Mass Drivers pt. 2 – Explosive Change

A Continuation of Part 1 – Tackling New Challenges

From the Depths (FTD) – Quick Summary

An obscure Indie game that was in open access for 7 years with a small but dedicated following that is a favorite of mine.

FTD is a voxel-based combat vehicle sandbox game that allows players to build vehicles from design to implementation. Everything in a vehicle from the engine to the armor is built from the ground up.

With a variety of different weapons, engines and armor types to choose from, designing involves trade offs and deliberate choices that give the game incredible depth and complexity.

What's Changed with Mass Drivers?

- An update to how collision damage is calculated completely nerfed the damage of kinetic projectiles into the ground.

Pre-nerf damage

1 mil – 800k

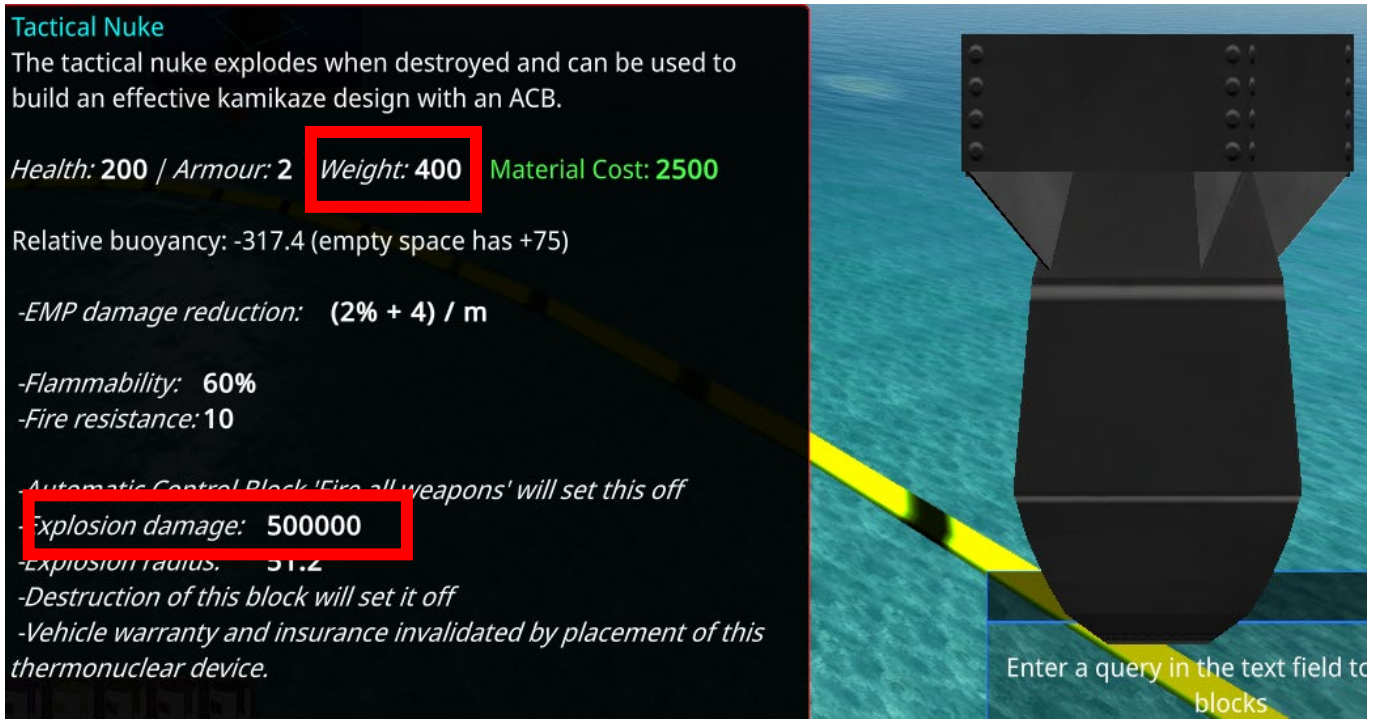
Post-nerf damage

1k - 500

For reference, a 4m metal beam in FTD has 1680 health

- But there is still an option available to us, using the projectile not as the main damage source, but as a delivery mechanism for an explosive payload.

The Payload in Question



- Let's look at the two most relevant stats for the mass driver, the weight, and explosive power.
- Clocking in at 400 weight, the tactical nuke itself is nearly 20x the weight of the old kinetic projectiles.
- This means much more power is needed to accelerate this to the kinds of speeds that we desire.
- But, clocking in at a base of 500k explosive damage, this weapon packs quite the punch, especially considering its small 2m^3 size, making it a perfect substitute for collision based damage.

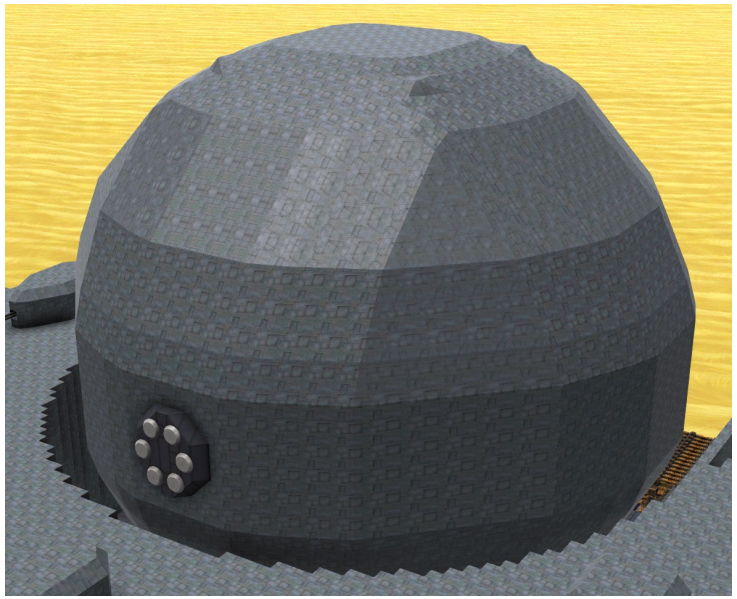
Why Still Pursue Mass Drivers?

- With the switch to nukes, the incredible cost efficiency of the old design is trashed, so why bother still?
- Simply put, the range. In theory, a well built system should be capable of hitting targets in excess of 40+ km. For context, typical engagement range in FTD is 1 to 2 km, with 3+ km being outside the effective range of most weapon systems.
- Along with that, the speed of the projectile makes it effectively impossible to counter, since it only stays in the effective range typical weapon systems for 2 or 3 game ticks. That isn’t nearly enough time to shoot the projectile down.

