My strategy...

Rosie Campbell

This shows the pay-offs for each strategy:

P1 \ P2	Hunt	Slack
Hunt	0,0	-3,1
Slack	1,-3	-2,-2

Clearly, Slack-slack is the dominant strategy and also Nash Equilibrium, but since this is a repeated game it is not so simple. I have calculated that the discount factor for the game is a third, meaning that as long as there is at least 1/3 chance that the game will continue, it is worth cooperating.

Obviously, the best outcome for me is when I slack and my opponent hunts. If a player has a very very low reputation, chances are they will slack, in which case there is no point hunting with them. And if a player has a very very high reputation, chances are they will hunt - I can then take advantage of this by slacking.

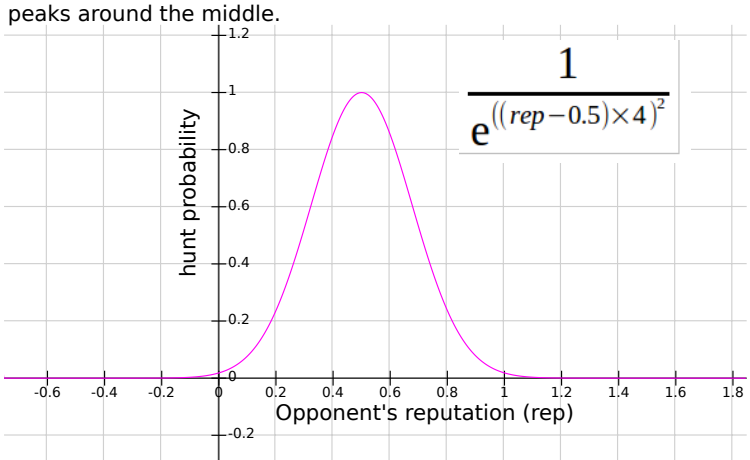
The basis of my algorithm, then, is to slack for low and high reputations and hunt somewhere in the middle. I implement this in a couple of ways:

Firstly and most simply, I check the reputation of my opponent and if it is less than 0.2 or greater than 0.9 I will definitely slack. This is not shown in the following graphs, but it is worth noting that for opponent reputations under 0.2 and over 0.9, the probability I will hunt is 0. This hard rule of always slacking at extremes only comes into play after 10 rounds, to allow the players to build up a representative reputation.

To decide what to do on the values in between, I use weighted randomisation.

I define a variable called hunt_probability, which is the probability I will hunt. I calculate this based on a number of factors, which I shall describe below. I then pass hunt_probability and 1-hunt_probability to a function which will return 0 or 1 with each probability respectively. If I get a 0, I hunt, if I get a 1, I slack.

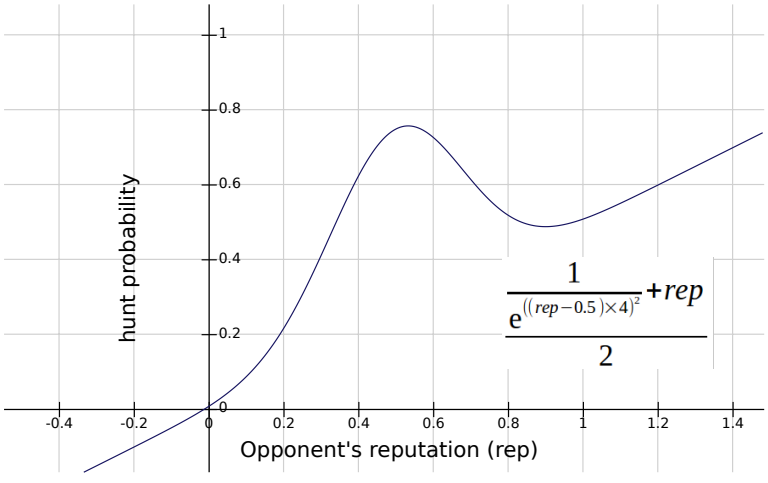
The main way I caluculate hunt_probability is to use a guassian probability distribution based on the players reputation. At very low and very high reputations, hunt_probability is very small. hunt_probability peaks around the middle.



The factor of 0.5 simply shifts the distribution so that it centers around 0.5, and the factor of 4 squeezes it in so that it starts at 0 and finishes at 1.

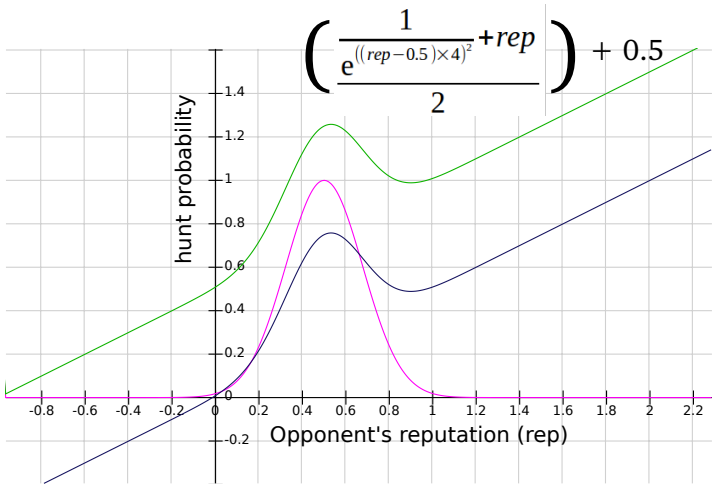
I was not happy with this being perfectly symmetric, as I would also like to punish slackers and reward hunters. I therefore decided to skew the gaussian distribution based on the reputation of the opponent.

To do this, I added the reputation of the opponent to the result of the above equation, and divided by 2. This produced a weighted probability distribution with the following shape:



Because this method is solely based on the opponents reputation, it is easy to end up with a low reputation if you play against others with a low reputation. Since most algorithms are likely to punish slackers, a low reputation means you risk getting starved out.

To combat this, I introduced a check on my own reputation - if it falls below 0.5 I aggresively ramp up my hunt_probability by adding 0.5 minus my current reputation to the previous calculations (capping it 1 obviously). This ensures I can still build up a good reputation even if I play against others with low reputations, so should prevent me being starved out. The green line below shows the maximum this can be (i.e. when my reputation is 0) and the corresponding formula:



This graph also shows the comparison between the other probability distributions used. To play around with them see http://fooplot.com/plot/fexxqzxw4f

Finally, I take into account the value of m. If m is relatively low, it is worth increasing my hunt_probability to try and reach it as it doesn't require lots of other players to cooperate. However if m is high, it relies on lots and lots of players hunting, which may be unlikely. To determine this threshold, each round I keep track of the proportion of hunts that have taken place throughout the game relative to the total number of interactions. If m is less than the average number of hunts per interations, I deem it worth going for! I then increase my hunt_probability by multiplying by an adjustment factor of 1.1.

Had I had more time, there are a number of factors I would have liked to take into account, such as the rate of player elimination, my food earnings, how my food was varying, and the values of public-good awards. On top of that I would like to have implemented a machine learning strategy which attempts to identify and track particular players based on the variations in their reputations. I also thought about keeping track of each interaction and it's outcome to try and learn the best response for different combinations of mine and their reputations. Sadly having a full-time job meant I couldn't spend as long on this as I'd have liked!!

Thanks to Chad Miller for creating the excellent test environment at:
https://github.com/ChadAMiller/hungergames