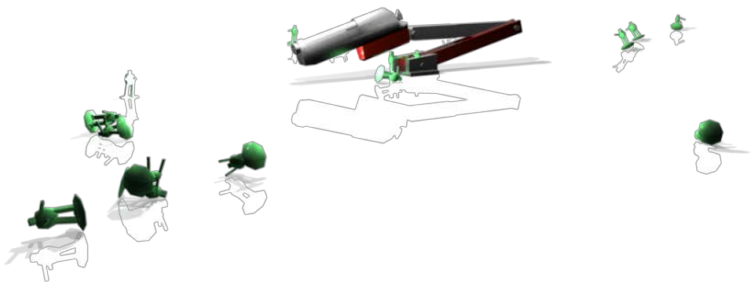
Projec` 3

Posted on November 1, 2011 b. stecman

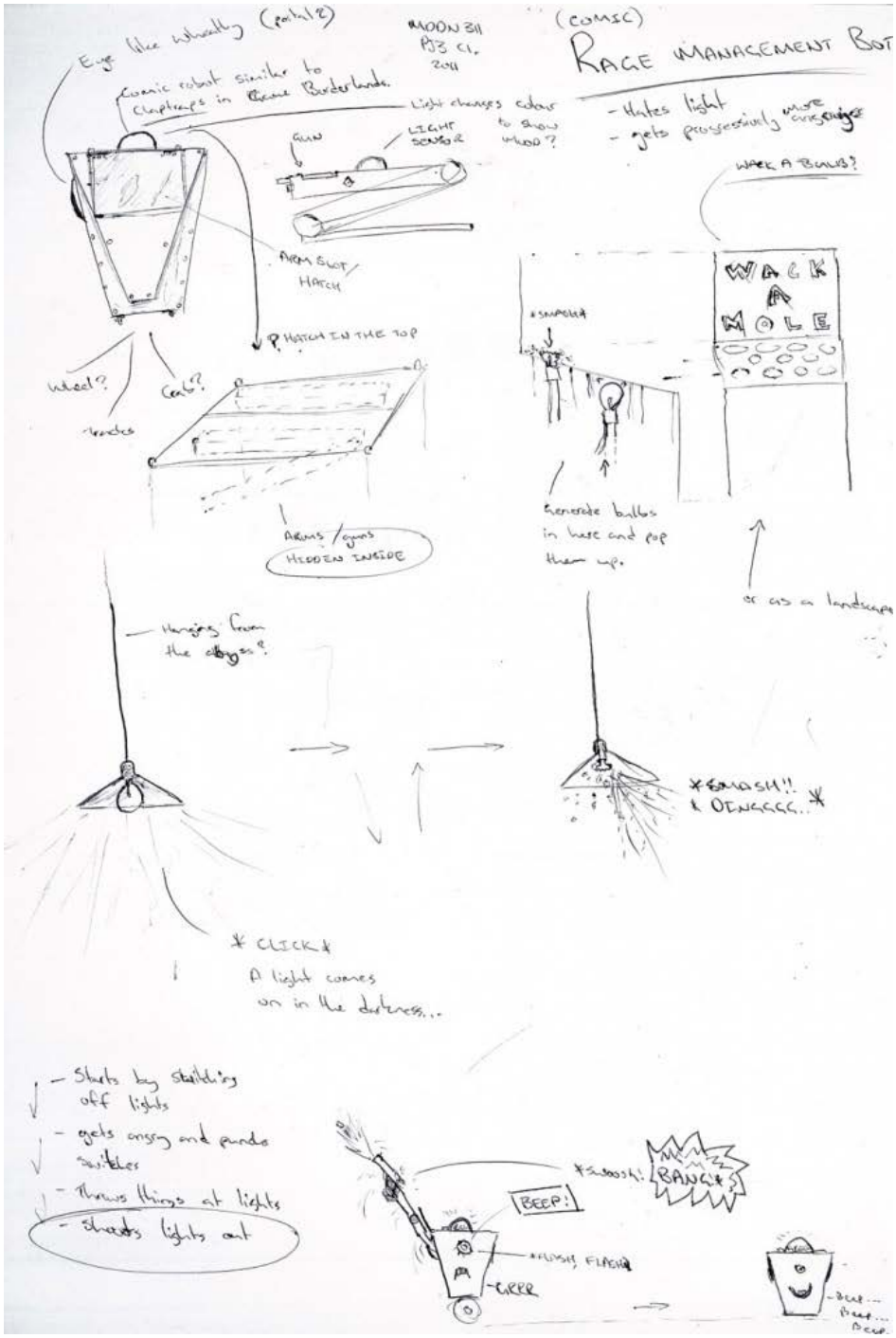
Video: Project 3 Video (5MB)

Source: MDDN311_PJ3_source (200KB)



Concept

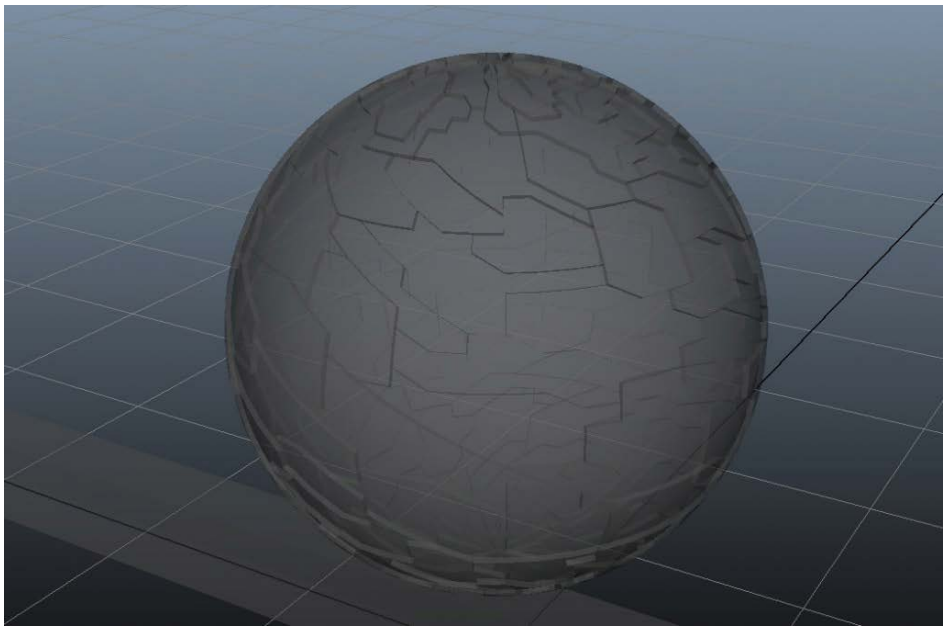
Right from the start of this project, I kne ` I wanted to do something with robots.



My initial inspiration was the . Claptrap" comic robot character from the 2009 video game Borderlands. Lacking any ideas for robot design when I drew my concepts, I used the basic design of a Claptrap as a place holder for a future robot design. As it turned out, I ended up drawing my final robot design on the same piece of paper (arm, top left-ish).

First Experiment

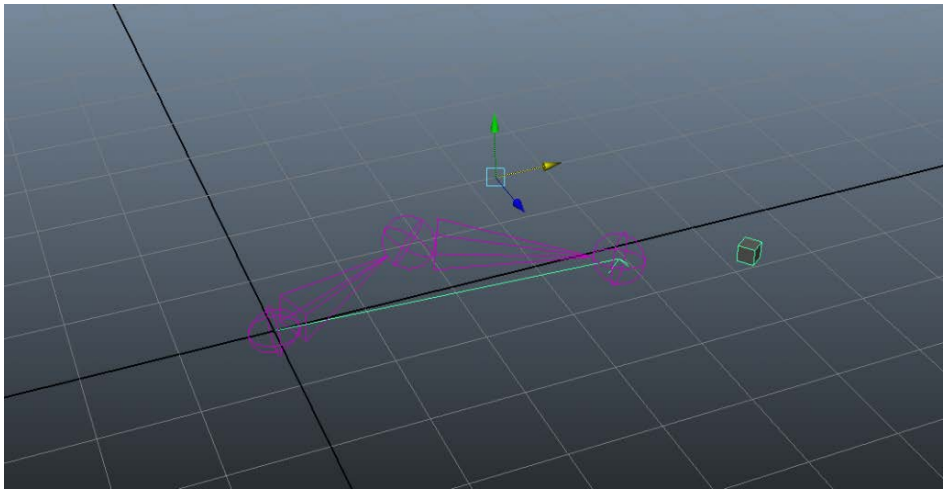
I had considered a whack-a-mole type scenario where a robot just plays whack-a-mole with a machine that pops out light bulbs, so I started with that and looked into generating shatters.