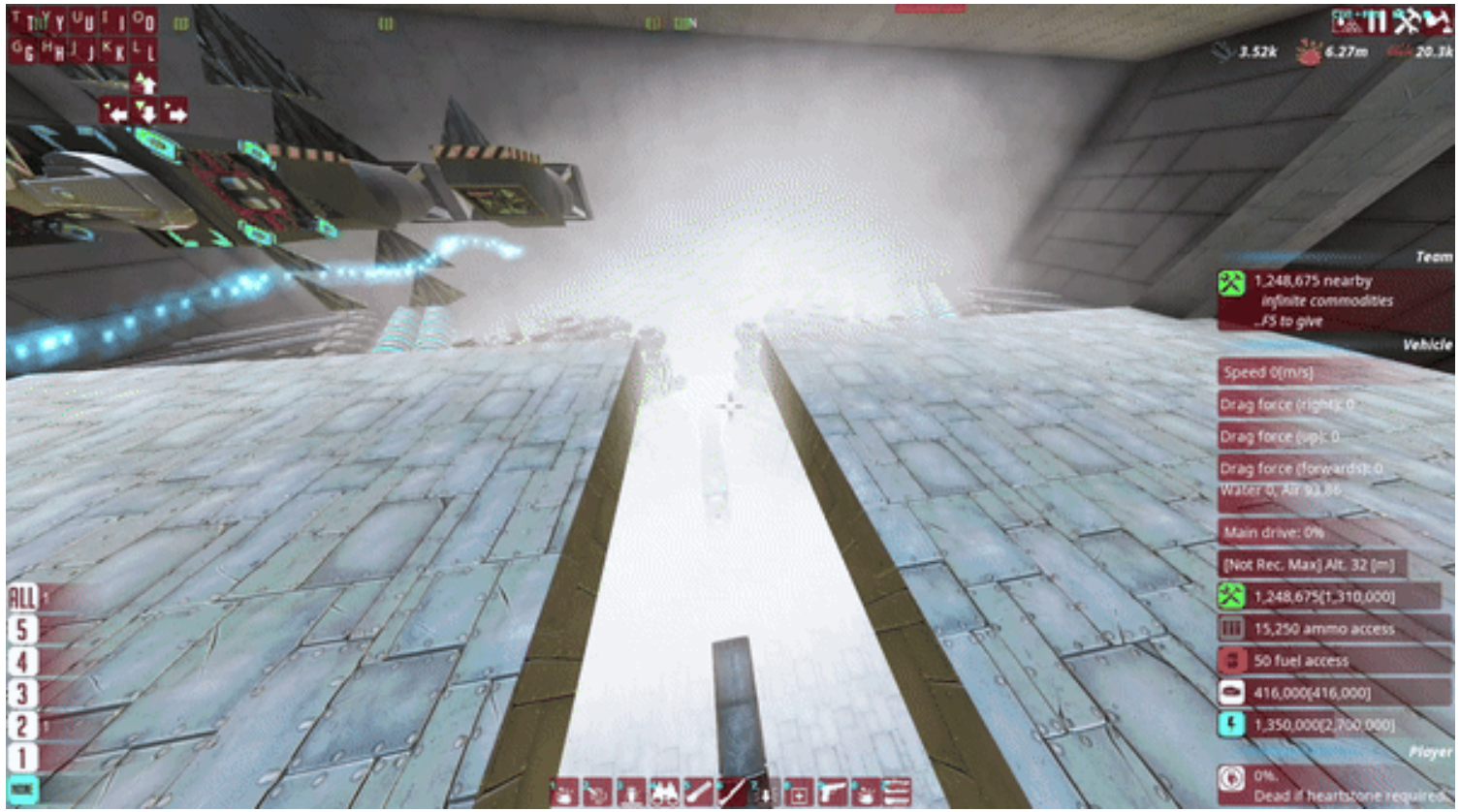
Mk 2

- First of 3 new design iterations
- Still using the spinblock method for power regulation.
- Sustained rate of fire of 15 RPM (4 second cycle).

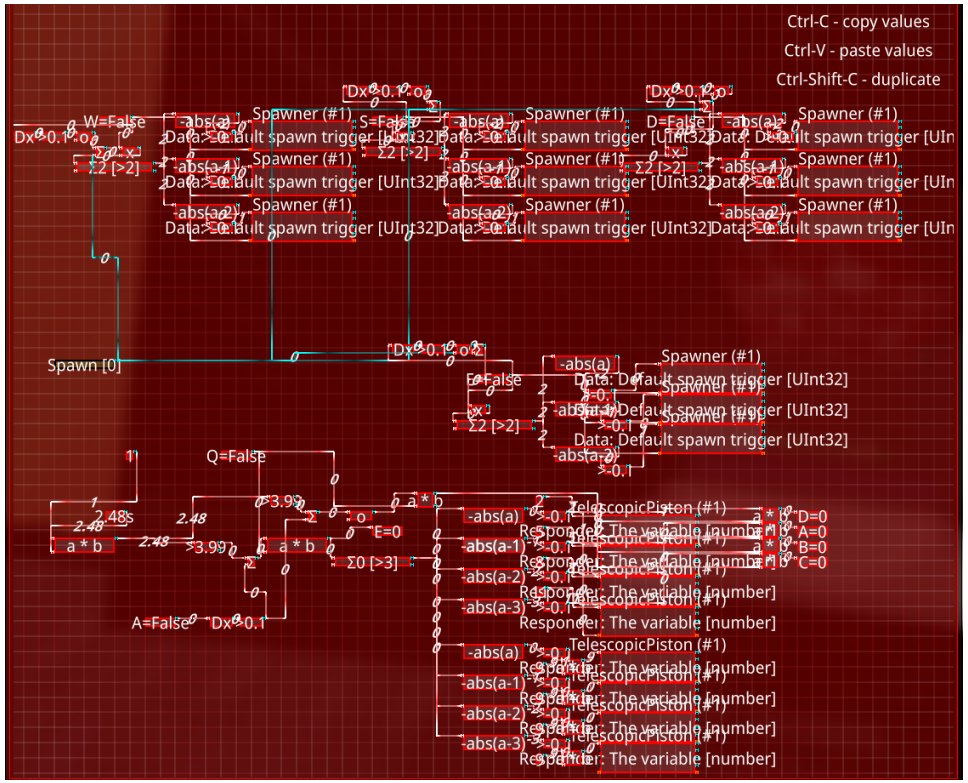
- Custom projectile creation system to bypass 30 second. cooldown on vehicle spawners
- Massive armored sphere to allow the driver to engage in 2 axis rotation while still being protected.



Mk 2 Firing



In case you're wondering, the smoke is coming from the smoke warhead components on the graviton ram rounds. In order for the shells to detonate after they hit the projectile, they need a payload, and smoke is the only payload that won't cause damage to the rest of the vehicle.



Control system for the spawners (more on this in the next slide)

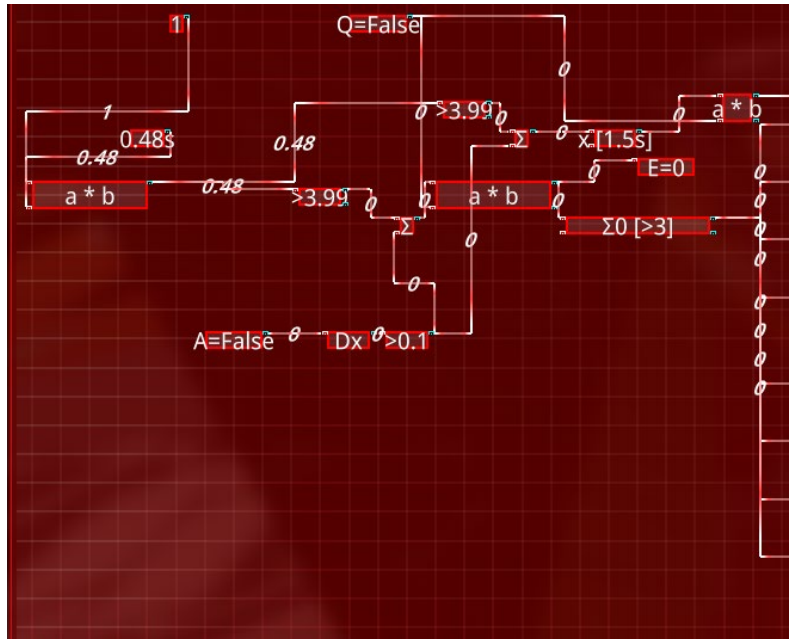
Spawner System

Vehicle spawners have a 30 second cooldown between spawns. This is far to slow for the 15 rpm I want to achieve, so I created a system that cycles thought 12 different vehicle spawners.

