# Insights from Simulations and Phase Diagrams of Animal Conflicts

An Extended Analysis of John Maynard Smith and George Robert Price's "The Logic of Animal Conflict"

Christine Shi

ECON 4022: Evolutionary Game Theory

Professor Lawrence Blume

29 May 2022

## Introduction

My final project is based on “The Logic of Animal Conflict” by John Maynard Smith and George Robert Price. First, I will summarize Maynard Smith and Price’s paper: their purpose, their model for animal conflict, and their analyses. Then, I will break down how and why they obtained their results. Then, I will explain the simulation in Python I created to attempt to replicate their results and experiment with changing their model parameters. Throughout the process, I use phase diagrams made in R to illustrate the dynamics resulting from their and my findings.

## Summary of Maynard Smith and Price’s Paper

Maynard Smith and Price were interested in conflicts between animals of the same species. They observed that animals of the same type often intentionally do not cause serious injury during conflicts. For example, in many species of snakes, the males do not use their fangs when they wrestle with each other. In mule deer, the males fight with their antlers, but intentionally do not attack when their opponent turns away, exposing the unprotected side of its body (you can see in this YouTube video of a fight between two mule deer bucks that they only ever fight with their antlers until one of the bucks retreats). They called these fighting strategies that do not inflict serious injury "limited war" strategies.

Photo by Ray F. on Flickr. Neither buck was injured during the battle.

Up to the time of writing their paper, people in the field had accepted the wide deployment of “limited war” strategies as being due to group selection, since if members of species did not adopt these “limited war” strategies, many members of the species would be injured, and this would be disadvantageous for the species as a whole. However, Maynard Smith and Price were skeptical of this explanation, since group selection shouldn’t play a big enough role to by itself account for the many and complex adaptations for “limited war.” Instead, they used game theory and computer simulation to show that a "limited war" strategy actually benefits individual animals as well as the species.

To do this, they considered simple formal models of conflict situations (like “Hawk-Dove” but their own version), with some strategies being “limited war” and others being “total war,” and analyzed strategies for evolutionary stability. That is, they were trying to find out if the “limited war” strategies commonly observed within species were actually ESS’s.

**Defining Maynard Smith and Price's Model**

A conflict between two animals was modeled as a series of alternate "moves." At each move, the set of potential actions for each animal was {C, D, R}. C is for "conventional" tactics, which do not cause serious injury. D is for "dangerous" tactics, which have a fixed probability of inflicting serious injury. R is for retreat, which ends the conflict, and the contestant which has not retreated is the winner. A contestant that is seriously injured always retreats.

They considered 5 strategies: "Mouse," "Hawk," "Bully," "Retaliator," and "Prober-Retaliator."

> Mouse: Always plays C. If receives D, retreats before there is possibility of receiving a serious injury. Otherwise plays C until the contest has lasted a preassigned number of moves, then plays R.
>
> Hawk: Always plays D. Continues the contest until he is seriously injured and retreats or until his opponent retreats.
>
> Bully: Plays D if making the first move. Plays D in response to C. Plays C in response to D. Retreats if opponent plays D a second time.[1]
>
> Retaliator: Plays C if making the first move. If opponent plays C, plays C. If opponent plays D, with high probability plays D ("retaliates"). Plays R if contest has lasted a preassigned number of moves.
>
> Prober-Retaliator: If making the first move or if opponent has played C, plays C with high probability and plays D ("probes") with low probability. After giving probe, reverts to C if opponent retaliates, but "takes advantage" if opponent plays C by continuing to play D. If opponent plays D, with high probability plays D ("retaliates"). Plays R if contest has lasted preassigned number of moves.

They defined these parameters:

> Probability of serious injury to opponent from a single D play = 0.10
>
> Probability that prober-retaliator will probe on the opening move or after opponent has played C = 0.05
>
> Probability that Retaliator or Prober-Retaliator will retaliate against a probe (if not injured) by opponent = 1.0
>
> Pay-off for winning = +60

[1] I decided not to include the Bully strategy in my analysis to save myself time, since it does not seem to be very relevant to investigating the fitness of "limited war" strategies and is only mentioned about once in their paper.

Pay-off for receiving serious injury = -100

Pay-off for each D received that does not cause serious injury (a "scratch") = -2

Pay-off for saving time and energy (awarded to each contestant not seriously injured) = varied from 0 for a contest that reached maximum length, to +20 for a short contest

**Results of Maynard Smith and Price's Simulation**

They matched up each of the five strategies against itself and each of the other four strategies, resulting in fifteen types of matchings. They used a computer simulation to run 2,000 contests and found the average pay-off from the 2,000 contests for each of the fifteen matchings. Their results are shown below (minus the Bully row and column):

**Table 1: Average Pay-offs in Simulated Intraspecific Contests for Four Different Strategies (Maynard Smith & Price, 1973)**

| | Mouse | Hawk | Retaliator | Prober-Retaliator |
|---|---|---|---|---|
| Mouse | 29.0 | 19.5 | 29.0 | 17.2 |
| Hawk | 80.0 | -19.5 | -18.1 | -18.9 |
| Retaliator | 29.0 | -22.3 | 29.0 | 23.1 |
| Prober-Retaliator | 56.7 | -20.1 | 26.9 | 21.9 |

The number in a given row and column is the pay-off gained by the row strategy when the opponent uses the column strategy.

**Analysis of Maynard Smith and Price's Results**

Using these pay-offs, Maynard Smith and Price evaluated the evolutionary stability of each of their five strategies.

For a strategy to be an ESS, it means that it must be the most advantageous strategy played against itself compared to all other strategies. If it does equally well as any other strategy against itself, then it must do better against any other strategy than that strategy does against itself.

That is, for *s* to be an ESS, for all other strategies *s'*, $u(s, s) > u(s', s)$, or if $u(s, s) = u(s', s)$, $u(s, s') > u(s', s')$.

Mouse is not an ESS since all other strategies do the same or better than Mouse against Mouse.

