W5m

onroad, C-1
hauler: 10C10m

onroad C, W+
upgrader: 6WC6m

onroad, 3W3C, W, W
builder: 5W3C8m

offroad, C1+
reserver: C12m

onroad, C+
refiller: 10C10m

offroad(5), WCq W+
miner/builder: 5WC10m

onroad: WC, WC, W, WC, W
repairer: 5W3C8m

Level 5 (1,300 energy)

onroad: W1-5
miner: 5W5m

onroad: C+
hauler: 13C13m

onroad: C, W+
upgrader: 9WC6m/5WC5m

onroad: WC1-3, W3-2, C+
builder: 5W5C10m

offroad: C1+
reserver: C12m

onroad: C+
refiller: 13C13m

offroad(5)WC, W+
miner/builder: 6WC12m

onroad: WC1-2, W, WC, W, C+
repairer: 5W5C10m

Basic room manager logic

Room Manager 1

With 0 energy surplus, no upgraders or haulers should be created. The mining manager should request 1 miner/builder. Two for the first mine at priority 99, the second at priority 49. The room planner should find the best positions for the mining containers. Prioritize the 2 build sites for the containers. Each miner/builder will mine for 1 tick, then build for 1 tick, until the container is built. Each miner/builder will report energy income and energy active cost which should 0 out. May need

1- p2p.bak
2- BLL
3- Balanced Nutrion Foods
4- Screaps
5- 3s - Algone Pool (area)
6- 3s - Roach Motel (area / ?guest)
7- 3s - Nanites Guild
8- 3s - P'layer Start Market

1-1

Initial negotiations
- VNVN
- EVN (if necessary)
- CVN
- Encryption expectations (ER, EP, NP, NR)
- Network short id
- Network id
- Network short authorization (if necessary)

Conversation - all communication from start to finish
Topic - part of conversation centered around completing a specific action
Command - single line of communication

Pre-VNVN command outline
- 0x00 - reserved (unused)
- 0x01 to 0x3D - (system)
- 0x3E - (system) close connection
- 0x3F - (system) error
- 0x40 to 0x5F - (user defined)
- 0x60 to 0x7F - (user defined, private only)
- 0x80 to 0xFF - reserved (future expansion)

Command values
- 1 byte per unique command
- or 2 bytes, using command groups

1-2

Validating server's ToFBC

One BC entry for each primary storage

? Seperate BC for secondary storage - need to track it

Primary storage BC entries (client ↔ server(primary storage))

[a] - Part of first level data, part of signing

[b] - Part of second level data, part of hash code calculation

[c] - Extra data ([c-c] implies client only, [c-s] implies server only)

[b] Server data network id

[c] Client data network id

[a] Data id

[a] ? Hash of data

[a] Datetime of start of this contract

[a] Datetime of end of this contract

[a] Datetime of start of first contract for this data

[c-c] Datetime of last validation of server data by client

[a] Length of data

[a] Server public key (? location)

[a] Client public key (? location)

[b] Server signature

[b] Client signature

[c] Hash calculation result

[a] ? Value to append to data for hash calculation

[c] Count of value appended

[a] Data multiplier requested

[c] Other server data ids holding a copy of data

[c] Datetime of last validation of other servers

[a] Server this BC location

[a] Client this BC location

[a] Server previous BC location

[a] Client previous BC location

[a] Server previous BC hash

[a] Client previous BC hash

[c] or [c-s] Server next BC location

[c] or [c-c] Client next BC location

1-3

Primary concepts overview

Backup

SET - send data remotely

GET - retrieve remote data (recovery)

VERIFY - verification in part or whole of some data

p2p - trust no one; assume single servers will constantly go offline

multiple copies of data

contract for storage between client and primary storage

contract between servers for secondary backup storage (amount of data)

contract for secondary storage between servers (single data)

encryption of client data before sending

other

communication encryption (short conversations possibly not encrypted by default)

version management (encryption, commands)

p2p

tertiary storage - not counted as one of the primary or secondary copies; ?keep record; (could be useful to quickly upgrade from tertiary to secondary servers); useful for trust building for new

trust - online/available, data validation, blockchain hash creation

Backup

storage of encryption key(s), public/private key pairs

storage of file organization (subdirectories), file names, ?datetime stamps, attribute(s), ?metadata