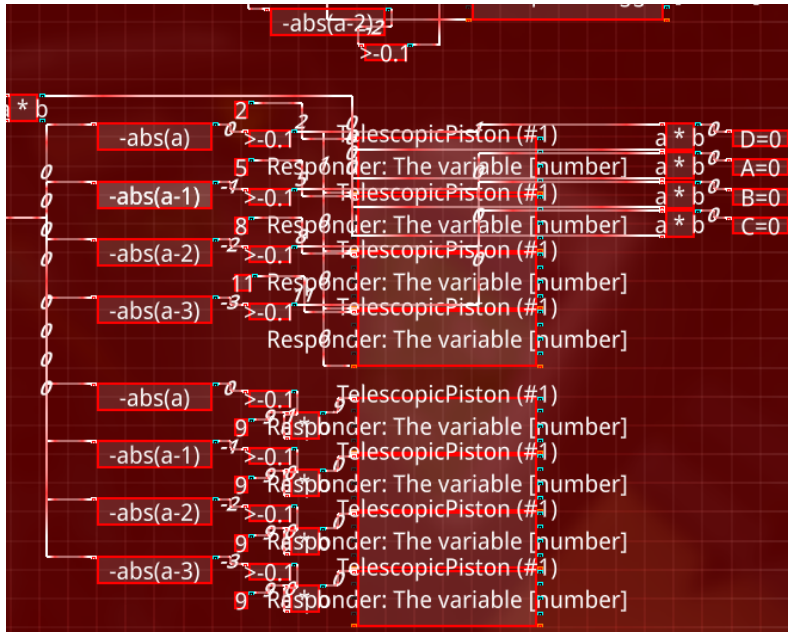
The next two versions use a system similar to this. I'll explain this in more detail so I don't have to go into the nitty gritty with the other two.

There are 3 main parts to this system.