Hawk is also not an ESS because Mouse does better against Hawk than Hawk does against Hawk. This makes sense because when more of the population are playing Hawks, there will be

more likely to be costs from serious injury, and strategies favoring less fighting will become more profitable.

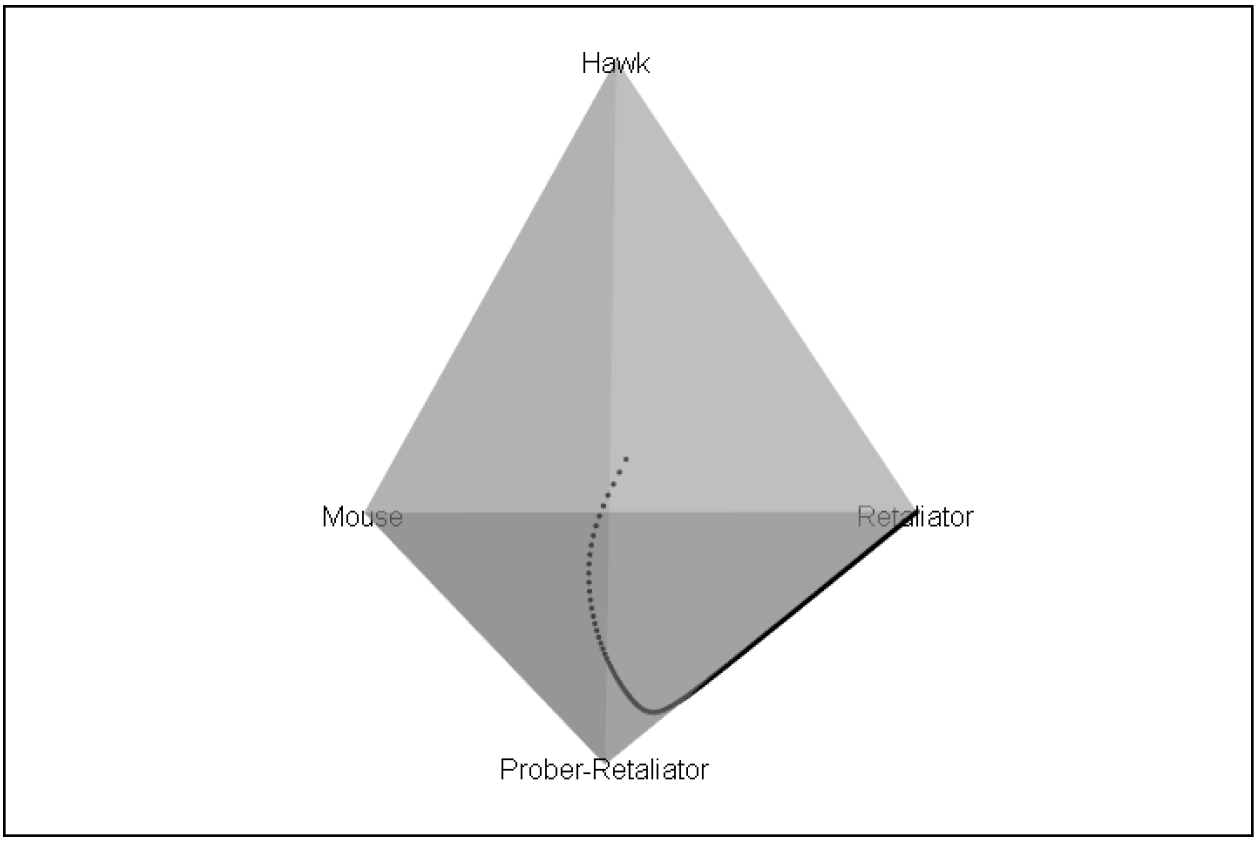
Retaliator is almost an ESS (it is a NSS). Retaliator does just as well as Mouse against Retaliator, and better than Hawk and Prober-Retaliator against Retaliator. Checking the second condition, it fails because retaliator does the same against Mouse as Mouse (29 = 29), worse than Hawk against Hawk (-22.3 < 19.5), and better than Prober-Retaliator against Prober-Retaliator (23.1 > 21.9). However, Maynard Smith and Price claim that Retaliator is an ESS. They probably are considering weak inequalities (NSS's) to be sufficient.

Prober-Retaliator is not an ESS, but Maynard Smith and Price say that it is almost an ESS because the payoff of Prober-Retaliator against Prober-Retaliator is almost as high as Retaliator vs. Prober-Retaliator.

Retaliator and Prober-Retaliator, which are "limited war" strategies, perform better than the "total war" strategy Hawk. This supports Maynard Smith and Price's hypothesis that there is a motivation from individual fitness to employ "limited war" strategies.

Using the EvolutionaryGames package (v0.1.1; Gebele & Staudacher, 2022) in R (v4.1.1; R Core Team, 2021), I created a phase diagram of the replicator dynamics in a single population with the four strategies, showing how the proportions of each strategy changed over time. I used the payoffs from Table 1 and set the initial population state to (0.25, 0.25, 0.25, 0.25):

**Figure 1: Mouse-Hawk-Retaliator-ProberRetaliator Replicator Dynamics**
Payoffs from MS & P (Table 1), $\sigma_0 = (0.25, 0.25, 0.25, 0.25)$




This shows that in a population with equal weight on each strategy, the proportion of Hawks immediately decreases due to its low fitness against all strategies but Mouse. The proportion of Mouse increases initially (maybe due to the decrease in Hawks and the moderate fitness of Mouse) then decreases. As the population of Mice increases, the population becomes increasingly dominated by Prober-Retaliators, as it does very well against Mouse and only slightly less well than other strategies against the other strategies. However, as the population of Hawks and Mice and Retaliators approaches 0, Retaliators begin to dominate, as it is the best performer against Prober-Retaliators. The phase diagram shows that there is a stable point where the population is completely made up of Retaliators.