INIT

INIT SETTINGS

INIT QUICK

ENCRYPTION VERSION NEGOTIATION

ENCRYPTION NEGOTIATION

ENCRYPTION SETUP

ENCRYPTION TEST

ENCRYPTION ENABLE

ENCRYPTION DISABLE

COMMAND VERSION NEGOTIATION

NEGOTIATE PRIMARY BACKUP CONTRACT

SET

GET

UPDATE

DELETE

VERIFY PRIMARY BACKUP

RENEGOTIATE PRIMARY BACKUP CONTRACT

NEGOTIATE SECONDARY BACKUP STORAGE CONTRACT

RENEGOTIATE SECONDARY BACKUP STORAGE CONTRACT

NEGOTIATE SECONDARY BACKUP CONTRACT

RENEGOTIATE SECONDARY BACKUP CONTRACT

VERIFY SECONDARY BACKUP

NEGOTIATE TERTIARY BACKUP CONTRACT

UPGRADE TERTIARY BACKUP CONTRACT

RENEGOTIATE TERTIARY BACKUP CONTRACT

UPGRADE SECONDARY BACKUP CONTRACT

VERIFY TERTIARY BACKUP

2-1

Basic values

integer - int[8,16,32,64][S,U], float - float[32,64], boolean,

string, block (block, function, object), array/hash/btree/etc

Interfaces

I/O - Network (TCP/IP, UDP/IP, HTTP/UDP/IP, Socket), stdin, stdout, stderr

String matching - regular expression, glob

Big numbers (integers with more bits needed than compiled version can do natively)

CGI interface

DB

ADO

threading - thread, threads

polling

3-1

Total Fat	<65g	Carbohydrates:	300g
Sat Fat	<20g	Dietary Fiber:	25g
Cholesterol	<300mg	Added Sugar:	50g
Sodium	<2400mg	Protein:	50g

*Carb % does not include dietary fiber

Sizes	Small	Medium	Large
	1" x 1" x 1/8 - 1/4"	2" x 2" x 3/4" - 3/5" 1.5" x 1.5" x 1/4" - 3/8"	3" x 3" x 3/4" - 1/2" 2" x 2" x 3/8" - 2/5"
All square (wafers)	Very dry (low water) ~22.7g (0.8 oz)	slightly moist ~34g (1.2 oz)	moist ~45.4g (1.6 oz)

Basic Nutritional Info (per wafer): Calories: [100] (420 kcal)

Total Fat: 3.2g (28.2 kcal) (5%) Carbohydrates: [16]g (64 kcal) (5%)

Sat Fat: [0] - 0.25g Dietary Fiber: [1]g

Cholesterol: [0]mg Added Sugar: [0]g (10 kcal)

Sodium: 0 - 50 mg Protein: [2.5] g (5%)

No allergens, minimum ingredients, natural ingredients

Flavors: Plain Sweet Chocolate

Nutrional Changes: - Added Sugar: 0-2.5g ←

Notes: try w/o milk

Future Flavors: Strawberry, Blueberry, Apple, Cherry, Raspberry, Peanut Butter

Future Types: High Protein Super High Protein

Nutrional Changes: Total Fat: [3.0]g (27 kcal) (4%) Total Fat: [2.2]g (19.8 kcal) (3%)

Carbohydrates: [15]g (52 kcal) (4%) Carbohydrates: [14]g (48 kcal) (3%)

? Dietary Fiber: 2g ? Dietary Fiber: 4g

Protein: [5] g (20 kcal) (10%) Protein: [10]g (40 kcal) (20%)

3-2

Future Types: Extreme Protein

Nutritional Changes: Total Fat: [1.5] g (13.5 kcal) (~2.5%)

Carbohydrates: [13] g (28 kcal) (~2.5%)

Dietary Fiber: 6g

Protein: [15] g (60 kcal) (30%)

Energy

Total Fat: [1] g (9 kcal) (1.5%)

Carbohydrates: [22] g (86 kcal) (7%)

Dietary Fiber: 0.5g

Added Sugar: 2.5-[5] g (10%)

Protein: [1] g (4 kcal) (20%)

+Caffiene: 20mg

Low Fat

Total Fat: [1] g (9 kcal) (~1.1%)

Carbohydrates: [21] g (84 kcal) (7%)

Protein: [3] g (12 kcal) (6%)

No Fat

Total Fat: [0] g

Carbohydrates: [23] g (88 kcal) (7%)

Protein: [3] g (12 kcal) (6%)

No Carbs

Total Fat: [3] g (27 kcal) (5%)

Carbohydrates: 7 g

Dietary Fiber: 7 g

Protein: [18] g (72 kcal) (36%)

High Fiber

Dietary Fiber: [2] g

Super High Fiber

Dietary Fiber: [4] g

4-1

- Hauler Manager

Classes:

RoomPlanner	RoomManager	RoadPlanner
RoadManager	CreepManager	?CreepPlanner
TripPlanner	Creep*	EmpireManager
Combat	RoomLayoutManager	RoomReservedManager
RoomRunnerManager	?RoomRoadManager [1-8]	?SpawnManager
LabManager	?TerminalManager	WallPlanner
WallManager	Diplomacy	Stats
MemoryManager	EmpirePlanner	CPUTimer
Room*	?CreepManager [1-8]	ControllerManager
BuildManager	?BuildManager [1-8]	RepairManager
?RoomLayoutStaticManager	?RoomLayoutDynamicManager	Cache