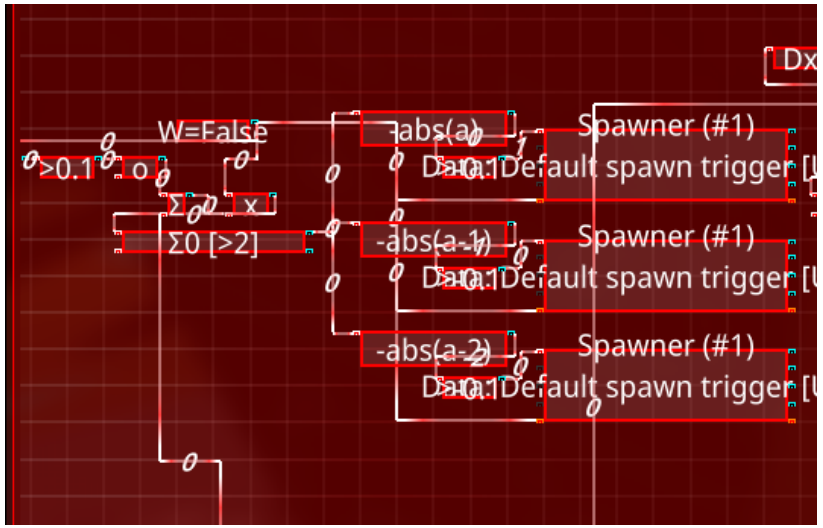
Ticker



Piston Governor

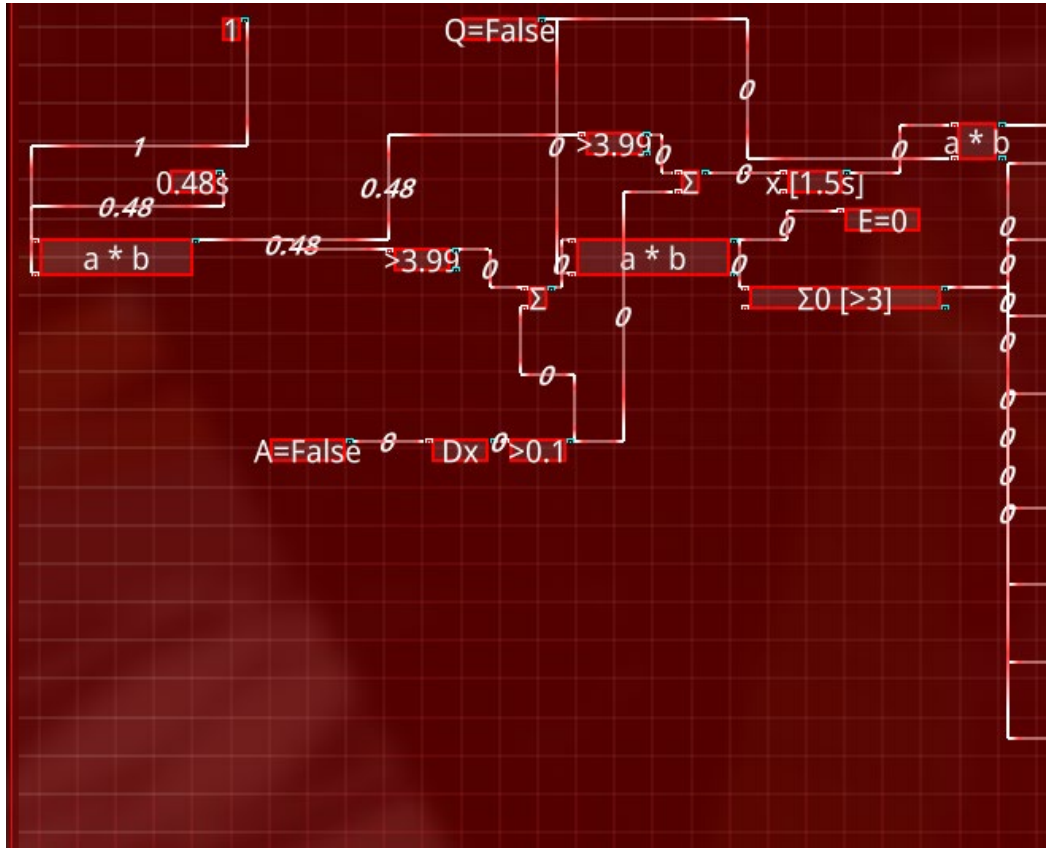


Spawner Governor



Spawner System

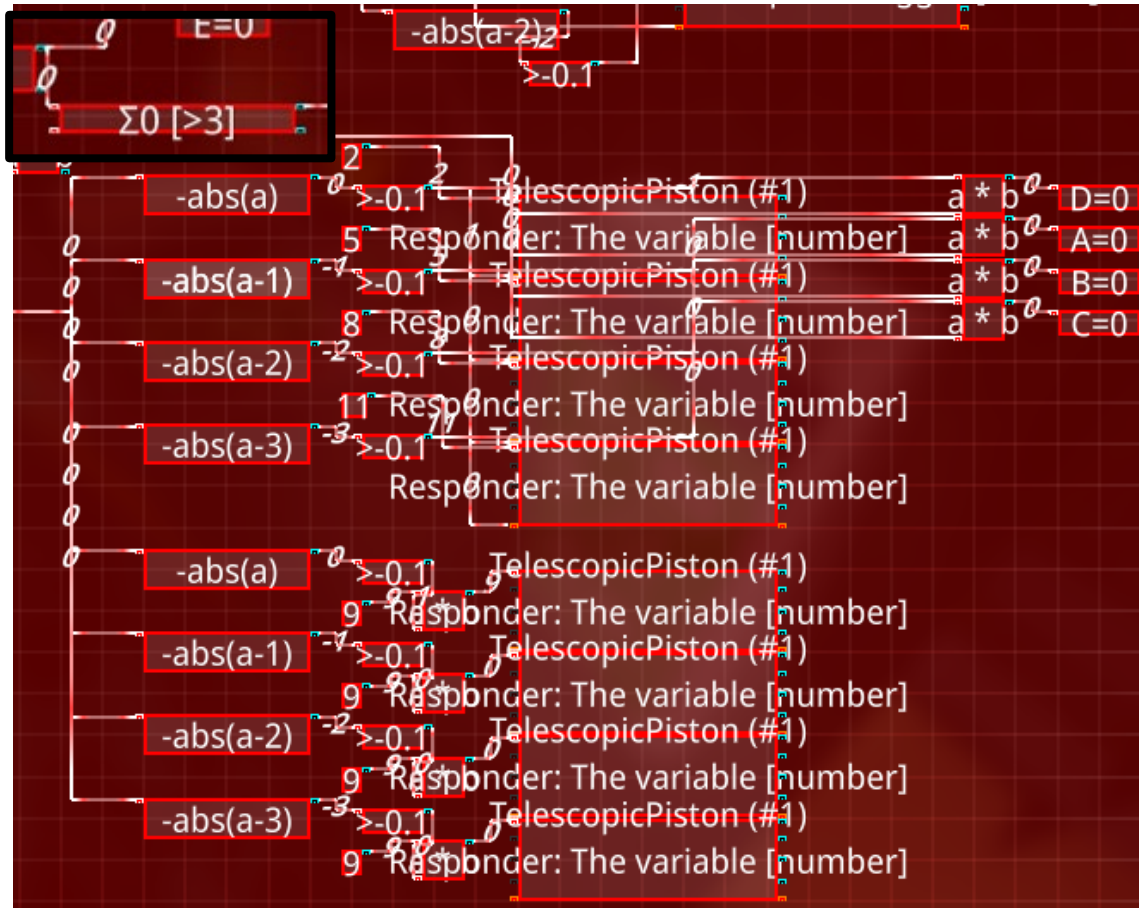
Ticker



- The ticker is a timer that produces a pulse once every 4 seconds.
- There is also logic setup to turn the ticker on and off and to allow for manual firing of the mass driver for testing purposes.
- The “Q=False” is the on/off switch. Those inputs are either 1 or 0, so by multiplying the Q input by the pulse signal, we can control the pulse.
- The “A=False” is the manual pulse, which can be combined with turning the constant value on the right side to zero to grant full manual control of the cannon.

Spawner System

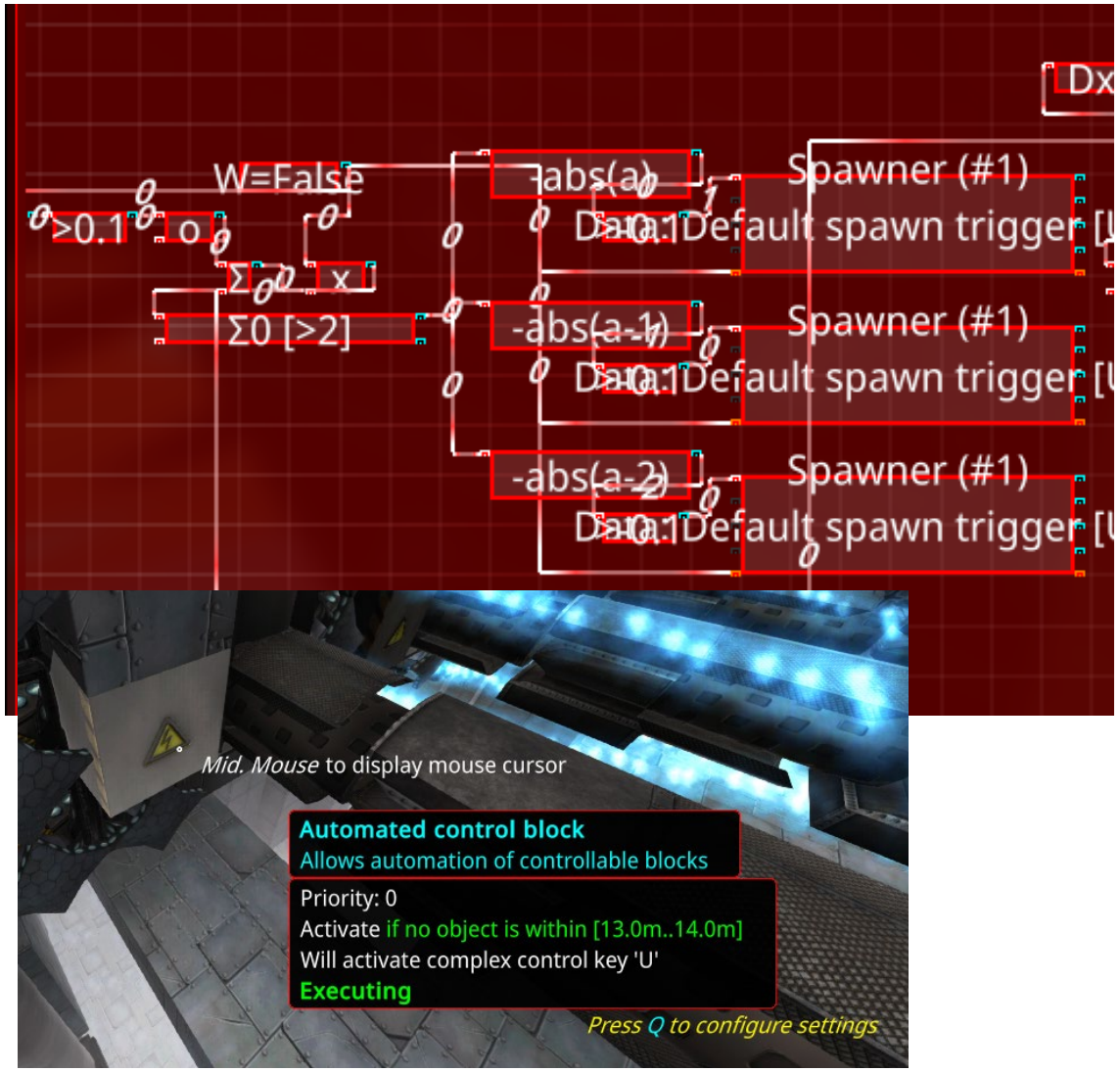
Piston Governor



- The piston governor is responsible for cycling through the vehicle spawners and lowering the correct set of vehicle spawners down into the firing position.
- The pulse from the ticker is fed into a “summation” component which goes through 0 to 3. This number is then read through a formula (abs(a-x) where x is the value that makes the formula return 0.
- Then there are “threshold” components that only return true when their input is > 0. Set up correctly this system will cycle through 4 different outputs.