**Deeper Analysis of Maynard Smith and Price's Model and Simulation**

Next, I tried to analyze Maynard Smith and Price's table of average simulated pay-offs of their four strategies (Table 1) and make sense of how/why Maynard Smith and Price found the average payoffs that they did.

First, I noticed there were some similarities between Maynard Smith and Price's model and the Hawk-Dove model.

In the Hawk-Dove model, there is a pay-off for winning a contest $v$, a cost of injury from fighting $c$, and rules about how the winning pay-off and cost of injury get split when different

pairings of strategies come together. When two Hawks meet, they split both the winning pay-off and the cost incurred from fighting. When a Hawk encounters a Dove, the Hawk makes off with the winner's pay-off and the Dove retreats and gets nothing (but does not receive serious injury). When two Doves meet, they split the winning pay-off without incurring fighting costs. The familiar Hawk-Dove game's pay-off matrix is shown below:

**Table 2: Traditional Hawk-Dove Pay-off Matrix**

| | Hawk | Dove |
|---|---|---|
| Hawk | $\frac{v-c}{2}$ | $v$ |
| Dove | 0 | $\frac{v}{2}$ |

Maynard Smith and Price's "Hawk" and "Mouse" are similar to the "Hawk" and "Dove" from Hawk-Dove. Their model also has a pay-off for winning ($v$), which is 60, and a cost of injury from fighting ($c$), which is 100. Unlike Hawk-Dove, however, their model is based on conflicts that are dynamic/sequential rather than a set outcome that always happens. In addition to Hawk-Dove, they also have a cost from a "scratch" (if one receives a D play but is not seriously injured) and a pay-off from saving time and energy if a contest ends earlier. They also define three other strategies that fall between the always-aggressive Hawk and the always-passive Dove: "Bully," "Retaliator," and "Prober-Retaliator."

First, for simplicity, I decided to just consider what would happen if we added a "Retaliator" strategy to the traditional Hawk-Dove game. If a Retaliator meets a Hawk, they would essentially behave like a Hawk (since they play D with high probability if their opponent plays D). If a Retaliator meets a Dove or a Retaliator, they would essentially behave like a Dove (since the Retaliator always plays C unless probed). The pay-offs for the Retaliator, then, would be the Hawk-Hawk pay-off ($\frac{v-c}{2}$) if matched with a Hawk, and the Dove-Dove pay-off ($\frac{v}{2}$) if matched with a Dove or Retaliator. The resulting payoff matrix is shown below:

**Table 3: Traditional Hawk-Dove-Retaliator Pay-off Matrix**

| | Hawk | Dove | Retaliator |
|---|---|---|---|
| Hawk | $\frac{v-c}{2}$ | $v$ | $\frac{v-c}{2}$ |
| Dove | 0 | $\frac{v}{2}$ | $\frac{v}{2}$ |
| Retaliator | $\frac{v-c}{2}$ | $\frac{v}{2}$ | $\frac{v}{2}$ |

How does this compare to Maynard Smith and Price's game? First, we plug in their values for *v* (60) and *c* (100):

**Table 4: Hawk-Dove-Retaliator Pay-offs with *v* = 60 and *c* = 100**

| | Hawk | Dove | Retaliator |
|---|---|---|---|
| Hawk | $\frac{v-c}{2} = \frac{60-100}{2}$ = -20 | $v = 60$ | $\frac{v-c}{2}$ = -20 |
| Dove | 0 | $\frac{v}{2} = \frac{60}{2} = 30$ | $\frac{v}{2} = 30$ |
| Retaliator | $\frac{v-c}{2}$ = -20 | $\frac{v}{2} = 30$ | $\frac{v}{2} = 30$ |

Now we can compare: Do the standard HDR pay-offs look similar to what MS & P obtained after their 2,000 computer simulations? Putting the two side-by-side (standard on the left, MS&P on the right):

**Table 5: Standard Hawk-Dove-Retaliator Pay-offs (Left) vs. Maynard Smith and Price's Average Simulated Pay-offs (Right)**

| | Hawk | | Dove/Mouse | | Retaliator | |
|---|---|---|---|---|---|---|
| Hawk | -20 | -19.5 | 60 | 80 | -20 | -18.1 |
| D/M | 0 | 19.5 | 30 | 29 | 30 | 29 |
| Retaliator | -20 | -22.3 | 30 | 29 | 30 | 29 |

Overall, we observe that with the exception of the Hawk-Dove and Dove-Hawk matchups, the simulated average pay-off is actually not far off from the pay-off projected from our Hawk-Dove-Retaliator model when using the same values of *v* and *c*. This seems to suggest that the standard HDR model does well at predicting what would happen when animals that play these strategies are actually interacting with each other.

First, we observe that the pay-off from a matchup between two passive types (Mouse and Dove), or between a passive type and the Retaliator, is $\frac{v}{2}$ = 30 in the standard game, and 29.0 in all the average simulated matches. In the standard game, this is because Doves split the territory with each other when they meet, and Retaliators act like Doves/Mice when they meet them. In the simulation, however, the contest is dynamic/sequential instead of an instantaneous decision to split; and the scenario is fighting over a female and not over territory, so there is no way to "split" the payoff. The payoff still ends up being around $\frac{v}{2}$ because in these matchups the game always runs to the end (when the pre-assigned number of moves is up), and the winner's payoff

is assigned to the last player standing. I am guessing that they randomized the order of who starts, so in about half of the contests, this will be in favor of the row player, and in the other half, the column player. In the absence of any payoff losses from injury or scratching, since they never play D against each other, this would make the average payoff around $\frac{v}{2}$.

Next, the pay-off from a matchup between two Hawks or a Hawk and a Retaliator is $\frac{v-c}{2}$ = -20 in the standard game, and between -18.1 and -22.3 in MS & P's simulation. In the standard game, the interpretation is that the Hawk wins (+$v$) with probability ½ and loses (-$c$) with probability ½, and the Retaliator behaves like the Hawk when meeting a Hawk. In the simulated game, around half of the time P1 deals the last D play that causes the other player to retreat, and their payoff is about 80 (60, the payoff from winning, plus the time-saved payoff, 20). The other half of the time, P1 is the loser and gets -100 from serious injury. This averages out to -10. The remaining ~-10 can be accounted for by scratches. The payoff for Retaliator-Hawk is likely slightly lower than for Hawk-Hawk because half the time, the Retaliator goes first and plays C, increasing the chance that it will be the first to suffer serious injury.