Empire Priorities

*Room Levels	Spawning
Controller Upgrading	Energy
Controller Upgrading	Creeps
Creeps	Refilling
Work Boost	Room Levels
Energy	Mining
*Creeps	Creeps
Spawning	Hauling
Energy	Creeps
Work Boost	Refilling
Labs	Creeps
Minerals	Energy
*Energy	*Construction
Mining	Creeps
Hauling	Energy
	*Maintenance
	Creeps
	Energy

Startup: Visibility is required to gather room data for the RoomPlanner class. For the first room, a spawn structure must be placed for visibility (future rooms may use a creep or observer structure). Once the room planner has found what it believes is the best position, it should notify (1) if you are already in that position and automatically continue, (2) the coordinates of the best position and if the current position would fit the preferred layout (and why it is less efficient), then allow you to select to (2a) destroy the current spawn to allow you to reposition it or (2b) keep the existing spawn and continue. If 2a happens, and the spawn is relocated to the best position, continue. If 2a happens and the spawn is placed elsewhere, perform another scan and follow (2) above.

On picking the first room, the following goals should try to be met:

1) Mineral should be useful for work boost (O, H,)

2) Two mines available (one if they made the change for single mines to have 3,000 energy), preferably with 2+ walkable tiles next to it (7,500 energy / 300 ticks)

3a) At least 5 available mines that are 1 room distance or (and none of them are "center" rooms) (ditto on the 3,000 change in 2)

3b) At least 2 available mines in 1 room that is 1 room distance and an exit that leads to a center room (bonus if center room falls on 1) (ditto on the 3,000 change)

4) Probably enough space to fit default layout

(cont)

5) Smaller exits are a bonus

6) Try to find a second room, 2 or 3 travel distance away, that fulfills rules 1-5 (the empire planner should picks it automatically if it thinks it is the best, or you should be able to force the empire planner to pick it)

Room planning (finding the best layout and position for all structures that will exist in the room, up to maximum level) involves the following classes: RoomPlanner (rp), RoadPlanner (rop), RoomLayoutManager (rlm), and WallPlanner (wp). The first room is the only one that should have a spawn in place before planning starts. For efficiency, this room will be placed into a PLANNING status. This should disallow standard creep building, etc. [The rp should notify the rop it needs visibility of rooms at all exits that have 1+ mine. The rop should request enough scouts be spawned and sent to the target room as soon as possible. The rop should notify the rop that it needs road plans through every exit that has 1+ mine and the road plan should include creating roads to within 1 distance of all mine(s) in the room. If there are 2+ mines, the road in the 1 distance room should try to merge into 1 road if efficiency is not lost. This is primarily pre-emptive, as the true road planning test cannot begin until a potential placement for storage is found. The early notification is primarily so the rop can have the RoomManager spawn 1 scout creep (1m) per exit and have them in the other rooms to give the rop visibility of these rooms when those calculations are needed.] The wp is contacted to create a layout for simple walls on every exit. It should find every exit position grouping and temporarily save the positions.

(cont)

It will use these groupings to create a solid wall that is 2 distance from each exit position and create a corner(s)and side(s) as needed. Once all wall group positions have been found, locate all positions between the wall and its exit. These are combat zones, where enemies can stand and attack. For each combat zone find all positions within 3 distance that are not walls. These are all part of the danger zone. Enemy creeps in the combat zone with ranged parts can damage creeps in the danger zone.

? (The wp will need to understand combat/danger zones)

The wp will feed info about the wall positions to the rp and the rp should calculate the combat and danger zones. The rp will consider all walls, exits, constructed wall positions, combat zone positions and danger zone positions as unavailable for a room planning position. The rp will then start to create a heat map of the room in order to find the position(s) in the room that are the furthest from walls in all directions. It will need to calculate values for every position in the room, then compare those against the eventual final array, and possibly update some or all of the final array. The heatmap involves scanning in all 8 directions, one at a time, and finding the distance (backwards) to the last unavailable space. Using "down" as an example, start on the top row and mark them all as Ø. On the second row, if the position is not available, mark it as Ø and continue to the next one. If it is available, look 1 position "backwards" (up in this case), get the current value (from the "down" data only), add 1 to that value and save the result as the value for the current position. Once all positions have been calculated, this new data will have to be merged into the final array. If it is the first one,

(cont)

then it can simply be dumped straight into the array. For runs 2-8, each position will have to be tested. For each position, if the value for the position in the current run is less than the value currently in the final array for that same position, update the final array position value with the current run position value. (Note: This can probably be done during the value building phase) Once all 8 directions have run and the final array is complete, scan the final array to find the largest value. Use a new array to store positions into, ordered by value descending. Find largest value down to 1. Scan the final array finding any values that match the largest. Save the position into the new array. Decrease the test value by 1 and repeat the scan and save. If the test value is 0 after the decrease, stop scanning. The ordered array should now have all potential locations to test in the room for the new layout.