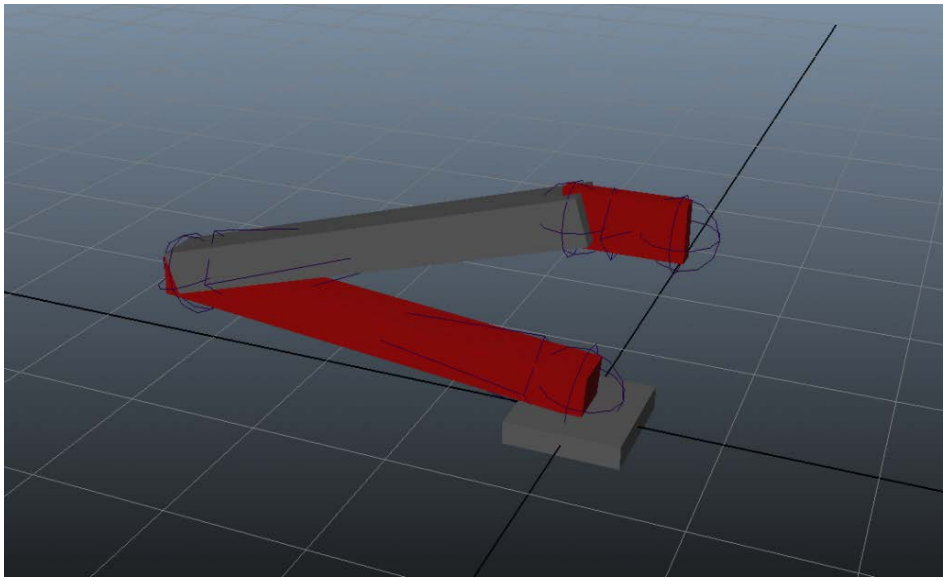
I quickly decided that making effects wasn.t a great place to start as that was something I would be best to finish the project with, so I abandoned modelling completely and got to work on the function and programming side of things.

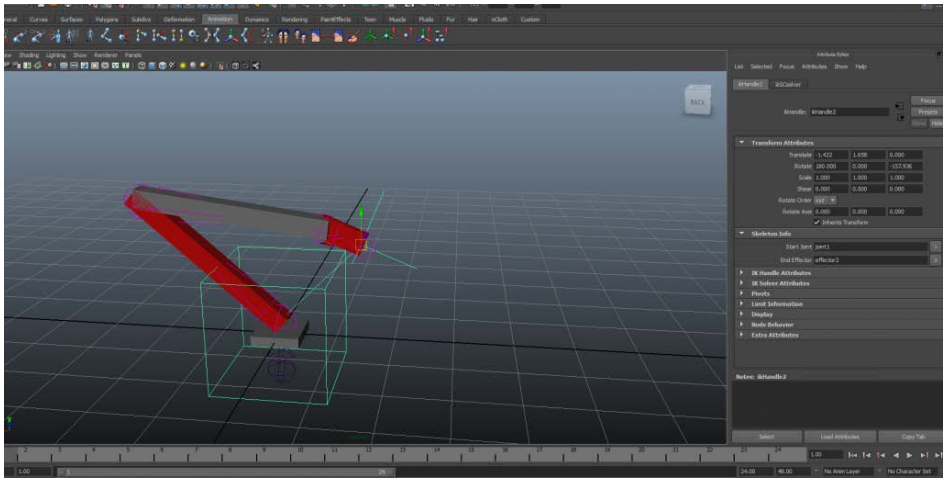
Robot Arm

I wanted to make life as easy as I could in the project, so right from the start I made the interface I had to program with extremely simple – getting as much automation as I could before going to code. First I experimented with a bone hierarchy + IK following a point.



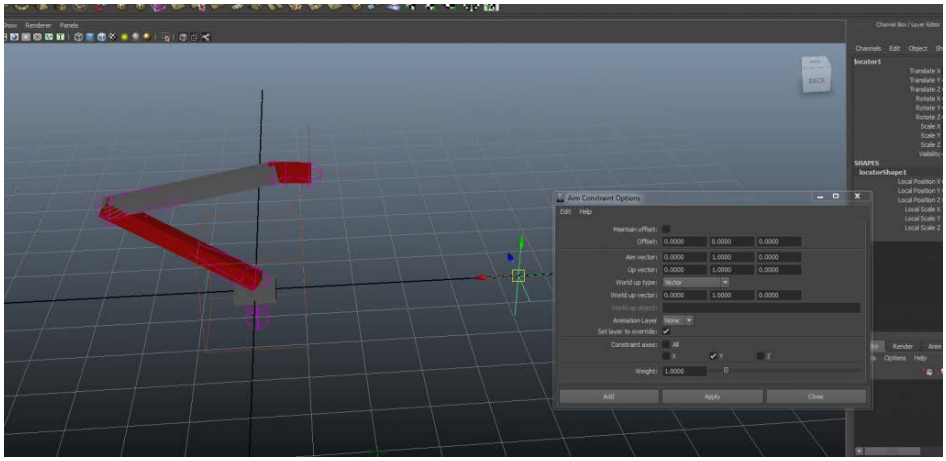
In the above image I had parented the cube to an IK handle. This worked ok, but wasn.t ideal as I wanted the cube to be a target point and the arm to aim at the cube. In this case the arm would stretch out if the cube ventured too far away. This setup also had accuracy and inverting problems as the IK handle was fixed on one side of the cube.

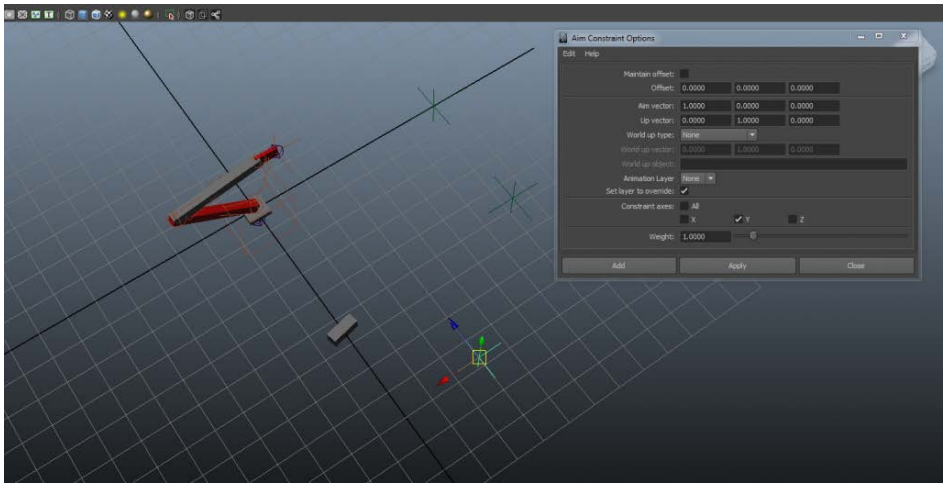




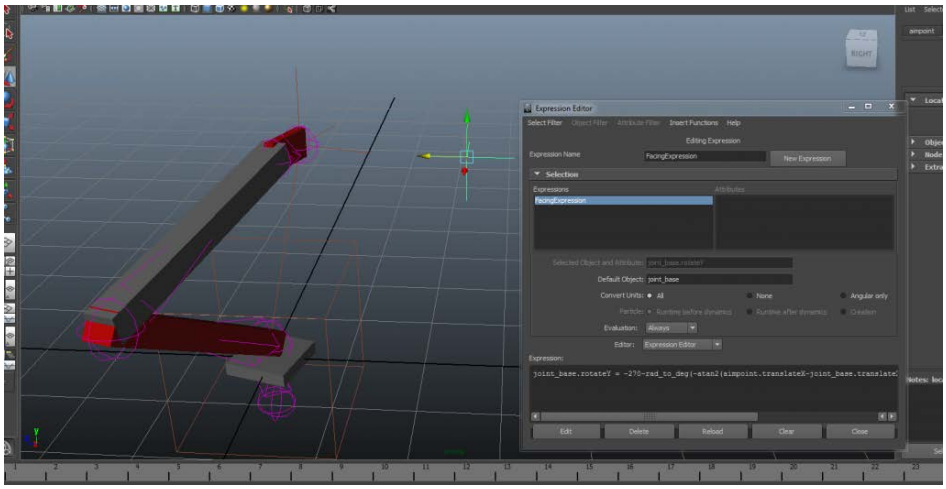
After putting some geometry around the bone structure and adding limits to each bone (to give some robo-realism), I tried out various types of constraints. Once I had found the right type of constraint – aim in this