The ~20 point difference between MS & P's model and our HDR model can be attributed to the pay-off for saving time and energy that they added, which could go up to +20 for short contests. We would expect this 20 point difference, actually, because in Hawk-Mouse and Mouse-Hawk contests, the same thing always happens: the Hawk always plays D on its first move and the Mouse always retreats (before there is a possibility for serious injury) as soon as the Hawk plays D. This means that both the Mouse and the Hawk are awarded the maximum payoff for time saved.

I have found no good/definite explanation for the small discrepancies between the standard HDR and MS&P's results, especially since they were very vague about how they implemented the simulation. I am not sure if the discrepancy is due to a difference in the rules of MS & P's model, or in the way they implemented their model on the computer (perhaps there is a difference in how they calculated the payoff from saving time and energy).

## Reproducing Maynard Smith and Price's Simulation

My next goal was to replicate Maynard Smith and Price's computer simulations and experiment with changing the model parameters, seeing what would happen to the pay-offs.[2]

[2] Looking at the problem at first, I had decided it was too hard to try to write code to simulate an alternating-move game with each contestant's move for a round prescribed by certain probabilities and depending on what happened the previous round, especially given that they did not provide much insight into how they implemented their simulation. How would they even have done this in 1973?? Plus, I only took CS 2110 and that was like almost 2 years ago now. But! I was able to do it thanks to the help of an amazing friend. He didn't straight up tell me how

I wrote a program in Python (v3.9.7; Van Rossum & Drake, 2009) on the platform Jupyter Notebook to simulate the animal conflicts. My program consisted of a class for each strategy, each strategy having its own initialization method and "move" method, and a main method in which the simulations were run. The following skeleton details how I implemented the model:

**Figure 2: Simulation Code Skeleton**

1. Initialization
    a. Set model parameters:
        i. `p_i = 0.10` = probability of serious injury after a single D play
        ii. `p_p = 0.05` = probability that a Prober-Retaliator will probe on its first move or in response to C
        iii. `p_r = 1.00` = probability that a Retaliator or Prober-Retaliator will play D in response to D
        iv. `v = 60` = payoff from winning
        v. `c = 100` = cost of serious injury
        vi. `s = 2` = cost of a scratch
        vii. `u_init = 20` = initial/maximum payoff from saving time and energy (counts down every other move)
        viii. `u_minus = 0.5` = increment by which the payoff from saving time and energy is decreased every other move
    b. `total` and `total2`, counter variables to hold the payoff from all trials of the simulation, are initialized to 0.
2. Run the simulation `num_trials` times with a `for` loop
    a. P1 and P2 are initialized to their different types (Hawk, Mouse, Retaliator, or Prober-Retaliator)
    b. The payoff for saving time and energy, `u`, is initially set to `u_init`.
    c. Contest: `while` loop
        i. P1 moves
            1. If P1 is a Hawk: Always plays D.
            2. If P1 is a Mouse: Always plays C.
            3. If P1 is a Retaliator: Plays C if unprovoked, retaliates with probability `p_r`
            4. If P1 is a Prober-Retaliator: If unprovoked, probes with probability `p_p`, otherwise plays C. After probing, reverts to C if

---

to do it but helped me think through the process of turning the English version of the game into structured code, helping me get unstuck and learn some good coding practices in the meantime. Also, he helped me when I stumbled with Python syntax.

the opponent retaliates, keeps playing D if the opponent does not retaliate. Retaliates with probability `p_r`.

ii. Payoffs are checked and changed based on P1's move
    1. If P1 has played R (retreat), the game ends and P2's payoff increases by *v* (win)
    2. If P1 has played D:
        a. With probability *p_i*, P2's payoff decreases by *c* (serious injury)
        b. With probability 1 - *p_i*, P2's payoff decreases by *s* (scratch)

iii. P2 moves

iv. Payoffs are checked and changed based on P2's move

v. The time-saved payoff `u` is decreased by an increment `u_minus`.

d. Post-contest:
    i. The remaining `u` is added to the uninjured players.
    ii. Each player's payoffs are added to the total counter variables.

3. The average payoff is printed for each player (`total/num_trials` and `total2/num_trials`, or `(total+total2)/(2*num_trials)` for a same-strategy matchup (e.g. Hawk-Hawk).

Notes:

1. I didn't write code for the Bully, as it is mentioned the least often in their paper and does not seem very significant to me.
2. I am guessing that Maynard Smith and Price, in their simulations of each matchup, somehow randomized who takes the first move. Since my way of coding it set one player as the first move, for each matchup, I averaged the player's payoff when they were the first mover with their payoff when they were the second mover. For example, to get the average payoff to the Retaliator in the Retaliator-Hawk matchup, I averaged P1's average payoff from the Retaliator-Hawk matchup (in which the Retaliator moves first) and P2's average payoff from the Hawk-Retaliator matchup (in which the Retaliator moves second).

Here is an example of one of the simulated contests:

**Figure 3: Example Output of My Code (Hawk vs. Retaliator)**

```
p1: D

      p2: D
```

```
p1: D

      p2: D

p1: R

Hawk vs. Retaliator avg payoff: -102.0

   Retaliator's payoff: 75.0
```

Explanation: The Hawk starts, and always plays D. The Retaliator always plays D in response to D (`p_r` = 1). They scratch each other until P2's D play results in serious injury to P1, who retreats. The Hawk's payoff is -102 (-100 from serious injury, -2 from the scratch it received in the second move). The Retaliator's payoff is 75 (+60 from winning, -4 from the scratches it received on moves 1 and 3, and +19 from time saved).