The rp should request a room layout from the rlm. It should get a list of positions relative to the "center" of the layout. At the time, storage is generally considered to be the "center" of all layouts. Start a new array to store viable positions for this layout. Scan every position in the ordered array against all relative positions from this layout. If any of them match an unavailable space, stop scanning early and move to the next test position. The heat map is probably the fastest way to check for unavailable positions. If all positions are available, add this "center" test position to the viable array and continue testing to find all viable positions. Once this run has concluded, the rp should contact the rlm

(cont'd)

to request the next layout. It should send the current layout and the number of viable positions it found for that layout. The rm should respond with a new layout structure, same relative position arrays, or nil if it believes the additional processing is not worth the time/value. Generally, the rm should return at last 4 layouts, the primary layout rotated 4 ways: 0°, 90°, 180°, and 270°. If viable matches are found for any of these, it will stop sending layouts. If additional layouts are required, prefix to change them in this order: test all tower layout pieces, test all extension layout pieces, test all spawn layout pieces.

Now that we should have all viable positions, possibly for multiple layouts, we need to find the one that has the best location based on mine positions in the current room and all future remote rooms. Test future remote rooms for visibility. If any are not visible, create a new scout if needed, or wait for visibility in all future remote rooms. Once all future remote rooms have visibility, continue. For each layout, the rp should request a relative position array that only includes positions that should not be walkable by the "no special access creeps". This will include all basic structures that are not able to be walked on, as well as any positions that have roads marked as limited access. An example of a limited access road are the 2 roads created inside the extension layout piece, which should only be available to a "refiller" creep. Once this relative position array is returned, every viable position for this layout needs to run a road test. Create a temporary absolute position array of unwalkable positions. Generate a path from storage to 1) each mine in the current room; 2) every mine in the future remote rooms. For each result,

(cont)

get the distance (if no path found, assume it is unavailable and set distance to 1000). Adjust the distance as follows: distance = distance * (miner -> max_energy_available / MINE_DEFAULT_MAX_ENERGY_AVAILABLE) This adjustment should give a boost to any future remote rooms that are "closer" rooms with miners leaving 3000 max energy available. Add all the distances together for a total road distance. Save the total road distance with the current layout + viable position info. If this total road distance is less than the current min total road distance (initialize as INT_MAX or 1,000,000), then update the min total road distance with this total road distance and save the layout and viable position in new arrays with the min total road distance (minTotal_road_distance = {value: INT_MAX, layout_list: [], position_list: []}) If this total road distance is equal to the current min total road distance, add this layout and viable position to their respective arrays. Test all layouts with each viable position for that layout. When done, if the position array has one value, then that is the best position for storage. Get the relative position for the first spawn structure, calculate the absolute position, and report the findings. If the position array has more than one value, run another road test, like above, but only on the miner(s) in this room (This may give us a slight boost at low room levels). If that returns multiple positions, pick the first. Find the spawn position as described above and report the findings. Also record the layout and position choices in an easy to access memory location.

Reporting should be sent to the console and via email for the first room (subsequent rooms may report, but it will be easier to manage the first spawn in these rooms). Reporting should only happen on the first tp. The player

(cont)

must choose to 1) leave the current spawn where it is (if it is a viable position, consider recording every layout that would work and the road distance value the that value and the best one can be reported to the player. If they are close, they may choose to leave it), 2) destroy their empire (bar spawn and a few scouts), wait for the timeout to expire, and re-place the spawn in the best position. If they choose 1), the room starts normal processing and room status is updated. Option 1) will have to be processed via a button in console output (which causes a console command) or by console command. The console command should be displayed with the message in both the console and email. If option 2) is chosen, when the spawn is placed, its room and position are verified against the stored data. If it matches, delete the room and position data (keep the layout) and start normal processing and update room status. If it does not match, repeat option 1) and 2) from above (the player may have accidentally set it in the wrong position). At this point, the rp should be finished processing for this room and should clean up any memory still in use.