Spawner System

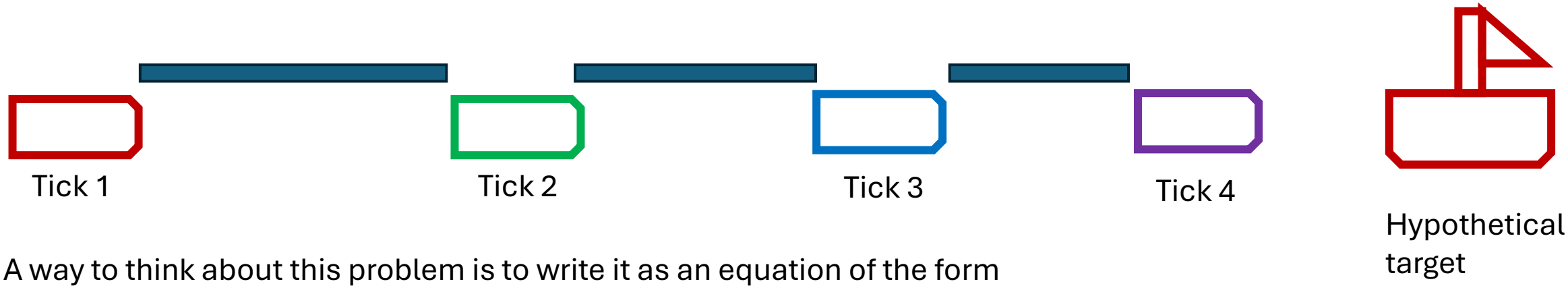
Spawner Governor



- There are 4 groups of spawners. Each group has an automated control block (ACB), pictured in the bottom left) that checks if there is a projectile currently loaded in that particular group.
- The spawner governor reads this input and checks when a change occurs (no projectile -> projectile, etc). We toss out the no projectile -> projectile change, and only pulse the system when we go projectile to no projectile since we only want to try to spawn a new projectile once the current one has been fired.
- Similar to the piston governor, this input is fed into a summation component that cycles through the 3 different vehicle spawners in each group.

Mk 2 Power Selection & the Drag Problem

At the speeds achievable by mass drivers, it's easy to traverse the whole game map in a few seconds. This means theoretically, a target can be hit anywhere on the map. However, we must contend with drag. The distance between each tick decreases and if we don't account for that, the simplistic method of dividing distance by first tick speed fails.

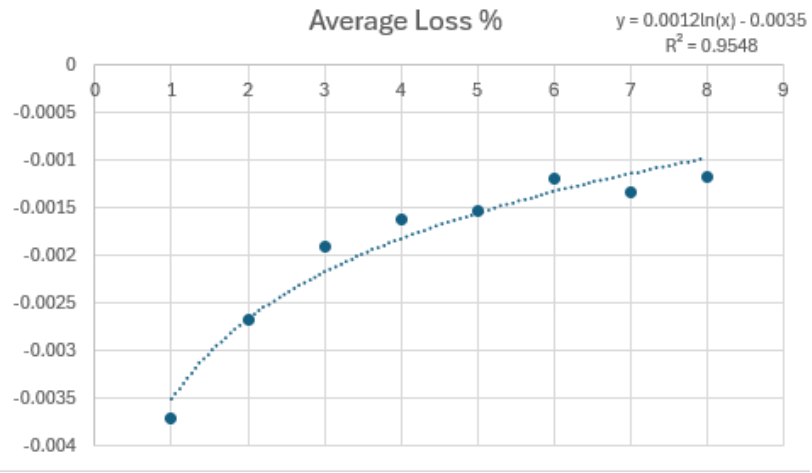


A way to think about this problem is to write it as an equation of the form $(x + c_1x + c_2x \ldots + c_nx)$ = Target distance where c represents what percent that tick is compared to the first tick, so they would be values like 0.98, 0.97, etc. N represents how many ticks we are using. Solving for x we get x = target distance / $(c_1+c_2 + \ldots + c_n)$.

Target distance is easily attainable in game, but finding n and the corresponding c values is trickier.