**Results of My Simulation**

Using my code, I ran 100,000 simulations with each matchup, obtaining the following results:

**Table 6: Average Pay-offs in My Simulated Intraspecific Contests for Four Different Strategies**

| | Mouse | Hawk | Retaliator | Prober-Retaliator |
|---|---|---|---|---|
| Mouse | 30 | 19.75 | 30 | 15.3 |
| Hawk | 79.75 | -20.1 | -15.2 | -14.9 |
| Retaliator | 30 | -25.2 | 30 | 12.2 |
| Prober-Retaliator | 67.6 | -25.7 | 14.9 | 3.1 |

**Table 7: My Simulated Average Pay-offs (Left) vs. Maynard Smith and Price's Simulated Average Pay-offs (Right)**

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | 29 | 19.75 | 19.5 | 30 | 29 | 15.3 | 17.2 |
| Hawk | 79.75 | 80 | -20.1 | -19.5 | -15.2 | -18.1 | -14.9 | -18.9 |
| Retaliator | 30 | 29 | -25.2 | -22.3 | 30 | 29 | 12.2 | 23.1 |
| Prober-Retaliator | 67.6 | 56.7 | -25.7 | -20.1 | 14.9 | 26.9 | 3.1 | 21.9 |

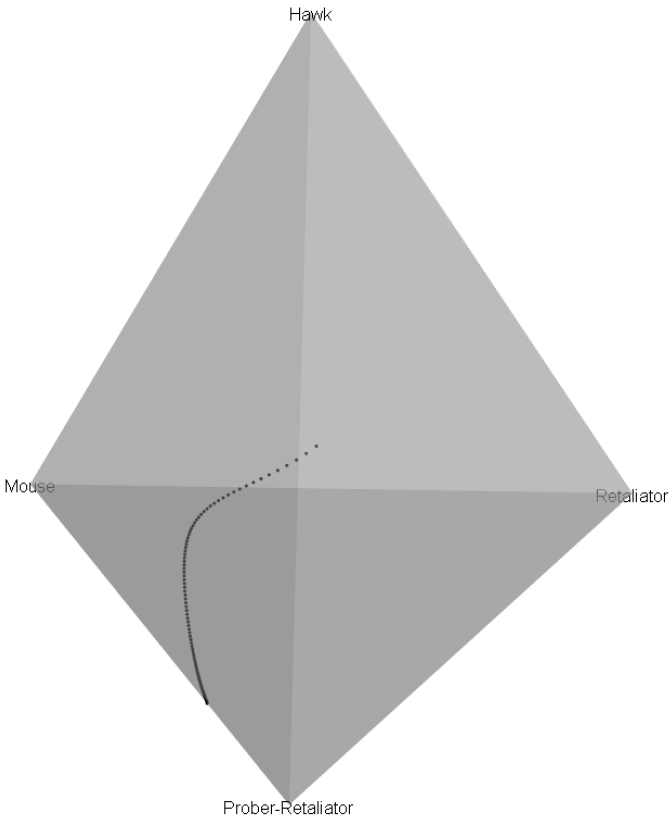
**Analysis of My Simulation Results**

Immediately, it looks like there are some differences between my results and Maynard Smith and Price's. As they did not provide many details about how they implemented this, I did my best and I have no way of knowing why my results differ from theirs. I suspect some of the differences are due to the ways we coded the "payoff from saving time and energy."

Based on my table of payoffs, the analysis of evolutionary stability is not too different: Retaliator is an NSS, just as they found, but Prober-Retaliator is no longer almost an ESS, as its fitness in my simulation is lower than it is in MS & P's.

This is what the phase diagram looks like:

**Figure 3: Mouse-Hawk-Retaliator-ProberRetaliator Replicator Dynamics**
Payoffs from me (Table 6), $\sigma_0 = (0.25, 0.25, 0.25, 0.25)$



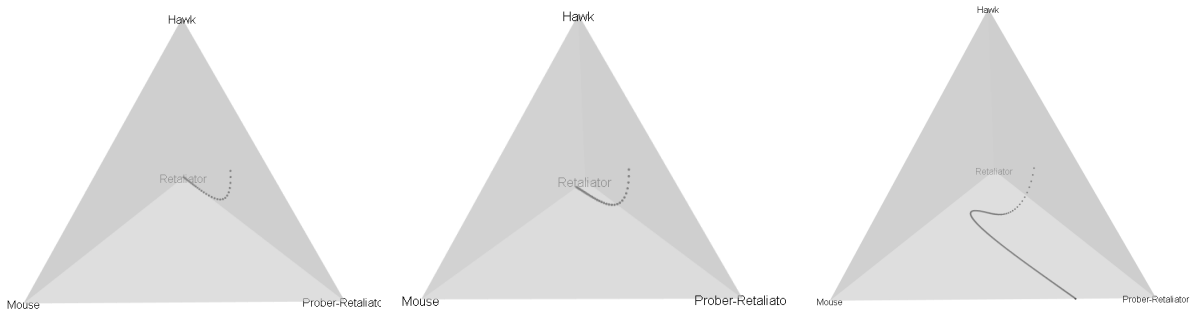

This is interesting, since it gives a different result from the phase diagram made with MS & P's payoffs. Even though the Prober-Retaliator strategy has decreased in fitness against all strategies except Mouse in my simulation, its increased fitness against Mouse causes the population to converge to a mix of around ⅔ Mice and ⅓ Prober-Retaliators.