Room Layout Manager:

At this point, all primary layout pieces are initially vertical. The tower piece is on top. It connects to the spawn piece via the first spawn structure. The spawn piece is in the center. It connects to the extension piece via the storage structure. The extension piece is on the bottom. There is also a lab piece, which is 4x4 in size, but its location can be calculated after the primary pieces and other local and exterior roads have been built. Also the observatory (place really anywhere), and

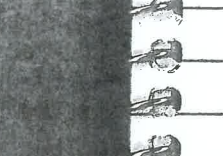
(cont)

the nuker (useful if close to storage). The rlm should combine primary pieces according to internal data showing which primary pieces fit together. Layout naming should look as follows: "layout-1-1-1-Ø" (layout-<spawn piece id>-<extension piece id>-<tower piece id>-<rotation[Ø;9Ø;18Ø;27Ø]>). The rlm must be able to rotate any basic layout (with all primary pieces) from Ø° to 9Ø°, 18Ø°, 27Ø°. The following are some possible piece layouts:

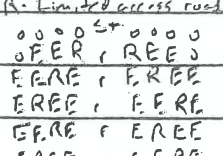
Extension layout piece options:

St - Storage r - road
E - Extension O - open
R - Limited access road

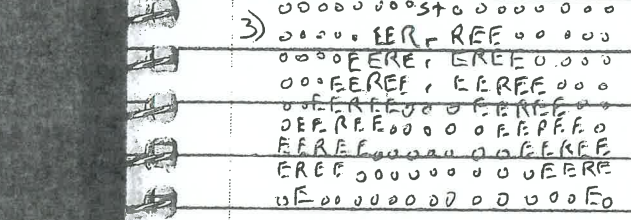
1)



2)



3)

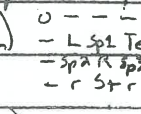


Spawn layout piece options:

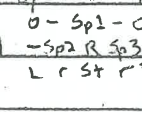
Sp[1-3] - Spawn 1-3 Te - Terminal O - open C - Container
St - Storage L - Link - - unused but must be
R - Limited access road r - road available for walking

1)

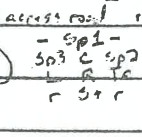
2)



3)

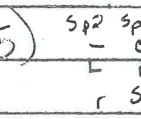


4)



5)

6)



4-10

Tower layout piece options: To-Tower, Sp1-Spawn, r-road, o-open (unused but must be walkable)

1)
o o To o To o o
o To r r r To o
To r - Sp1 - r To

2)
o o To o To o o
To o r r r r o To
o r - Sp1 - r o
To - - - - - To

3)
To o To o To o To
o r r r r r o
To r - Sp1 - r To

4)
To To o To To
o r r r r o
To r Sp1 r To

5)
o To o To o
To r r r To
o r Sp1 r o
To - - - To

6)
To o To
To r To
To r To
- - -
- Sp1 -

7)
o To o To o
To r r r r To
o To r To o
- - - - -
- - Sp1 - -

Lab layout piece options: L-Lab, R-Limited cross road, o-open

1)
o L L R
L L R L
L R L L
R L L o

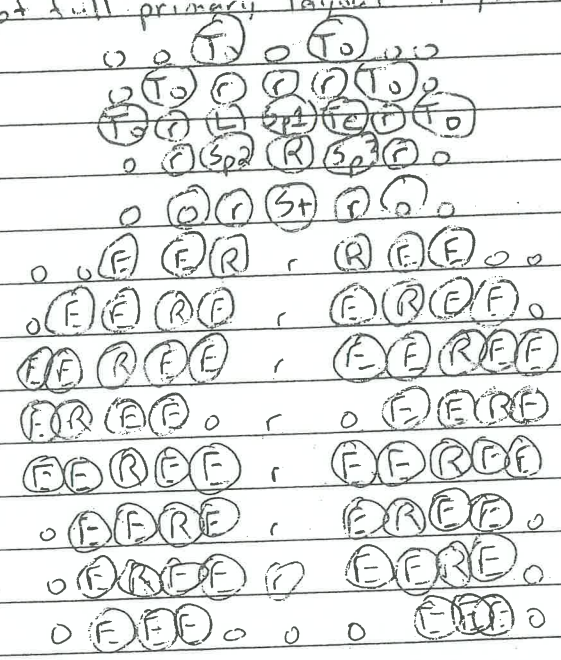
2)
R L L o
L R L L
L L R L
o L L R

Note: The road furthest from the room storage is unraveled

Example of full primary layout: layout-1-1-1-Ø

Circles: shield walls

Shield walls will have to be eroded into the layout for special positions. Mainly roads.