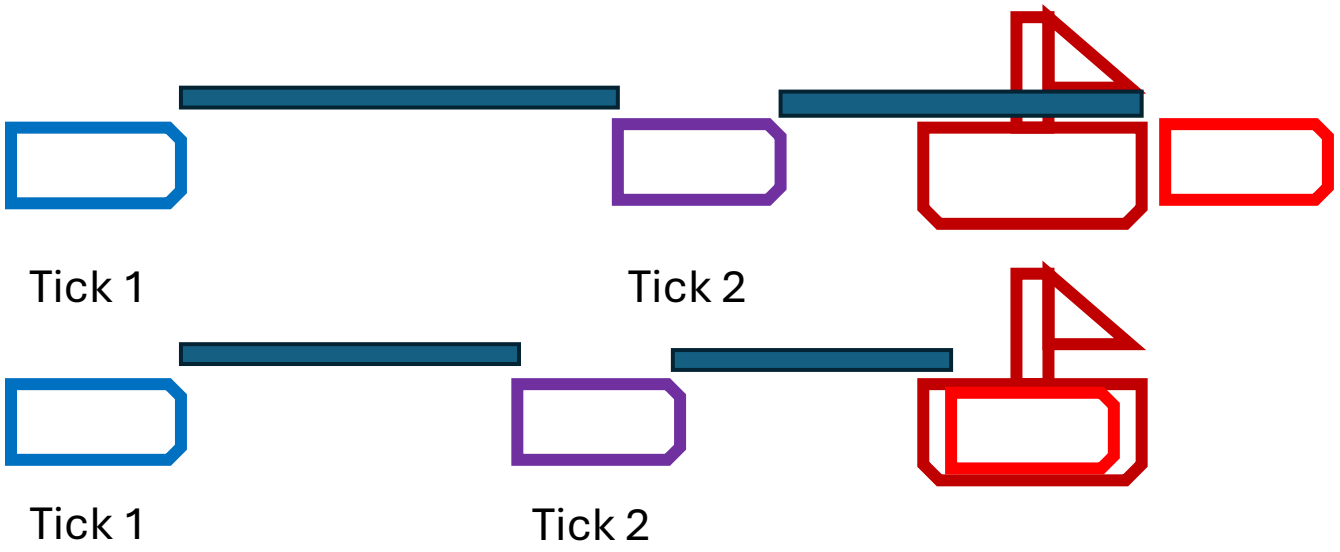
Mk 2 Power Selection & the Drag Problem

Tick	Trial 1	Loss1	Trial 2	Loss2	Trial 3	Loss3	Trial 4	Loss4
Tick1	42986		43000		42978		42880	
Tick2	42768	-0.0051	42774	-0.00528	42769	-0.00489	42784	-0.00224
Tick3	42442	-0.00768	42692	-0.00192	42673	-0.00225	42706	-0.00183
Tick4	42380	-0.00146	42615	-0.00181	42599	-0.00174	42628	-0.00183
Tick5	42328	-0.00123	42548	-0.00157	42524	-0.00176	42556	-0.00169
Tick6	42283	-0.00106	42489	-0.00139	42387	-0.00323	42494	-0.00146
Tick7	42249	-0.0008	42440	-0.00115	42383	-9.4E-05	42396	-0.00231
Tick8	42194	-0.0013	42395	-0.00106	42287	-0.00227	42356	-0.00094
Tick9	42169	-0.00059	42327	-0.00161	42172	-0.00273	42321	-0.00083
		#DIV/0!		#DIV/0!		#DIV/0!		#DIV/0!
		#DIV/0!		#DIV/0!		#DIV/0!		#DIV/0!
		#DIV/0!		#DIV/0!		#DIV/0!		#DIV/0!
Note, Driver at ~70% power								



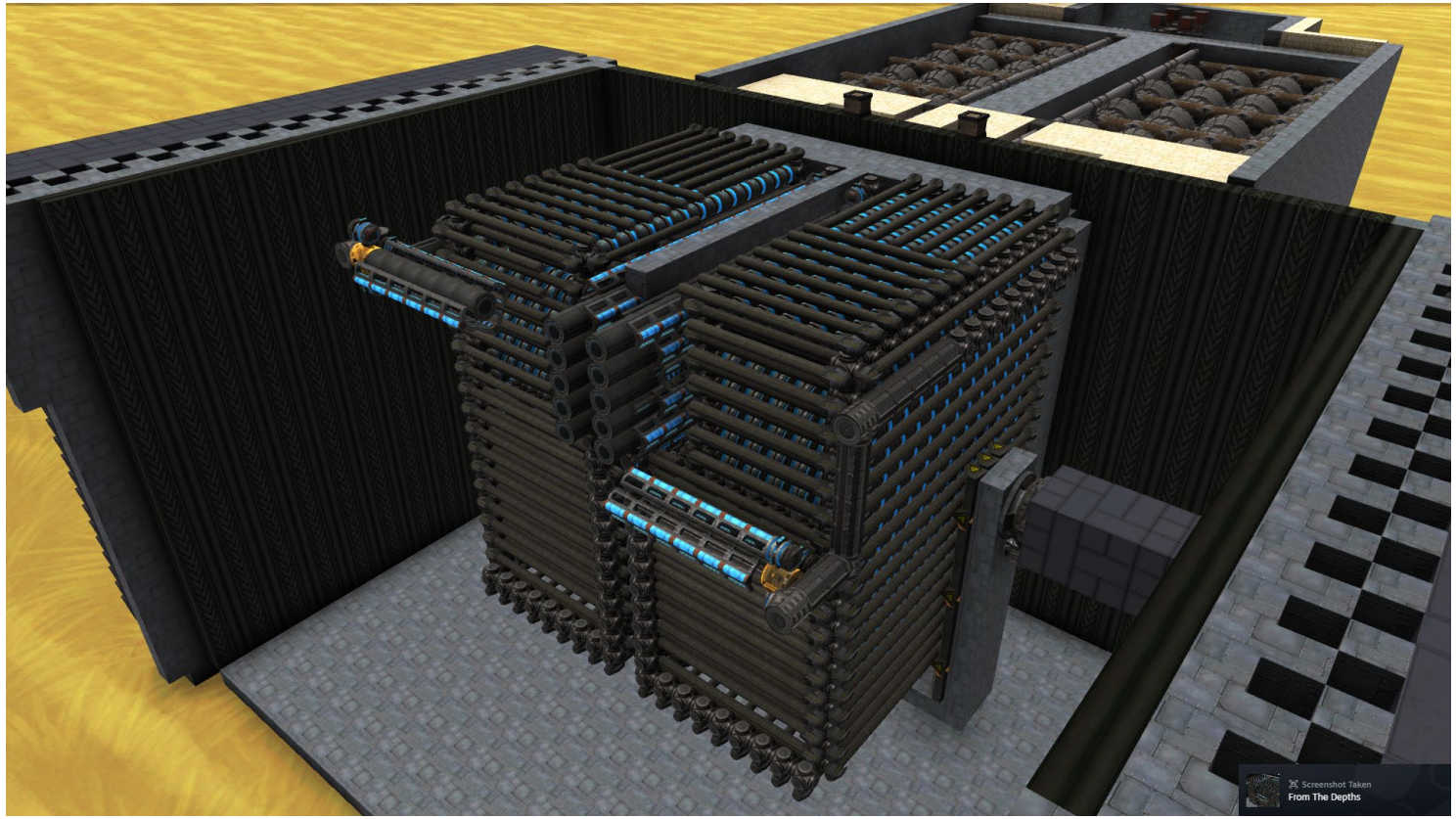
- C and n must be determined empirically through data, in this case looking at the speed data that I got by going frame by frame through gameplay footage I captured.
- Across 12 different trials, I gathered the absolute speed values of each game tick and then calculated the percentage lost from tick to tick. Then, I average the loss for each tick across the 12 trials and plotted the result.
- As can be seen in the bottom left, the loss values follow an exponential decay over time.
- To find n, I first start by taking the average speed of the first tick and dividing by 40 to go from meters/second to meters/game tick. Then, we approximate n as target distance / first tick speed. This effectively approximates n as the number of ticks we would need if there wasn't any drag, and we can get way with this because the drag is small enough to do so.