It is likely that the presence of Prober-Retaliators in the ending steady state is due to the presence of Mice in the initial state. I confirmed this by experimenting with different values of $\sigma_0^{Mouse}$:

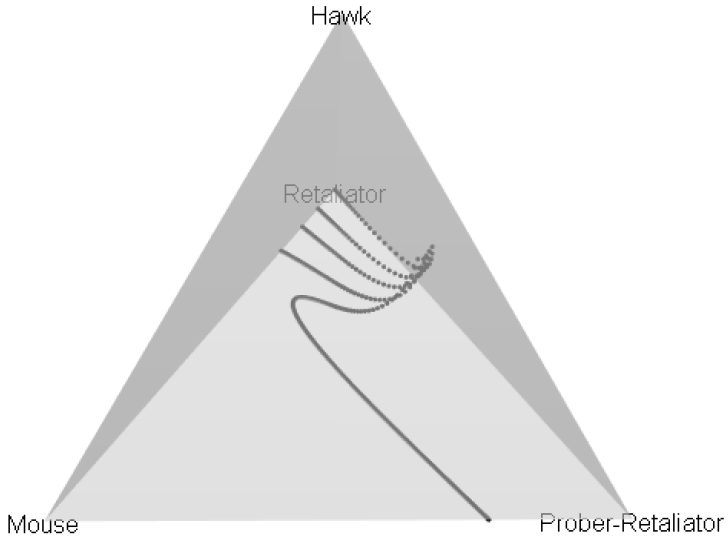
**Figure 4: Mouse-Hawk-Retaliator-ProberRetaliator Replicator Dynamics with Different Initial Population States**

Payoffs from me (Table 6)

$\sigma_0 = (0, 0.33, 0.33, 0.33)$ $\sigma_0 = (0.01, 0.33, 0.33, 0.33)$ $\sigma_0 = (0.04, 0.32, 0.32, 0.32)$




From right to left: $\sigma_0^{Mouse} = 0, 0.01, 0.02, 0.03, 0.04$:




This shows that for sufficiently small starting values of the proportion of Mice in the population, a population of all Retaliators is actually the steady state. As $\sigma_0^{Mouse}$ increases from 0.03 to 0.04, the presence of Mice in the initial population becomes large enough that Prober-Retaliators have enough of an advantage to turn the population onto the path to a Mouse-ProberRetaliator mix.

## Changing the Model Parameters

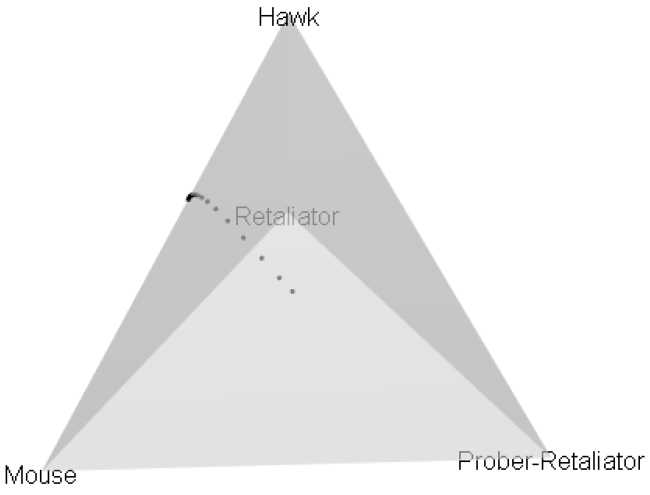
My final goal of investigation was to change the model parameters and see what happens. In each of the following sections, I change only one of the parameters at a time from the settings in Figure 2. I run the simulation with the parameter change (`num_trials = 100000`). I display a table that compares the new payoffs to the payoffs from my simulation with the original parameters. Then, I plot the new phase diagram after the parameter change. Finally, I interpret the results.

**Increasing the Probability of Serious Injury**

Changing `p_i` from 0.10 to 0.90:

Left: Payoffs from my simulation with original parameters
Right (bold): New payoffs after increasing the probability of serious injury
Significant changes are highlighted

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **30** | 19.75 | **19.75** | 30 | **30** | 15.3 | **15.4** |
| Hawk | 79.75 | **79.75** | -20.1 | **-10.1** | -15.2 | **63.4** | -14.9 | **59.8** |
| Retaliator | 30 | **30** | -25.2 | **-83.9** | 30 | **30** | 12.2 | **-69.5** |
| Prober-Retaliator | 67.6 | **67.6** | -25.7 | **-80.3** | 14.9 | **53.8** | 3.1 | **-11.7** |

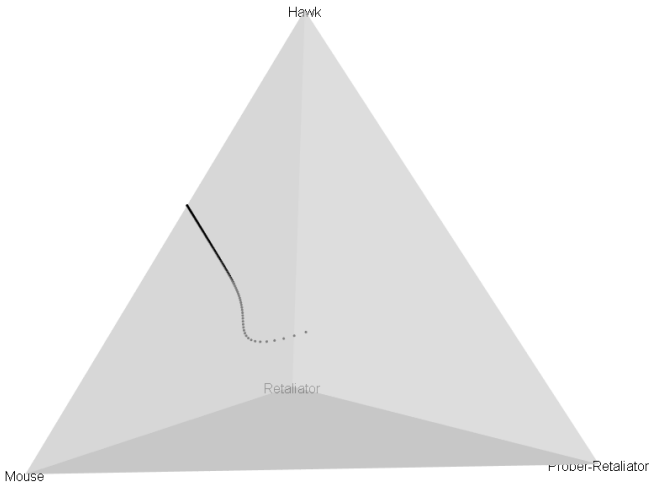



D plays being more likely to cause serious injury causes “preemptive strike” strategies to be more profitable, so Hawk becomes much more profitable. As the phase diagram shows, the population converges to a mix of Hawk and Mouse.

**Increasing the Probability of Probing Behavior**

Changing `p_p` from 0.05 to 0.50:

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **30** | 19.75 | **19.75** | 30 | **30** | 15.3 | **19.2** |
| Hawk | 79.75 | **79.75** | -20.1 | **-20.2** | -15.2 | **-14.8** | -14.9 | **-15.3** |
| Retaliator | 30 | **30** | -25.2 | **-25.6** | 30 | **30** | 12.2 | **-23.5** |
| Prober-Retaliator | 67.6 | **79.2** | -25.7 | **-25.1** | 14.9 | **-13.8** | 3.1 | **-20.4** |




Basically, Prober-Retaliator does better against Mouse and worse against Retaliator and Prober-Retaliator. This is basically because the Prober-Retaliator becomes more and more similar to the Hawk when the probability of probing is turned up. However, it still does not do as well as Hawk against Mouse, so the population still converges to a mix of Hawk and Mouse. Compared to the previous example, Retaliator decreases to 0 faster, perhaps because as Prober-Retaliators become more aggressive, Retaliator is less profitable and Mouse is more profitable against Prober-Retaliator.