4-11

Room Types:

- There are quite a few permutations available for room types.
1) The room name can be used to identify if a room is in the center of a sector (4 mines @ 3000 energy mineral, 1 hostile creep spawns, no controller), a highway between 2 normal sectors (no mines, no controller) or a standard room (1 mine, mineral, controller). 2) A room can be unknown if there is no visibility and no logged information stored in memory, or known for the rest. 3) The known rooms can be owned by someone, reserved by someone, managed by me, etc. (since center rooms can't be owned/reserved, the user can mark it as reserved or if another player seems to be actively using the room it can be marked as managed), or empty. 4) Last is who manages the room, either me or another player (other players may be expanded further by diplomacy settings, ex: friendly, hostile, neutral/peaceful, or neutral/hostile)

Energy availability calculations:

Every owned room should be able to accurately calculate energy surplus per tick. This should help the Room Manager (rm) to avoid over-building creeps that could cause creep management efficiency to degrade. The rm should query each local manager and remote rooms (which will query their own managers) for the following info (this data should be able to be added together for multiple rooms, so each manager and remote room would return just 3 values):

(cont)

4-12

energy_income: This is probably only useful for miners. Report the minimum of the following: 1) count of work parts, or 2) the miner's mine's maximum possible energy / mine's regeneration time. This is expected to be a value of energy per tick

energy_active_cost: Any creep that requires energy to perform work has an active cost. This should include upgraders, builders, repairers and possibly more. Generally, report work parts as a simple result. Controllers should limit the total work part count of upgraders to a maximum of 10. This is expected to be a value of energy per tick

energy_build_cost: All creeps should have this stored in memory at creep.memory['build']['cost']. Divide this value by maximum possible lifetime for the creep (900 default, 450 if they have a claim part). This should be the cost per tick of the creep over its lifetime. This is expected to be a value of energy per tick

The rm should add all energy_active_cost and energy_build_cost together and store as energy_cost_total. The rm should add all energy_income values together to get energy_income_total. Next, the rm should query the hauler manager to get hauler efficiency. If this value is below 100%, adjust energy_income_total *= hauler_efficiency_percent. Calculate energy_income_total - energy_cost_total to get energy_surplus per tick. Also possibly useful (floats can get messy), calculate energy_surplus per tick * (CREEP_LIFETIME) and round to the nearest integer to get energy_surplus_per_lifetime.

Optional: Once storage is created, if it is below 100% of target value, adjust energy_surplus_per_tick *= (100% - storage% of target) * 0.5. If it is over 100%, adjust energy_surplus_per_tick *= (storage% of target - 100%) * 2

4-13

Calculating hauler efficiency / utilization:

The hauler manager would query the mining manager to get the following information from each mine, individually: distance from storage (should be stored in mine memory already) and the minimum of either the total work parts active on the mine or the mine's maximum possible energy/min's regeneration. For each mine, calculate ((distance * 2) + 2) * usable work parts. Sum all these values together and store as hauler_need_total. Next add all hauler maximum capacity together to get hauler_capacity_total. Hauler_capacity_total / hauler_need_total is hauler_efficiency. Hauler_need_total / hauler_capacity_total is hauler_utilization. Generally each owned room wants to keep hauler_utilization at less than 100%. If it is equal or over, more haulers/size probably needed.

Storage energy target calculation:

STORAGE_ENERGY_TARGET_LEVEL_1 = 50,000

STORAGE_ENERGY_TARGET_LEVEL(l) = 2 (Each level doubles the target until it matches the level 8 target at level 8)

start = [Room level == 1] ? 0 : S.E.T.L.8 / pow(S.E.T.S, 8 - [room level])

range = [Room level == 8] ? 0 : ((S.E.T.L.8 / pow(S.E.T.S, 7 - [room level])) - start)

target = start + ((controller upgrade complete for this room level] / [controller upgrade total needed to complete this room level]) * range)

4.14

CPU usage monitoring: (if object might stay in memory, use this to skip constructor)

```
obj CPUTimer = {
    [illegible]: nil,
    memory: nil,
    active: nil,    [illegible]
    name_list: nil, data: nil,
    [illegible]: function(name) { this->tick = get CurrentTick();
        this->active=[];
        var m = Memory['timer'];
        if (m === undefined) { m = Memory['timer'] = [[],[]]; }
        var md = m[1];
        if (md === undefined) { md = m[1] = []; }
        var mnl = m[0];
        if (mnl === undefined) { mnl = m[0] = []; }
        if (md.length() !== mnl.length()) { if (md.length() > mnl.length())
          { md = []; } if (md.length() < mnl.length()) { while (md.length() < mnl.length()) {
          md.push([0.0, 0.0, 0.0, md.length()]); } } }
        this->memory = m; this->name_list = mnl; this->data = md;
        return this->start(name);
    },
    tick_complete: function (pos=nil) {
        this->stop(pos);
        var a = this->active;
        if ( a.length() > 0) { this->report Error(); a = []; }
    },
    clear: function() { var m = this->memory = [[],[]];
        this->active = []; this->data = m[1]; this->name_list = m[0]; },
    stats: function (sort, column) { /* to be done later */ },
```

