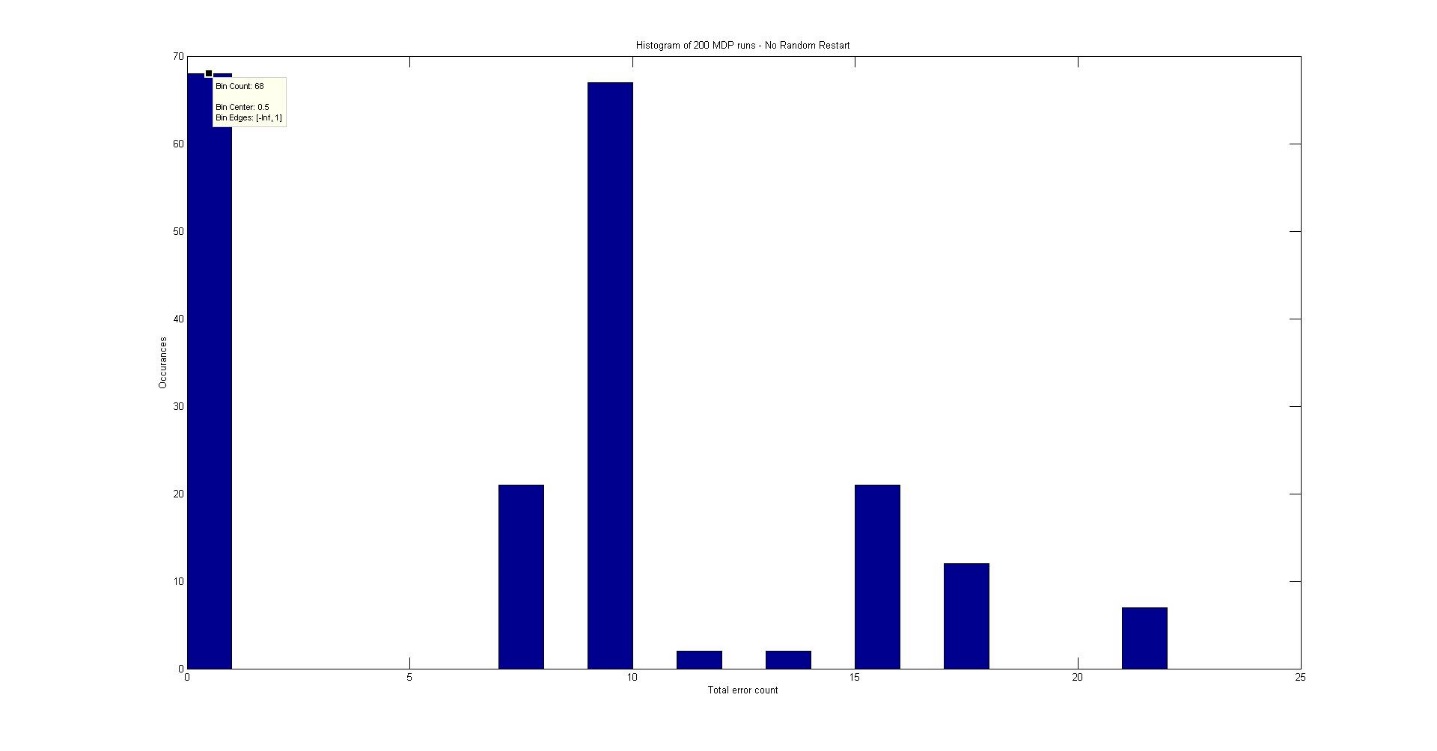
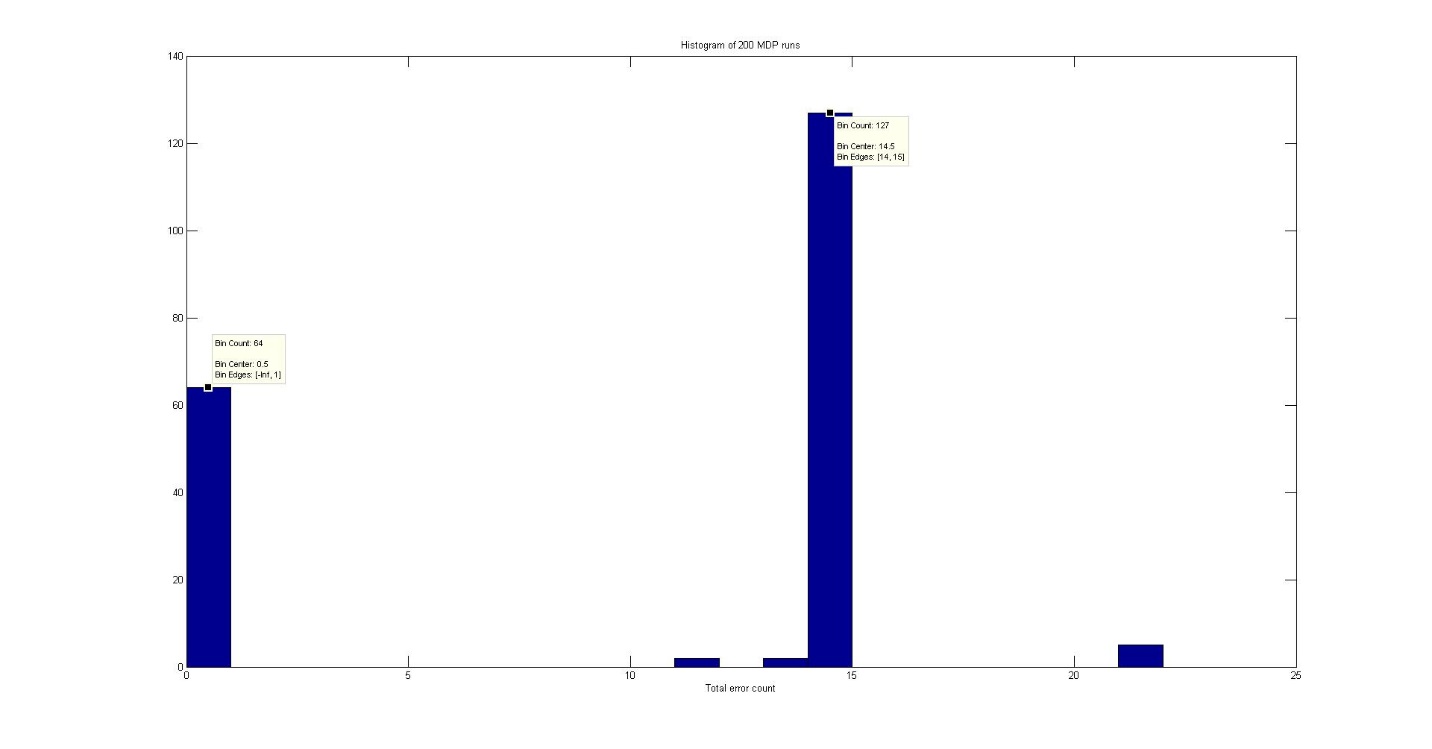