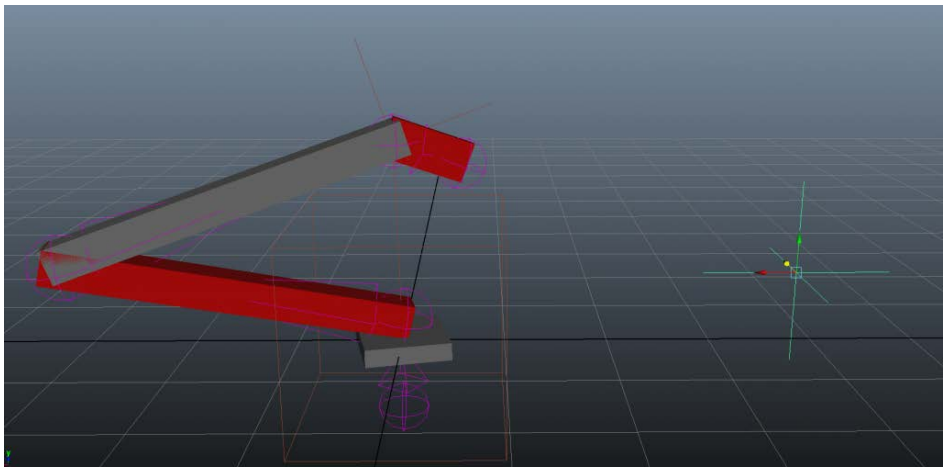
case . I set up the blocked-up robot arm to follow a locator. This process was quite involved as I had trouble with weird rotations when constraining to only one axis (Y).

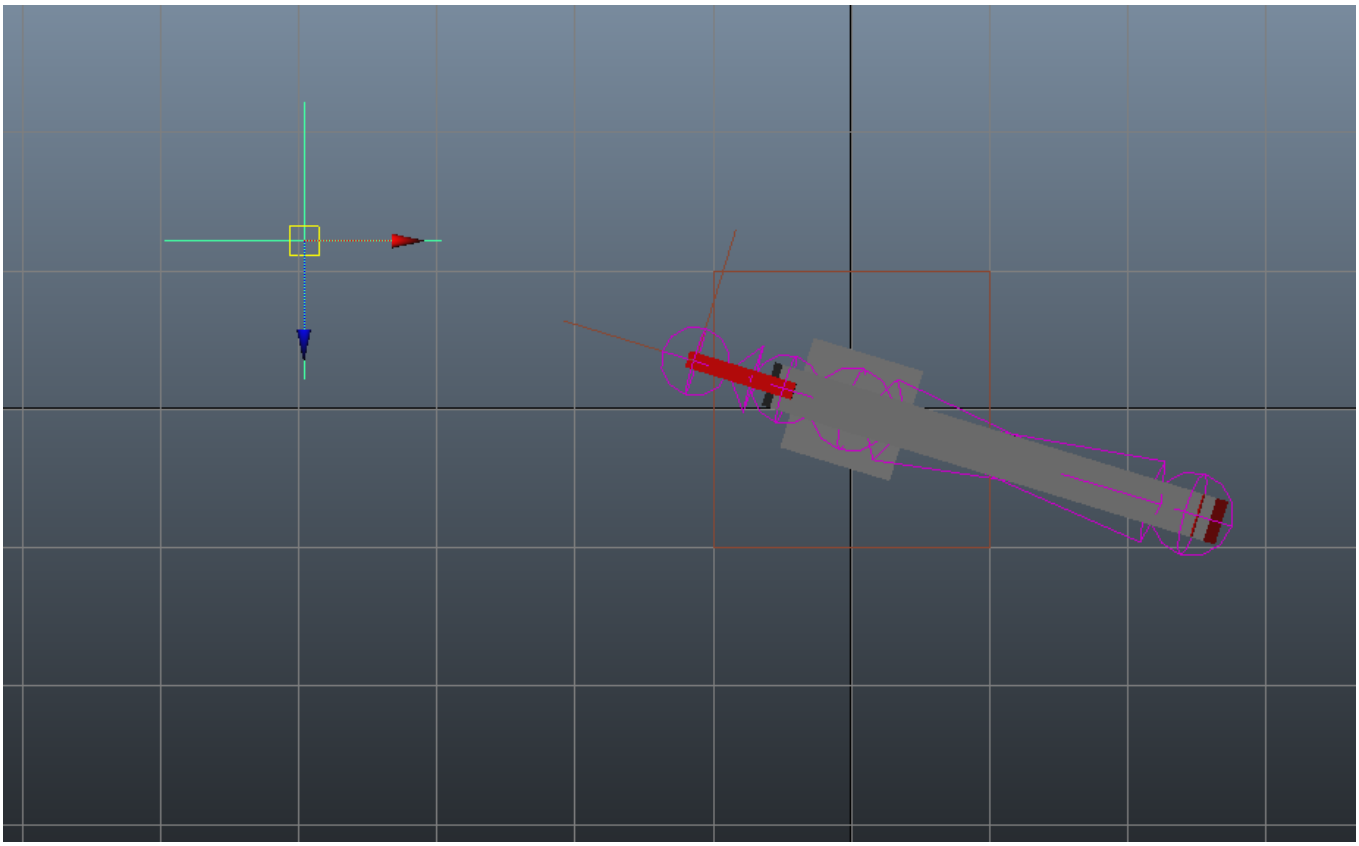




I solved this problem by using expression rather than constraint driven rotation for the robot arm:

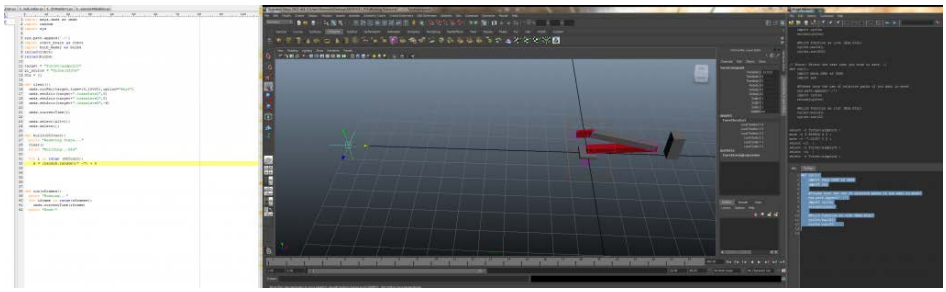




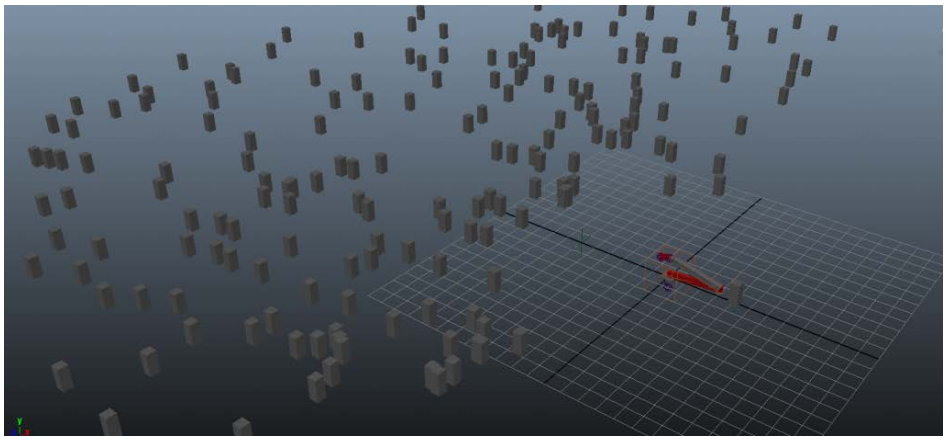


The Robot.s Nemesis(s)