4-15

```
start: function(name) {
  var nl = this.name_list, l = nl.length, pos = -1;
  for (; ++i < l; ) { if (nl[i] === name) { pos = i; break; } }
  var d = this.data, dp;
  if (pos === -1) { pos = l; nl.push(name); d.push(dp = [0,0,0,0,0,0]); }
  else { dp = d[pos]; }
  this.active.push([dp, gr(cpu)]);
  return pos;
},

stop: function(pos) {
  var dp = ...(pos), a = this.active, l = a.length;
  if (l === 0) { this.reportError(); return false; }
  if (pos !== -1 && pos !== a[l-1][0][4]) { this.reportError(); return false; }
  var c = a.pop(), dp = c[0], tick = this.tick;
  dp[0] += cpu - c[1]; dp[1] += 1;
  if (dp[3] !== tick) { dp[2] += 1; dp[3] = tick; }
  return true;
},

add: function(name, cpu_time) { // expires from start
  var nl = this.name_list, l = nl.length, pos = -1;
  for (; ++i < l; ) { if (nl[i] === name) { pos = i; break; } }
  var d = this.data, dp, tick = this.tick;
  if (pos === -1) { pos = l; nl.push(name); d.push(dp = [0,0,0,0,0,0]); }
  else { dp = d[pos]; }

  dp[0] += cpu_time; dp[1] += 1;
  if (dp[3] !== tick) { dp[2] += 1; dp[3] = tick; }
  return true;
}
```

7.1

Nanites guild

The primary concept behind this guild is the use of tiny robots (nanobots) called nanites. The guild is based around a corporation that hires players in order to use, test, and generally advance their technology. The players get no income from the employment, however they do get an "employee discount" on purchasing higher level nanites. The entrance to the guild has 4 exits: 1) leads back to the main city; 2) leads further into the building (guild members only); 3) leads to an operating room where players can join or leave the guild (? limit to adventurers or guild members only); 4) leads to a donation room where players from other guilds can donate equipment (which gets moved into primary donation box) or money (which can go to every guild member currently logged in and not link dead, or to a specific guild member; this money can not be withdrawn and can be used for any guild purchases). Inside the guild should be rooms containing the following: 1) donation box; 2) general posting board; 3) nanite use board; 4) area info board; 5) random post board. The guild should have 2 rooms to install nanites. The first that is entered will be a "pre-order" room. Players can pre-request a nanite injection pack and have it waiting for them in the future. After reboot, nanites must be fully replenished (guild run required before reboot runs), so this allows a way for it to happen quickly if the player prepares for it. The first room should have an exit to the second room. The second room has an NPC (Bob, the nanite distributor and application specialist, and sometimes when Bob is out, Jenny the nanite application technician; possibly others to keep an interesting "8 hour shift" concept going, possibly with breaks and lunches),

who oversees preparing and injecting nanites. Requests are handled through a computer interface. Multiple packs can be created by a player and saved with a unique name for faster access. The player can request a code to share the pack setup with other players. Once the request is entered and paid for, it will take approximately 10 seconds per nanite type to prepare the injection. If the player leaves/goes link dead, Bob(etc) will hold it for 30 minutes, then send it to the pre-request room for storage. Other than nanites, there will be 2 other primary objects: 1) a wristband that can have components installed into it, starting with 2 and increasing at certain guild levels, probably up to 5 (player junior admin having access to +1 component; player senior admin hopefully 1, maybe 2 players) having access to +2 components); 2) a small band that fits around the neck (possibly, power thereof as well) (possibly limits access to amulets and/or helmets) which is an AI computer monitoring service, [illegible] and is given at a low to mid guild level [illegible] and slowly learns and is able to help the player in one specific, focused area of expertise. The learning speed should be very, very slow, with a very slight speed increase at the beginning (slow as in 5 years of non-stop playing, where the player is able to perform actions that causes the AI to activate every heartbeat, should get it to 100% efficiency. ? Possibly add some degradation of about 10% at that speed if the player is idle for more than a certain amount of time)

Combat nanites

Damage reduction by damage type (possibly 2 types [illegible])

None for radiation

Weapon damage increase

Healing (heal over time; probably a core nanite)

(cont)

1-3

Dodging (possibly a core nanite)

Speed boosting

Nanites that don't directly affect combat, [illegible]
replicator, [illegible] (0-1 [illegible]), white blood cell suppression [illegible] [illegible]

Optical monitoring

Optical [illegible] (adds things in your vision)

Advanced communication (by default, a simple communication nanite is injected with every injection; the simple version is never monitorable for availability, assume it always exists enough to be useful)

Optical weapons analysis

Optical armor analysis

Optical remote vision (used with a tiny device [illegible] from the guild, allows someone to see into the next room; at higher levels, allows remote monitoring of 1+ rooms if the device is placed in it/them)

Optical identification (only for that player; low levels, they can tell it needs to be identified in the name (in inventory / on ground / on others); at mid-levels, sometimes they will see what it really is, [illegible] and not locked in once they see it once; at high levels, automatic identification with at least 75% to 90% [illegible])

Speedy feet (a trance-like ability, only 1 [illegible] at a time, only commands accepted are visible exits, standard [illegible] limits (possibly add this limit on any injection where it is adding a [illegible] that is <25% left) ([illegible])

Optical target [illegible] ([illegible])

Optical perception ([illegible] when [illegible] players [illegible] ([illegible]

(cont.)