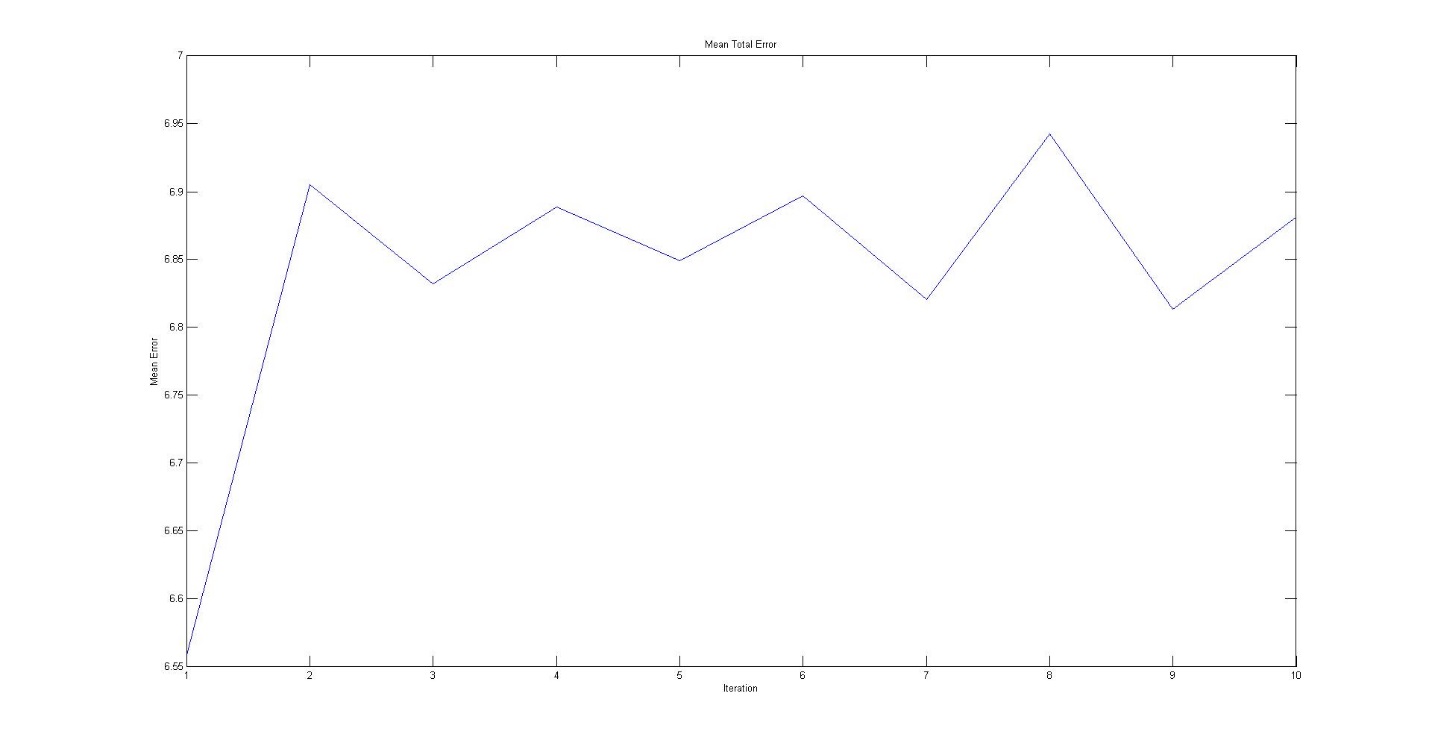
**26 May to 30 May**