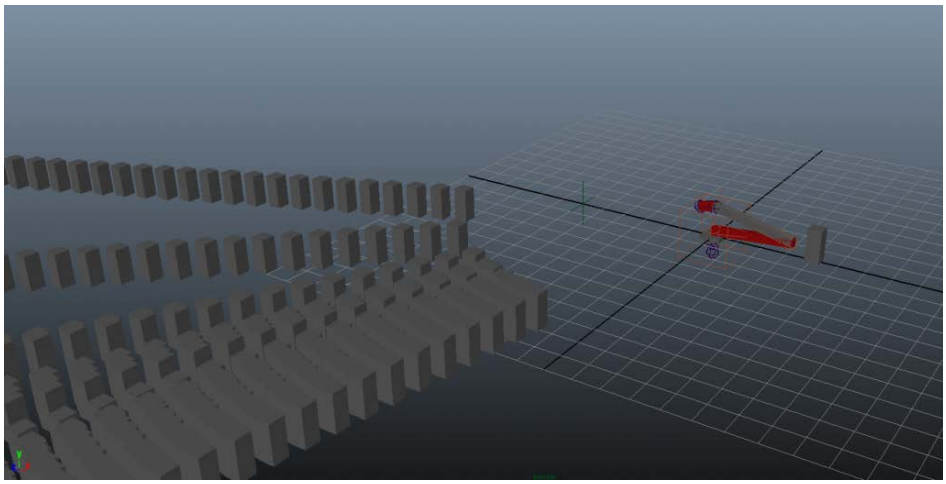
Next I moved onto the target for the robot (I actually started and finished the basic targets before I had solved the aiming robot aiming problems, but that's irrelevant). I referenced the arm and a newly made GI-Joe-block into a new scene and worked on the spawning algorithm for the invaders. I'm not sure at exactly what point my idea changed from lightbulbs and whack-a-mole to GI Joe plastic soldiers, but it did.



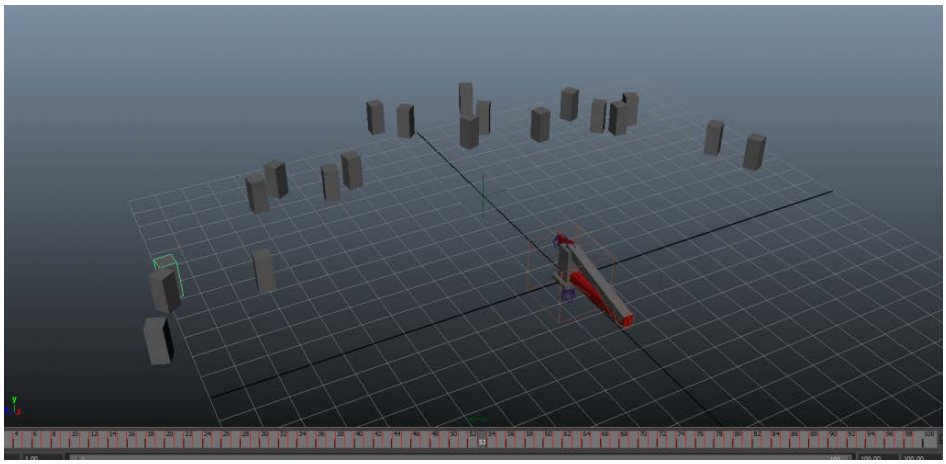
A formation of military precision had crossed my mind, but I thought that would be too boring, so I started with (pseudo-)uniform distribution in a semi-circle.



I tried lines of targets in a semi-circle but it didn.t leave much room for variation as all targets in a line would have to move at the same rate to avoid collisions, or take on some other random behaviour.



Next I added the movement of the enemies, trying to make them shuffle like a plastic soldier might if they were being moved by hand. At this point the gun was static.