7-4

[illegible] [illegible] (Plus shows target health on [illegible])
([illegible] in long descriptions [illegible])

Combat [illegible]:

Increased weapon speed ([illegible])

Nanites affecting NPCs (Requires [illegible] module for [illegible])
(Available at [illegible])

Damage reduction by damage type

Defense reduction by defense type

Damage causing by damage type

?Slowing target combat speed

?Temporarily stopping target from attacking

Stat reduction by stat

Nanites affecting other players (Able to [illegible])
([illegible])

Healing ([illegible])
([illegible])

Dodging ([illegible] at [illegible] guild rank)

Stat boosting

Nanites affecting [illegible] (Requires [illegible] module for [illegible])
([illegible] at [illegible] guild rank)

Weapons: Damage boosting by type ([illegible])

Armor: Damage reduction by type ([illegible])

AI unit types

Offense

Defense

Dodging

Healing

Nanites affecting NPCs

Nanites affecting other players

7.6

(Avoiding anything that affects [illegible])

Wetware [illegible] modules:

Injection module: Allows carrying additional injections of nanites and nanite food for replicating nanites. (?Possibly a separate module for each one)

Nanite management module: Allows info on hp bar regarding some/all current nanites injected and how much is currently in the player's body. (?This will also be available on a guild [illegible] screen). Players can disable or destroy any type of nanite in their body.

Guild donation monitoring module: See when anyone (in or not in guild) donates equipment. Also see when a monetary donation is made to the player.

Remote board notification and reading module: See immediately when a new post is added to any board. Able to read any board from anywhere in game.

Advanced remote communications module (by type): When in a room (and visible) with another player (not in guild) that currently has nanites in them, provides a boost to those nanites' effectiveness (?25%-50% increase)

?How to handle guild chat: Maybe "An earpiece with a GTI logo" which would be a visible guild [illegible]. Company name: Global Technology Industries

Nanite levels: There will be 4 standard level rankings (newbie, basic, standard, advanced) and 1 extra (experimental, it will occasionally backfire). Each use of a nanite [illegible] gets it closer to the next level. Player cannot monitor advancements, they only know a new level is available in the guild (maybe [illegible] will tell them)

Player guild levels: Based on nanite level data.

Player guild ranks: Four (non-admin) total, similar to nanite ranks, names yet to be determined

Nanite replication: At some point (probably beginning of advanced non-replicating nanite version), the player can get access to replicating nanites. This allows (with a specialized injection port and nanite food - which is cheaper than meat) the player to get their nanite levels back up, or keep them from falling, while out of the guild. The replicating version is considered a new nanite and must be trained starting at a basic level again. ? Perhaps make new levels that start slightly, several times, easier to replicate faster than the previous. Base replication speed should be 1) on a heartbeat where they are used, 1% increase based on how much is in their bodies (10% left; 10.1% after that heartbeat); 2) when not used, 5% increase based on how much is available in their bodies (20% left; 21% after that heartbeat). Nanite food in the body does not degrade unless used for replication

Nanite degradation:

No nanites are foreign bodies inside the player, the player's immune system will constantly be battling to remove them. Active nanites (healing, dodging, etc) are noticed more while they are active and removed faster. Passive nanites (optical, radio) that generally remain in place and thus don't need to move or seem to do anything (to the immune system) are removed slower. This causes a constant need to replace nanites to keep the levels high.

7-7

Nanite weaknesses:

Primary weakness is versus radiation damage. Radiation will cause boosted nanite degradation (on all currently injected nanites) based on damage amount. Secondary weaknesses are poison and toxin. Primarily healing nanites have a significant degradation based on damage amount (the only way they can deal with it is to absorb it and enter the blood stream to get it out of the body). Defensive nanites for poison and toxin should reduce the damage and thus reduce the healing nanite degradation.

[illegible] modules

Living target nanite application module: Required for a guild member to use nanites that affect NPCs or other players

Object nanite application module: Required for a guild member to use nanites that affect objects in the game.

The guild donation box should show who first donated the item and who last donated the item. Keep track of donation box donation value; primarily to limit players that take and never return items.

Interest discount on nanites/guild store: Only happens on a new reboot using information from previous reboot. Money deposited and not yet used, money invested, and donation box donation value increase (possibly with monitoring item usage), are all used to calculate if an additional discount happens. Discount is always 20% and lasts until reboot or for a few days.