The problem with the hashing function turned out to be an intermittent floating point error that would cause the results of a multiplication to occasionally round down. I added a *ceil()*  and there were no more problems (I’m mentioning this partially for my own reference in case I run into a similar problem again).

I executed this policy in matlab with random initial assignments (disregarding location). The agents would terminate after either a state with no errors was reached or 1000 cycles. I recorded the total errors (so far, still having a perfect assignment possible) over 200 trials.w

I noticed from a few trial runs where I watched the evolving error that the policy would often times get stuck and the error would remain constant for the rest of the run, so I implemented a round of random retargeting if the error was constant for 10 cycles. This didn’t end up helping, but it is worth looking at again later if we end up using MDPs. One possible improvement to this would be random retargeting only for agents that are in large coalitions (a sort of hybrid of clustering and MDP approaches). Again, this is something to consider much further down the road. The large peak at 15 errors is due to a problem with the random retargeting that would generate disallowed states and thus would select target NaN. I decided that, at this time, it wasn’t worth the time to track this problem down.

I decided I wanted a more fundamental way of evaluating the policies. I wrote a program to generate 10,000 allowable initial states, score each one and for each one implement the policy. Assuming no other agents changed targets in the interim, this was repeated 9 more times for each initial state. Plotting the mean error across all 10,000 states at each iteration, we would expect a trend of improvement with each iteration. That is not what we see though.

I went through the policy by hand next. Of the five states with no errors, the policy says to not change targets for only one. This suggests that there is a problem with the policy generator. I rewrote several chunks of this code and now it runs orders of magnitude slower but it finds a solution in one iteration.

I also spent a good deal of time talking with Dr. Brink about how the problem is framed and this is what we came up with. (I believe he mentioned you guys had discussed this earlier this week)  
Previously just had total weapons needed per target and attempted to have agents self-select targets based on that one number. This framework would replace that.

Targets are assigned to tiers

* Tiers are sorted by priority
* Tiers delineated by required Pk for that tier
* Begin filling lower tiers when above tier is satisfied or unreachable

Many factors effect Pk

* Effectiveness of weapons (Pki)
* Likelihood of weapon reaching target (1-Ai)
* Combination of weapons (possibly synchronicity)
* Make decisions based on effective change in Pk for each target and target priority
* Example:
  + One weapon has Pk of 0.80 and attrition of 0.25, effective Pk = 0.60
  + Second weapon has Pk of 0.95 and attrition of 0.50, effective Pk = 0.475
  + Overall Pk on target is 0.79

Attrition depends on many factors

* Depends on type of weapon
* Depends on time in air
* Depends on path/defenses
  + Separate process for path planning?
* During engagement can update attrition model and re-plan

On the Clustering side of the problem, added a cutoff to ignore targets which would require a change in heading of more than a set amount (currently 30o). Didn’t have a chance to run experiments, as most time was spent on MDPs and the Pk framework.