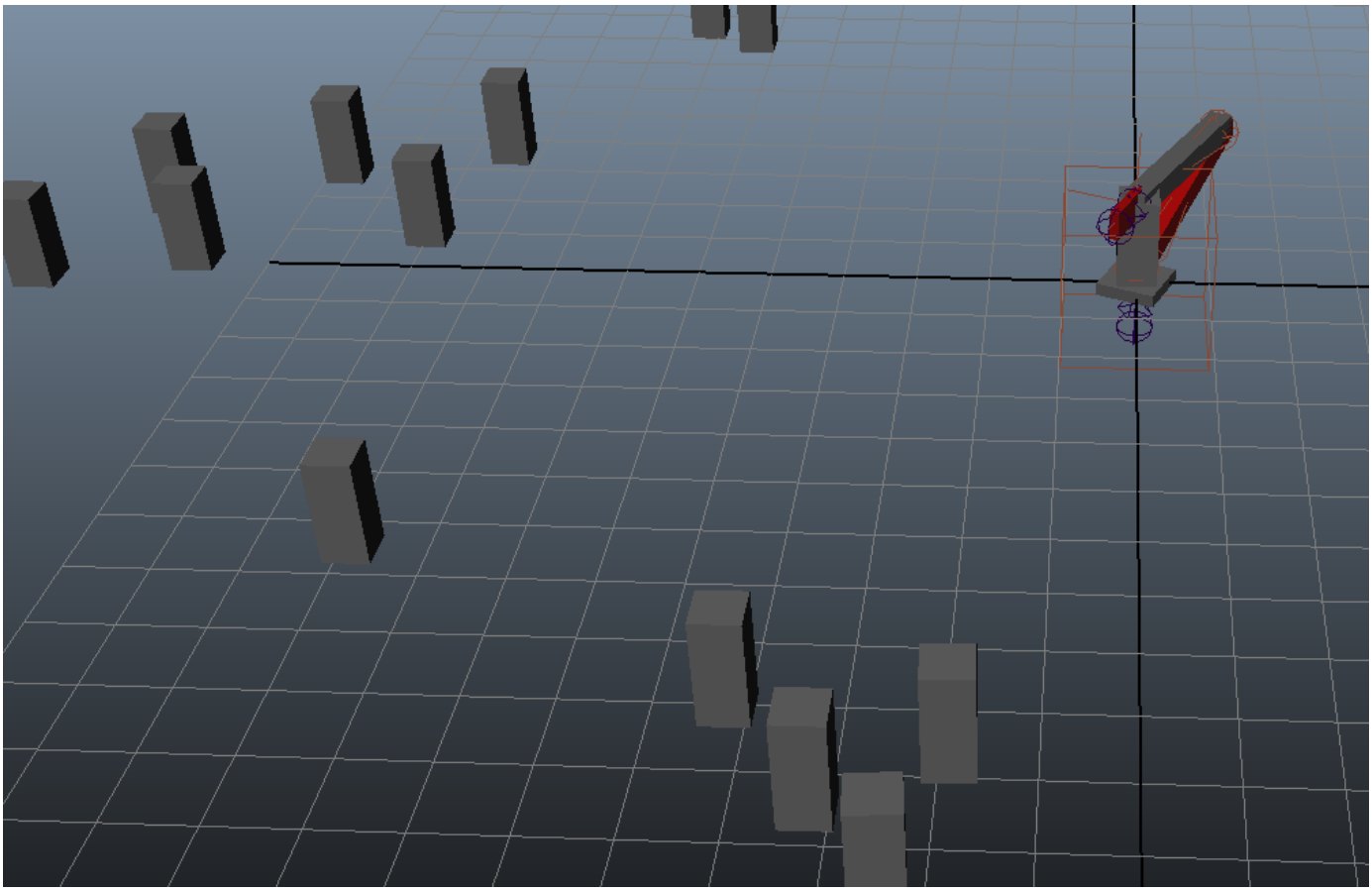


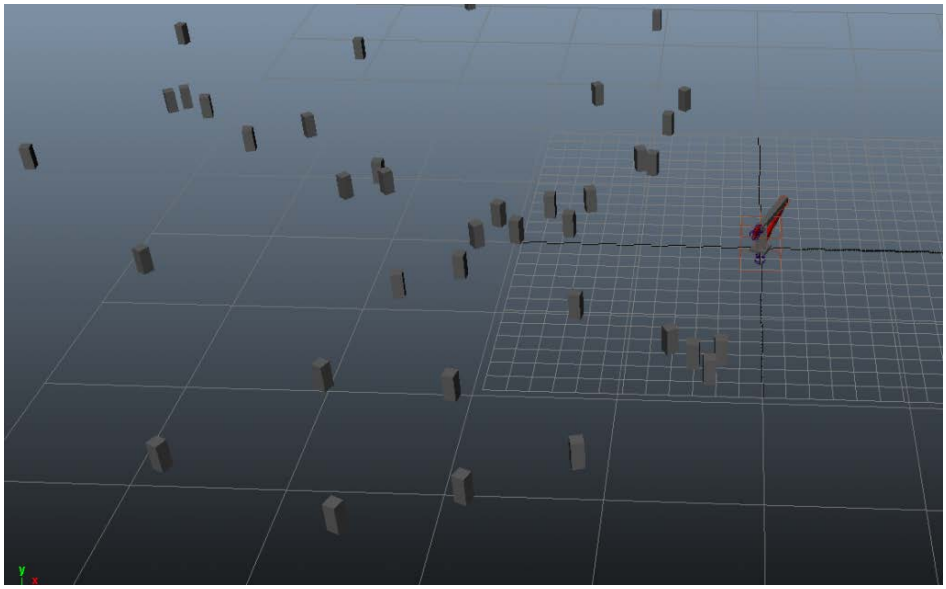
Playblast video of the above image: Target Movement Playblast (3.8MB).

In the video (linked just above), the GIs had a random wait cycle but all moved at the same rate. After this I added a variation in speed so that some targets would always be fast, and some would always be slow.

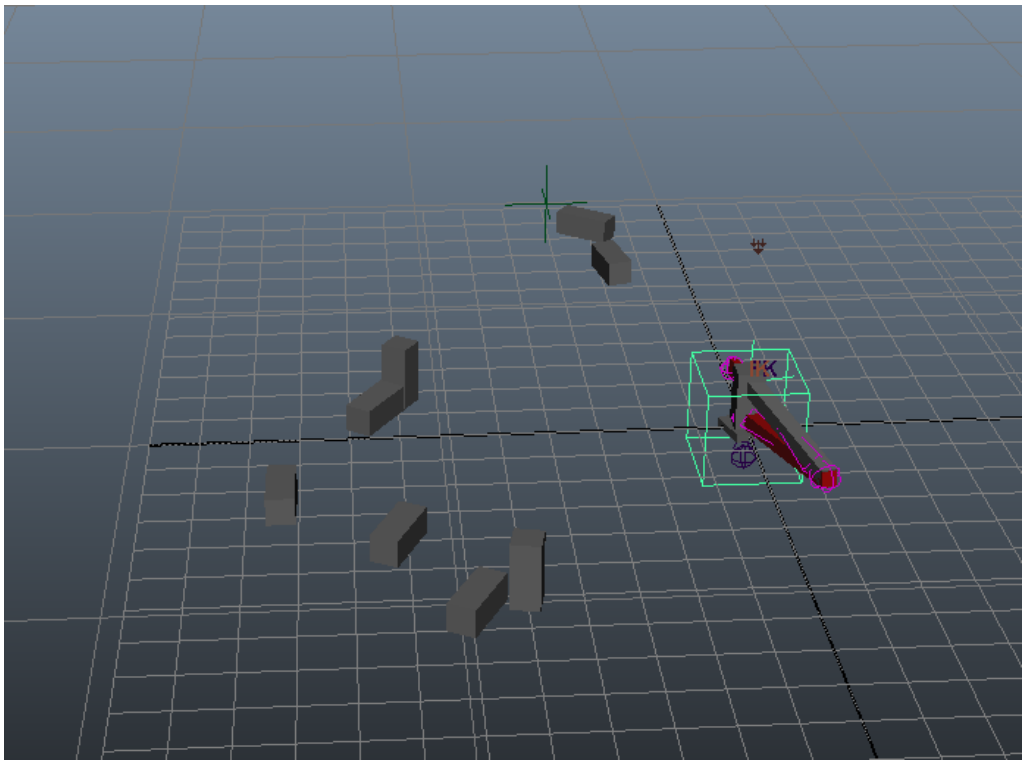
Robot Brain

To complete the scripted side of this project I next had to program the robot arm to aim and “shoot” at targets. I added targeting for the robot:

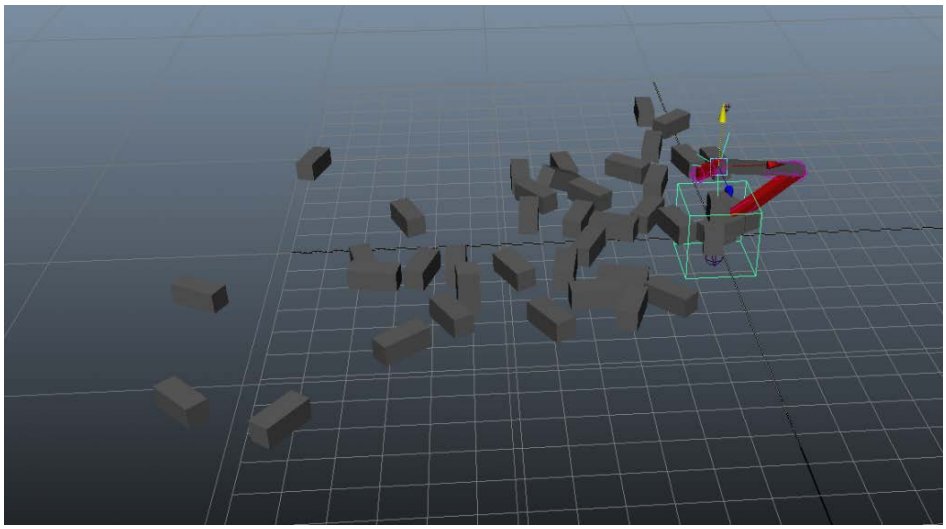




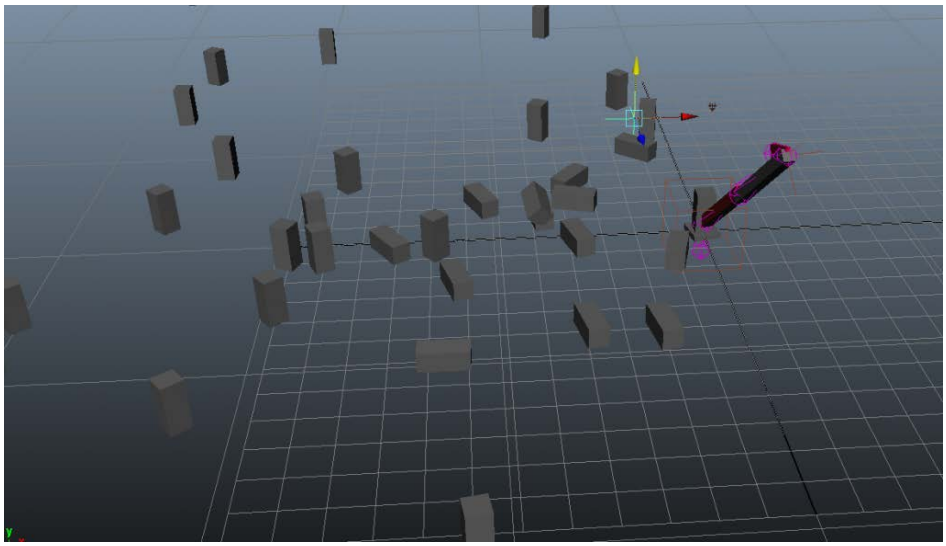
And to simplif. again, I used rigid body simulation in the die function of the GIs:



Using rigid body simulations meant I could tell an object to die and immediately delete the reference to it – it would do it.s own thing from that point. After the script ran, each GI had two objects that swapped visibility at the time of death: the original animated version and a physics version. One problem I ran into that I didn.t get time to revisit was caching/keyframe-converting the resulting transforms from the dying simulation. In the final video the targets jump around after being hit because the simulation (with an initial rotation force set on the dying objects) starts computing from frame one rather than the frame where a target dies. When the script is running however, the dying simulations look perfect.