Mk 2 Power Selection Implementation



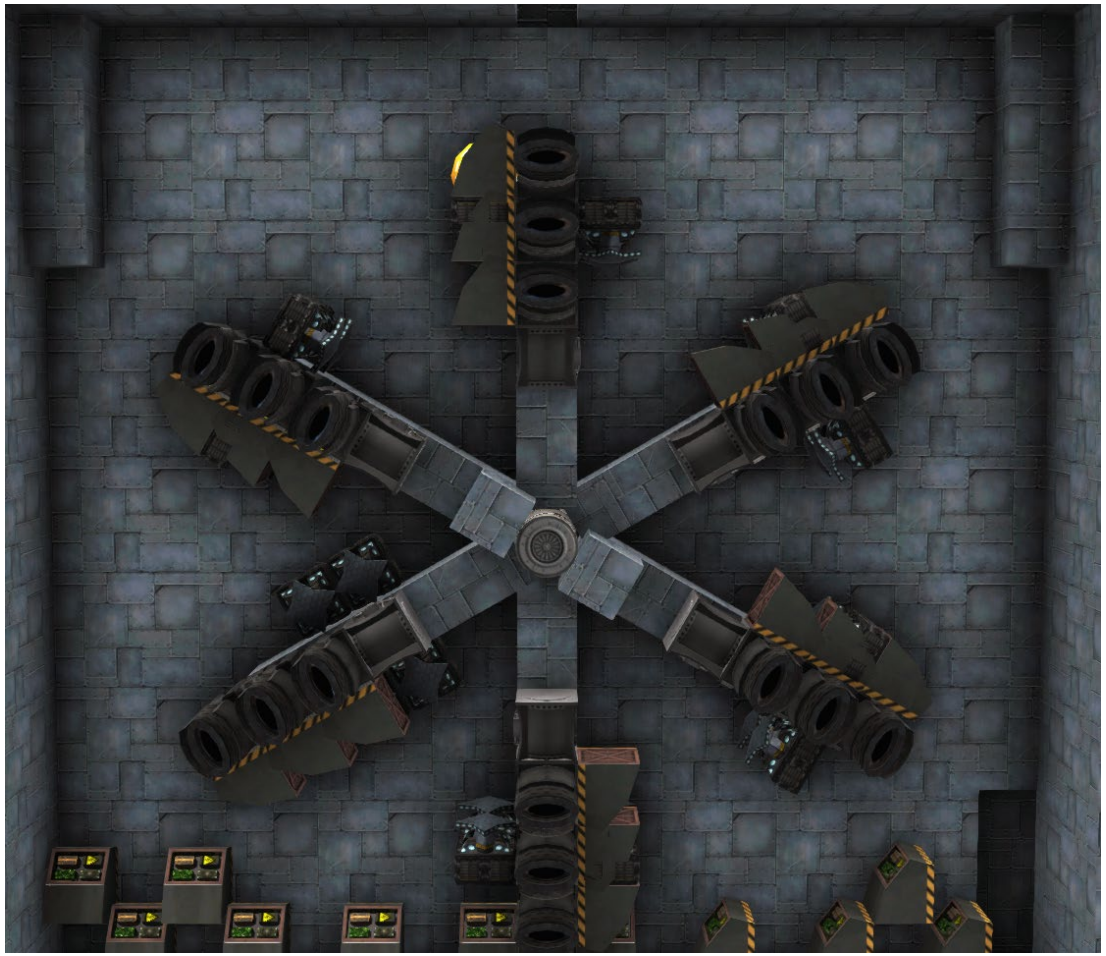
- To find c_n we multiply all the loss values up to the nth tick. What we are finding is effectively what percentage the nth tick is compared to the first tick.
- As an example, if the 2nd tick is 98% of the first, and the 3rd tick is 97% of the 2nd then the 3rd tick is 97%*98% of the first tick.
- Since all the c values are less than one, we end up with a value less than what the no drag approximation would be and, in effect, bring the tick that overshoots the target backwards onto the target.

Mk 3



- Unlike the previous two iterations, this design doesn't use spinblocks for power regulation. This is because a new feature was added into the game that allows for direct control of the amount of power a railgun uses.
- This design is much more compact since the driver cannons don't need to rotate and therefore, take an enormous amount of space to move in.
- The Mk 3 also achieves twice the fire rate of the Mk 2, shooting at 30 rpm.
- Each gun is also at the maximum rail power draw of 200,000. For comparison, the guns on the Mk 2 only operated at 135,000 rail draw.
- The Mk 3 ended up being a testbed for experiments in increasing accuracy in ways I hadn't tested before.

Mk 3 Spawner System



- The Mk 3 uses a revolving spawning system that operates very similar to the Mk 2 system.
- The main difference is that the spawning of the projectiles isn't controlled by spawner sensors but is controlled in a fixed pattern like the pistons on the Mk 2.
- The Mk 2 system wouldn't have been able to sustain the 30 pm since the pistons move too slow to reset the cycle in 2 seconds. Along with that this design is more compact than the Mk 2's setup.

Mk 3 Experiments

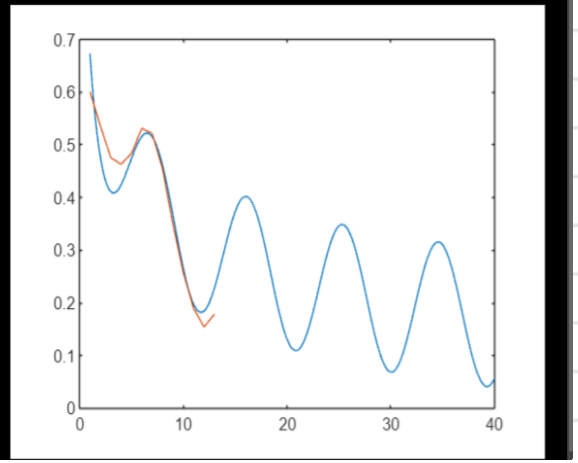
Step 2	0.000677	0.000709	0.00072	0.000667	0.00072	0.000677	0.000709	0.000709	0.000667	0.000656	6.911402
Step 3	0.000602	0.000635	0.000645	0.000624	0.000634	0.000624	0.000645	0.000635	0.00057		6.238436
Step 4	0.000581	0.000581	0.000581	0.00056	0.000592	0.000559	0.000592	0.00057	0.00056	0.000528	5.704595
Step 5	0.00056	0.000506	0.000528	0.000528	0.000528	0.000517	0.000517	0.000517	0.000582	0.000528	5.309429
Step 6	0.000506	0.000485	0.000485	0.000506		0.000485	0.000485	0.000474	0.000604	0.000496	5.02874
Step 7	0.000453	0.000464	0.000442	0.000474		0.000474	0.000442	0.000464	0.000572	0.000518	4.779751
Step 8	0.000421	0.000421	0.000431	0.000485	0.000431	0.00055	0.000442	0.000518	0.000561	0.000615	4.875344
Step 9	0.00041	0.000421	0.000475	0.00055	0.000496	0.000712	0.000486	0.000669	0.000583	0.000659	5.46044
Step 10	0.00041	0.00041	0.000497	0.000583	0.000507	0.000777	0.000507	0.000713	0.000616	0.000821	5.841569
Step 11	0.000453	0.000529	0.000659	0.000735	0.000713		0.000659		0.000778	0.000822	6.684835
Step 12	0.000421	0.000529	0.000659	0.000746	0.000735		0.000659		0.000811	0.000617	6.472951
Step 13	0.00027	0.000519	0.000411	0.000552	0.000443	0.000725	0.000411	0.000639	0.00066	0.000466	5.096701
Step 14	0.000249	0.000454	0.000325	0.000433	0.000346	0.000542	0.000325	0.000455	0.000542	0.000368	4.038287
Step 15	0.000227	0.00039	0.000271	0.000347	0.000292	0.000423	0.000281	0.000358	0.000423	0.000304	3.314188