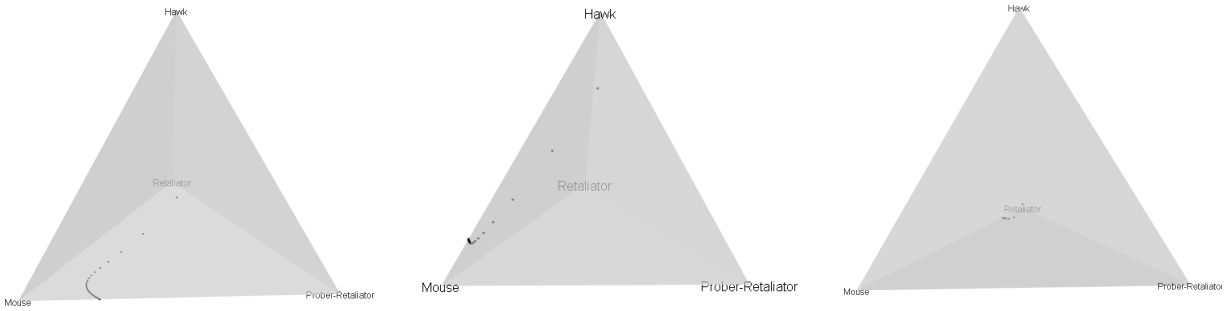
**Increasing the Cost of Serious Injury**

Changing `c` to 500:

| | Mouse | Hawk | Retaliator | Prober-Retaliator |
|---|---|---|---|---|

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **30** | 19.75 | **19.75** | 30 | **30** | 15.3 | **15.4** |
| Hawk | 79.75 | **79.75** | -20.1 | **-220.2** | -15.2 | **-205.8** | -14.9 | **-205.3** |
| Retaliator | 30 | **30** | -25.2 | **-234.6** | 30 | **30** | 12.2 | **-53.4** |
| Prober-Retaliator | 67.6 | **67.6** | -25.7 | **-235.1** | 14.9 | **-42.5** | 3.1 | **-97.2** |

$\sigma_0 = (0.25, 0.25, 0.25, 0.25)$ $\sigma_0 = (0.1, 0.7, 0.1, 0.1)$ $\sigma_0 = (0.1, 0.1, 0.7, 0.1)$




Unsurprisingly, when serious injury is more costly, the fitness of Hawk, Retaliator, and Prober-Retaliator decrease drastically. The dots in the phase diagram being further apart represents a faster change in the population, which makes sense because Hawk-Hawk, Hawk-Retaliator, and Hawk-ProberRetaliator contests are basically a whole degree of magnitude more costly. However, in the final state of the population, it is not only Mice but there is always the presence of one of the other strategies, depending on which one is represented more heavily in the initial population state. When the proportions of each strategy start out equal, the Mouse - Prober-Retaliator mix ends up dominating.

**Increasing the Cost of a Scratch**

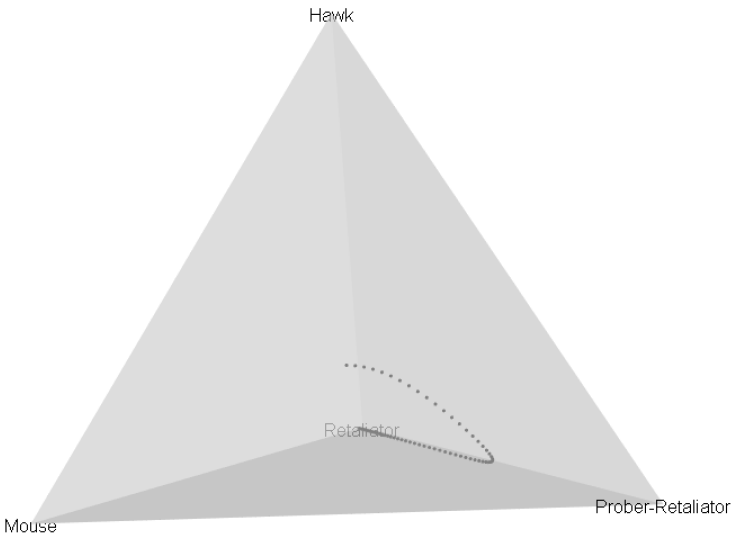
Increasing the cost of a scratch effectively causes the same change to the game and dynamics as increasing the cost of serious injury, just less dramatically.

**Increasing the Payoff of Winning**

Changing $v$ to 120:

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **60** | 19.75 | **19.75** | 30 | **60** | 15.3 | **19.2** |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Hawk | 79.75 | **139.75** | -20.1 | **10** | -15.2 | **15** | -14.9 | **14** |
| Retaliator | 30 | **60** | -25.2 | **5.5** | 30 | **60** | 12.2 | **41.3** |
| Prober-Retaliator | 67.6 | **123.9** | -25.7 | **5.4** | 14.9 | **46.2** | 3.1 | **34** |