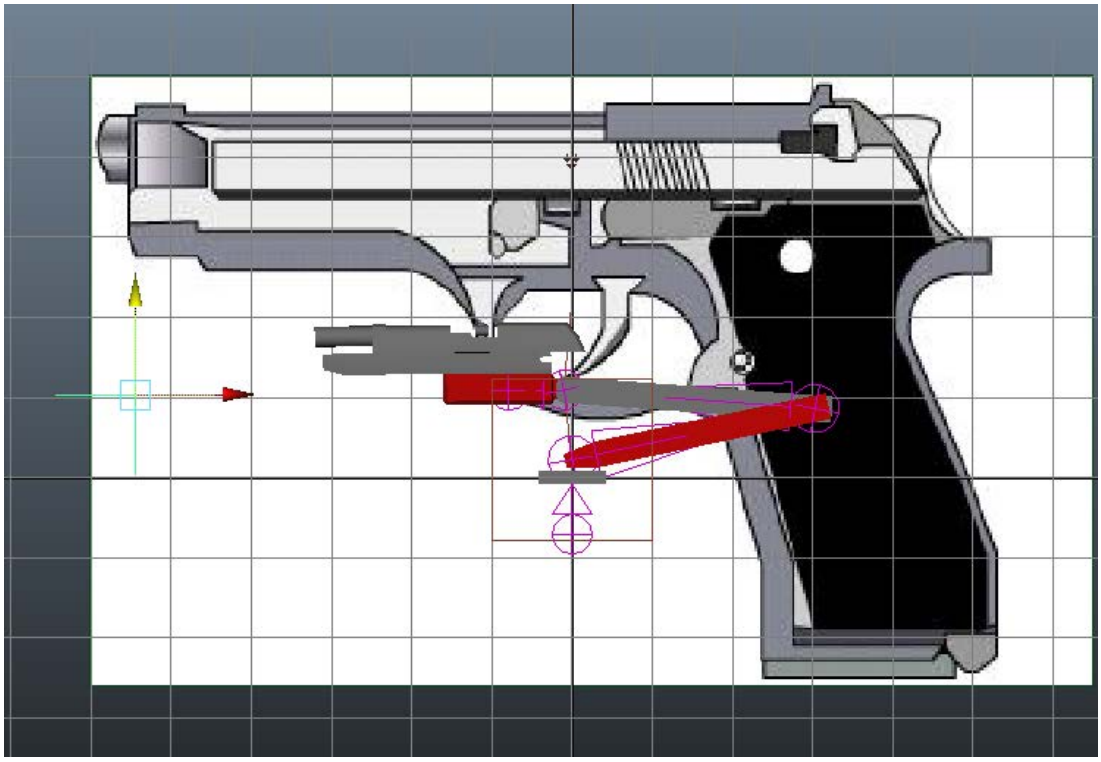


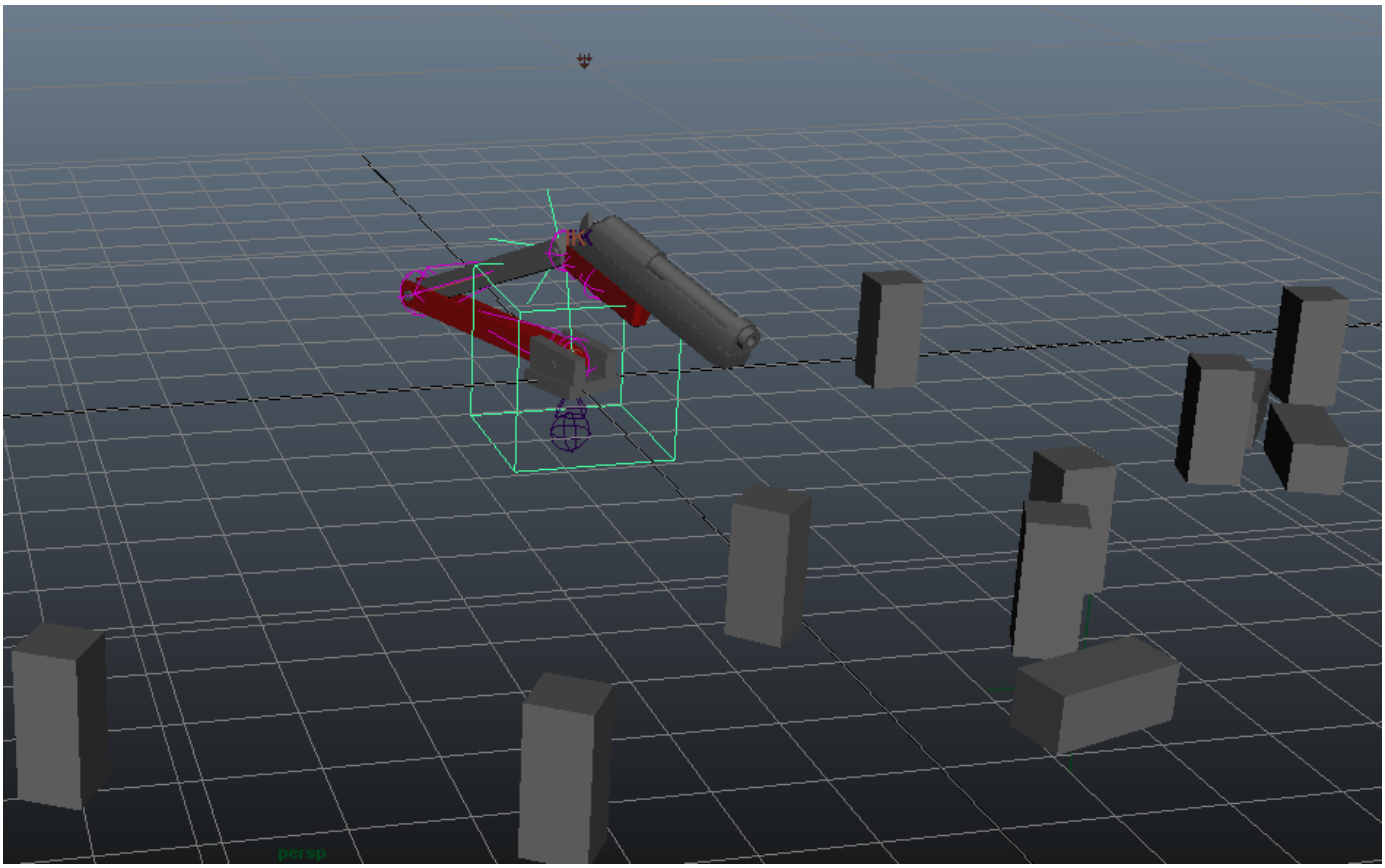
Next I added recoil when the robot fired. This was just a simple operation of relatively moving the rig's IK handle whenever the robot fired, and making the robot IK want to return to it's default position every frame it wasn't firing. It took a little bit to get the recoil and recovery rates right:

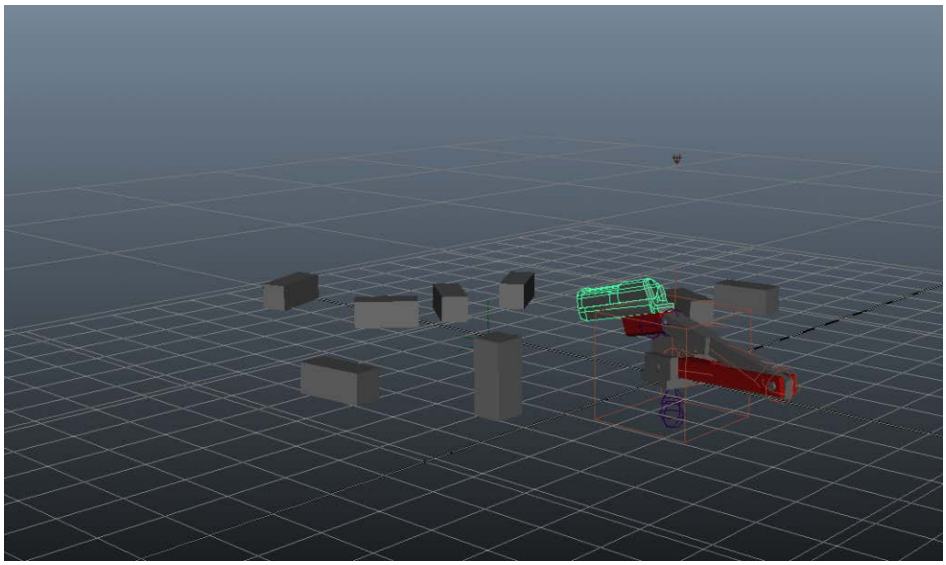


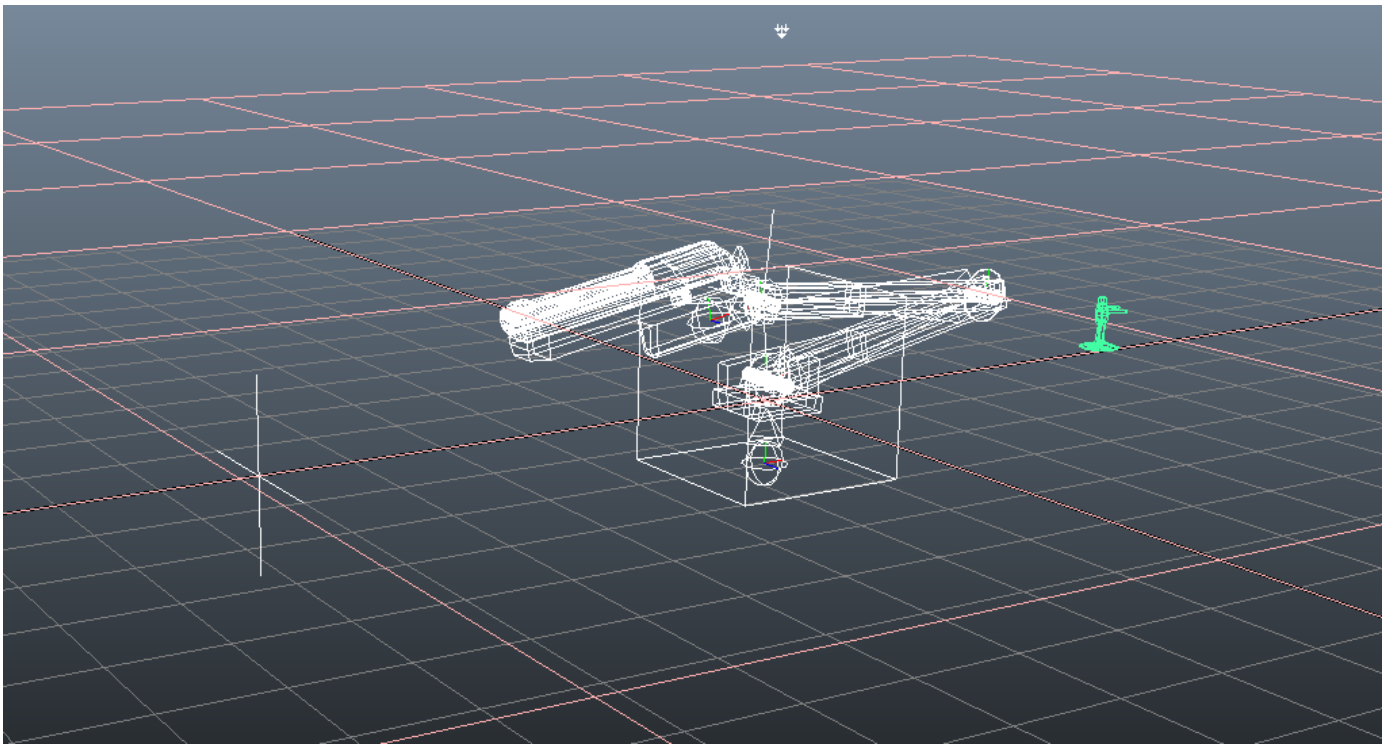
Models

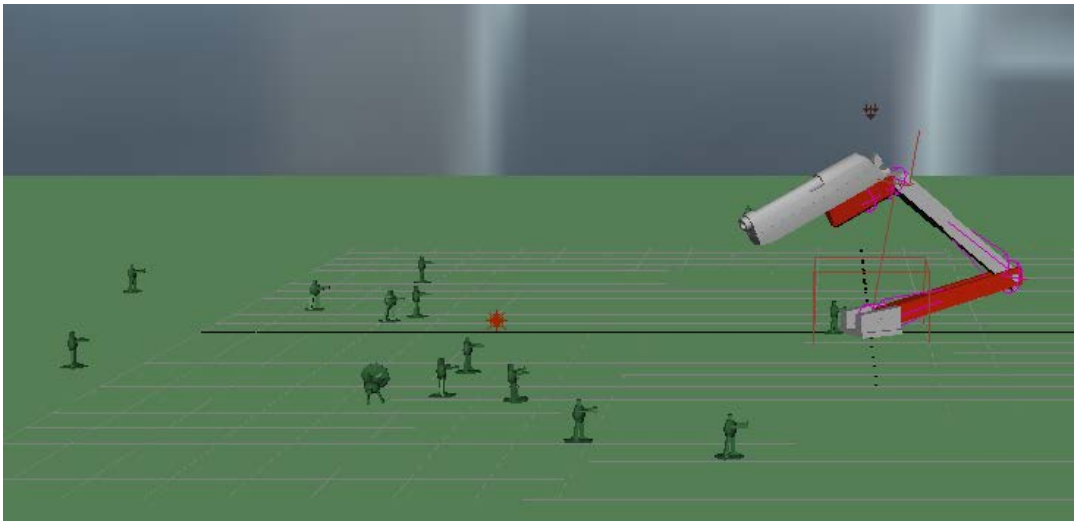
Lastly I got round to replacing the blocks with actual models, though I didn.t have a huge amount of time left!