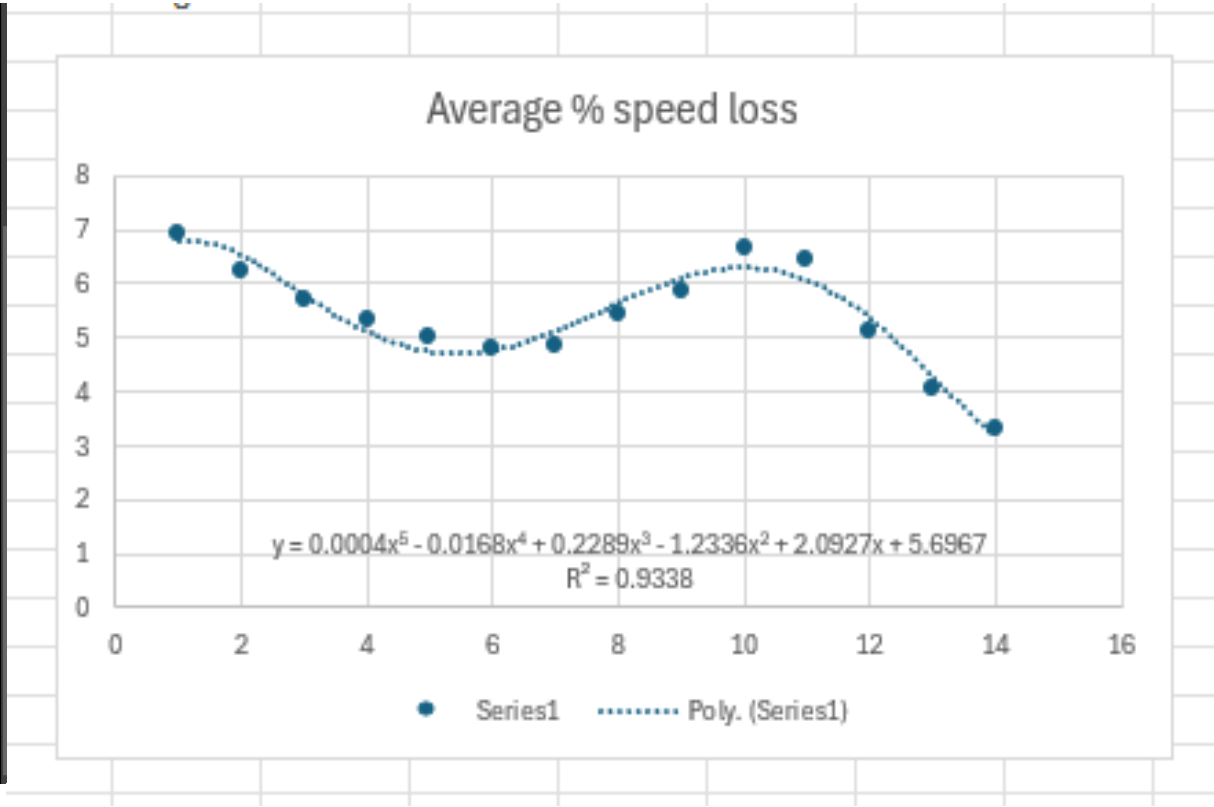
- My experiments with the Mk 3 involved adding spin to the projectile since I had observed it tumbling and becoming unstable at longer ranges.
- Like I did previously, I redid my drag analysis and found that the spin introduced a sinewave into the loss percentages (next slide)
- Excel can't make a good model to fit this data, so I used MATLAB instead through a least squares regression on an equation with both an exponential and sine wave component. (next slide)

Mk 3 Experiments

```
xval = linspace(1,14,14)
xval2 = linspace(1,40,1000)
data = [0.696876843,0.600724526,0.53575022,0.475896621,0
f = @(x)   x(3).*sin(x(4)*(xval+x(5))) + xval.^x(1) + x(
f2 = @(x)   x(3).*sin(x(4)*(xval2+x(5))) + xval2.^x(1) +

score = @(x) sum((data-f(x)).^2)
solution = fminsearch(score,[-0.5,0.5,0.5,1.5,-0.03])
score(solution)


plot(xval2,f2(solution))
hold on
plot(data)
```

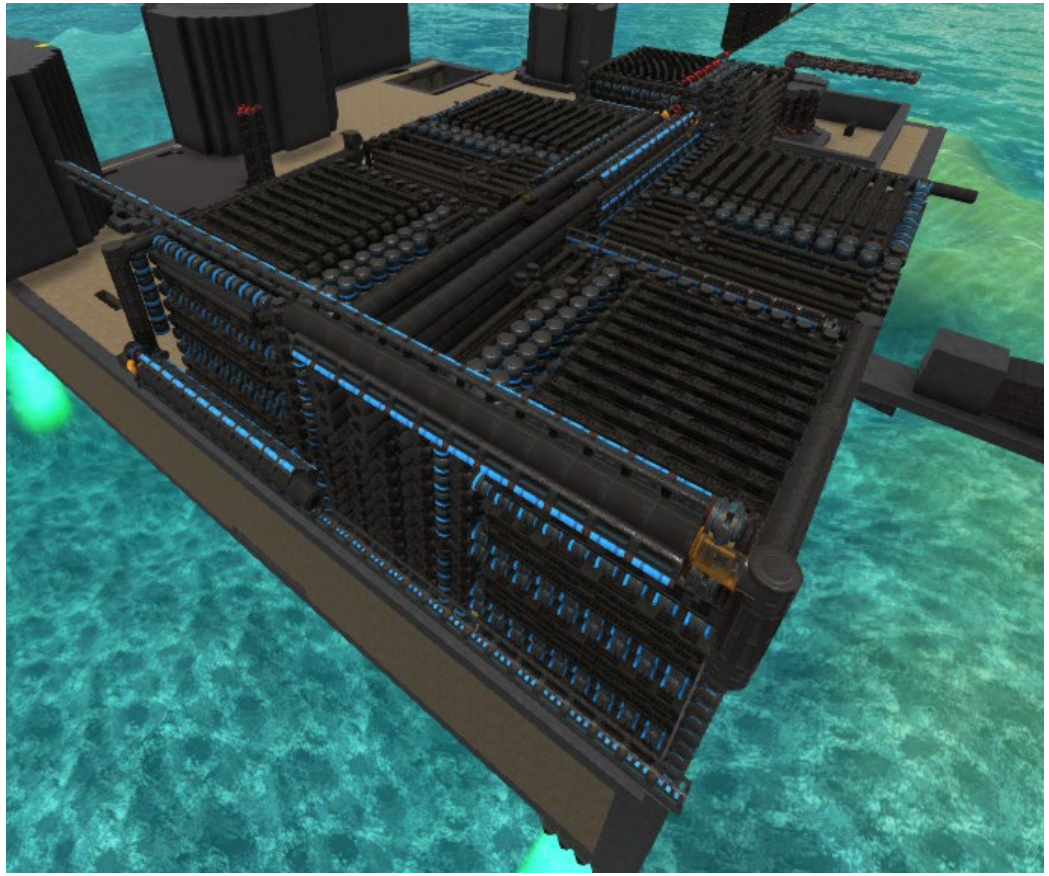
```
f2 = function_handle with value:
    @(x)x(3).*sin(x(4)*(xval2+x(5)))+xval2.^x(1

score = function_handle with value:
    @(x)sum((data-f(x)).^2)
solution = 1×5
    -0.2323   -0.2480    0.0022    2.6353 ...

ans = 0.1184
```





MATLAB code (orange is data points, blue is fitted model) | Excel plot of loss percents

Mk 4 Prototype



- I was unsatisfied with the accuracy of the Mk 3, so I moved on to a new design.
- This design makes use of another accuracy increasing mechanic in the game, specifically tracer rounds. The cannons on this design alternate between the normal graviton ram and a tracer round which gives a 30% accuracy bonus.
- Currently being used as a test bed for experimentation as I try to figure out the ideal setup for maximum effective range.
- Other than that, it uses many of the same systems that the Mk 3 uses