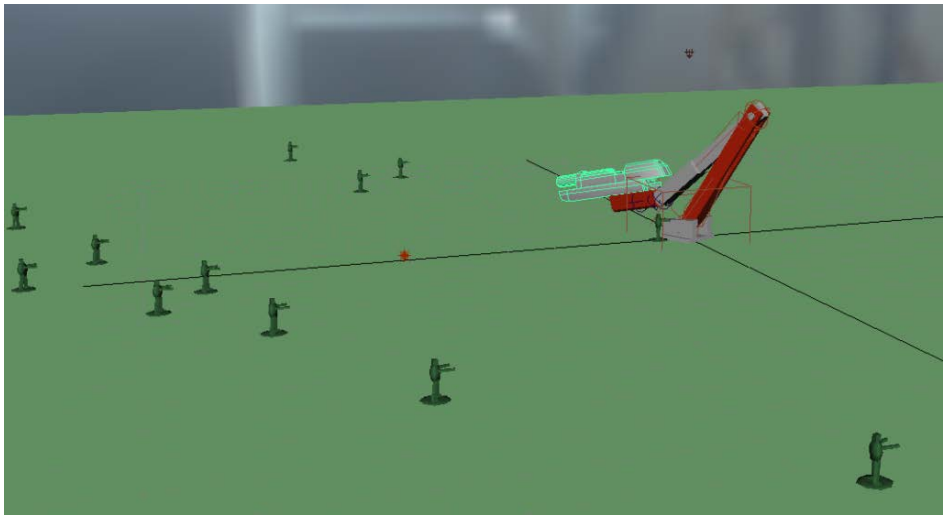


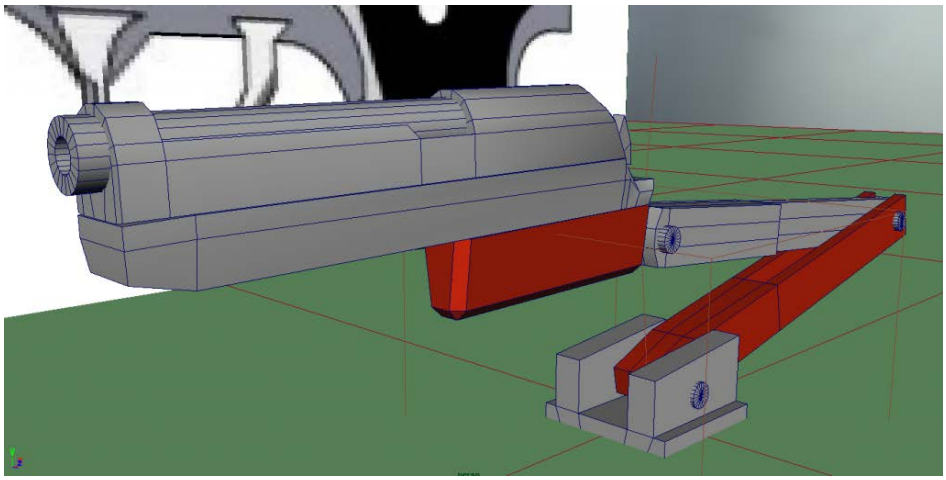

persp

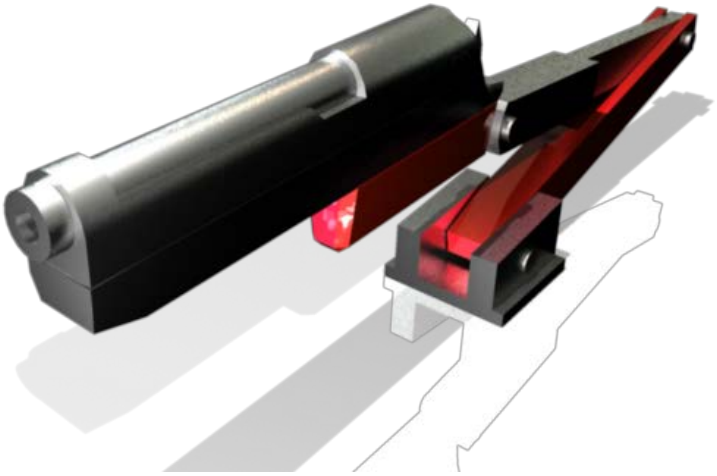


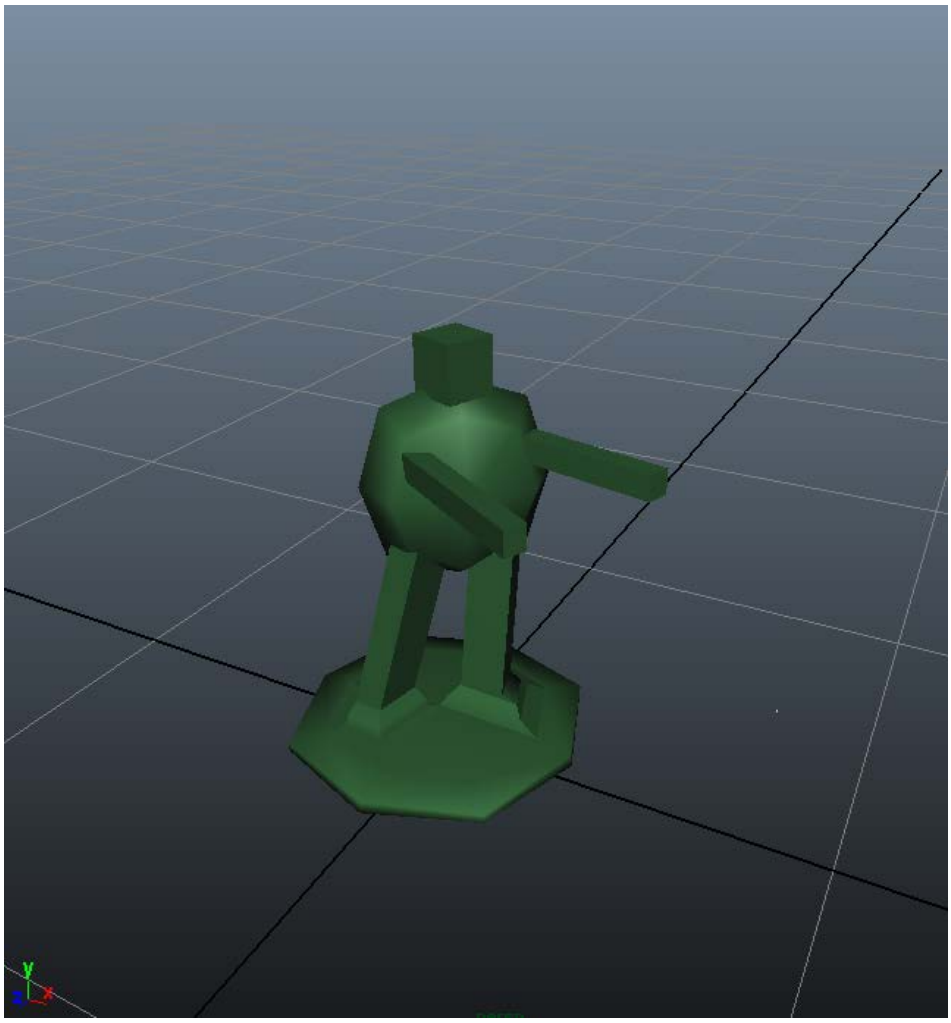






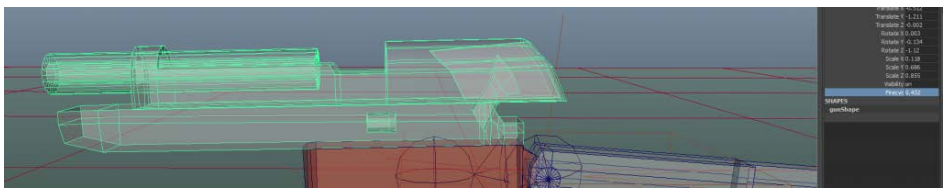






I would.ve really liked to make a decent looking plastic soldier, but this fella gets the job done ok.

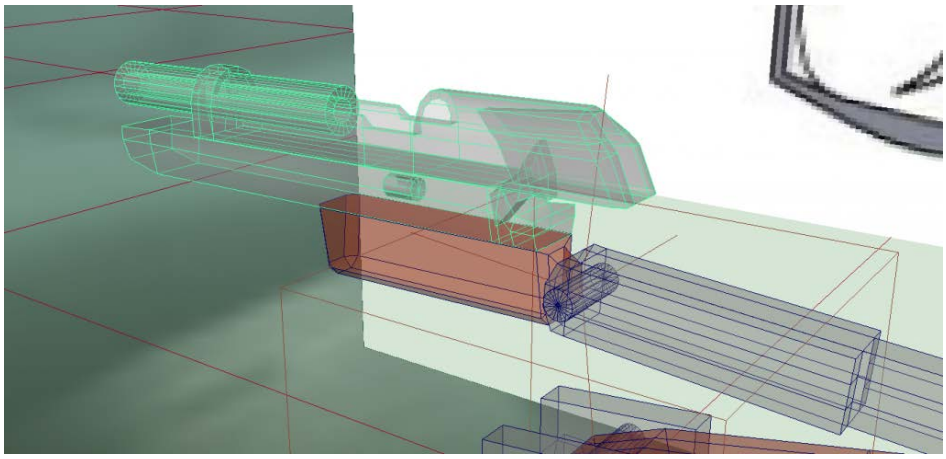
Random end note for this section – you can see the slide cycling in the video when the gun fires. This action would have been much better with cartridge ejection, but I'll get to that next. Rather than animating the whole thing over and over in code, The gun object has a custom float (range 0-1) property called "fireCycle". This property is the driver in a driven key that controls the slide and hammer position, and is keyframed in the script to get the animation.



Playblast of firecyc property 0. to 1. over 300 frames: firecyc_demo (1.2MB)

What was left out

I began working on two things that ended up getting left out because of time and bugs: particle muzzle flash and cartridge ejection from the gun. I almost had cartridge ejection nailed, but it unexpected caused a massive bug that I couldn't trace and meant the code wouldn't run at all unless I deleted duplicates of the cartridge reference (probably a duplicate name issue, but I didn't have time to debug it and it was affected by the same physics starting at frame zero problem I had with the targets dying). The cartridge ejection was completely physics based, and it looked pretty darn awesome during generation the one and only time I managed to get it to run. You can see the cartridge reference model hiding inside the gun here:



I started experimenting with particles for muzzle flash, but ran out of time as it caused Maya to crash frequently. I didn.t get any screenshots of this, but it was pretty basic – just a directional particle emitter in the end of the barrel which would have its generation rate hooked into the firecyc property that controls the slide and hammer action.

Last notes

I really enjoyed this project and had a good bit of fun. While it wasn't central to this project, I do wish I had spent more time on the aesthetic and render quality. The models weren't great (though they were functional), and the render quality really suffered because of time (had to significantly lower the render settings). In the near future I intend to add the missing muzzle flash and cartridge ejection, spice up the models a bit more, render the whole thing out nicely (with more enemies and shots), and compose it into a nice portfolio piece.