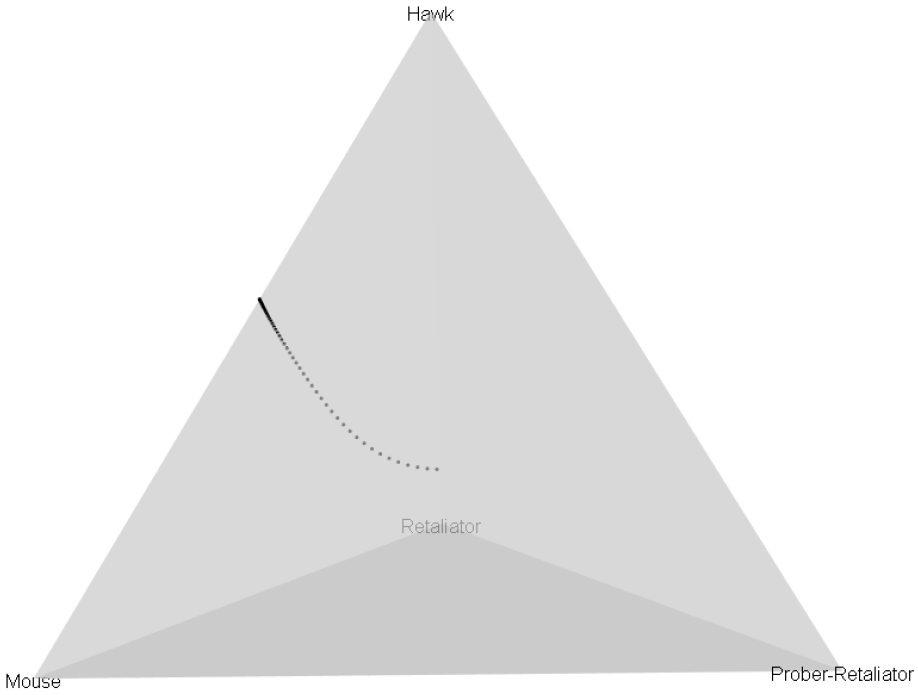



The payoff of all strategies increases, except for Mouse against Hawk and Prober-Retaliator, since they always lose in these matchups. In addition, all the payoffs are now positive. This makes sense because with v > c, it is always worth the cost to fight. With equal weights on each strategy in the initial population state, the population converges to all playing Retaliator.

**Decreasing the Probability of Retaliation**

Changing p_r from 1.0 to 0.5:

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **30** | 19.75 | **19.75** | 30 | **30** | 15.3 | **15.4** |
| Hawk | 79.75 | **79.75** | -20.1 | **-20.2** | -15.2 | **14.5** | -14.9 | **-8.6** |
| Retaliator | 30 | **30** | -25.2 | **-54.4** | 30 | **30** | 12.2 | **-3.2** |
| Prober-Retaliator | 67.6 | **67.9** | -25.7 | **-32.4** | 14.9 | **21.5** | 3.1 | **-2** |



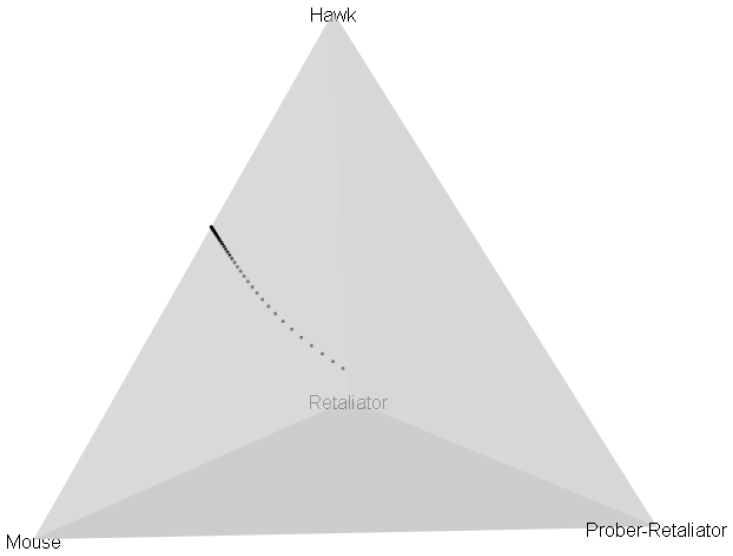

Decreasing the probability that a Retaliator or Prober-Retaliator will retaliate can be thought of as making them more Mouse-like. This decreases their payoff against Hawk, and is worse for the Retaliator than the Prober-Retaliator.

This suggests that when playing a Retaliator or Prober-Retaliator strategy, it is best to always retaliate.

**Increasing the Maximum Payoff of Saving Time and Energy (Producing Longer Contests)**

Changing `u_init` from 20 to 40:

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **30** | 19.75 | **39.75** | 30 | **30** | 15.3 | **30.9** |
| Hawk | 79.75 | **99.75** | -20.1 | **-10.2** | -15.2 | **29.9** | -14.9 | **4.2** |
| Retaliator | 30 | **30** | -25.2 | **-51.2** | 30 | **30** | 12.2 | **-23.6** |
| Prober-Retaliator | 67.6 | **89.9** | -25.7 | **-24.9** | 14.9 | **24.2** | 3.1 | **-8.8** |



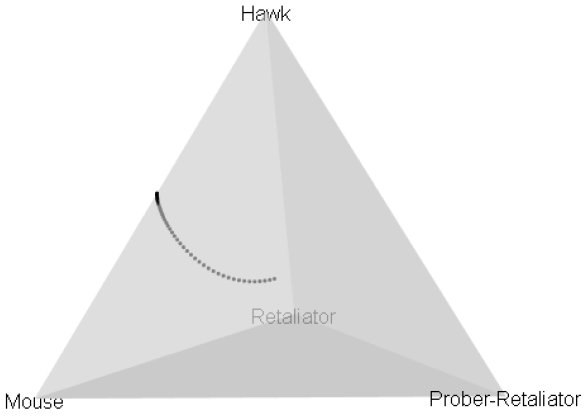

Lengthening the contest increases the payoff of Prober-Retaliators against Mice and Retaliators because longer contests mean they have a higher chance of getting in a probe that wins them the contest. However, against Prober-Retaliators, Retaliators and Prober-Retaliators do worse, perhaps for the same reason: more time means they have more chance to be probed and seriously injured. The Retaliator does worse against Hawks and Prober-Retaliators when the contests are prolonged, since the opponent always strikes first, and they get more chances to strike. Hawks do better in every circumstance, presumably because they always end contests earlier since they always play D. Mice also get higher payoff against Hawks and Prober-Retaliators because they get more payoff from saving time and energy as soon as the opponent attacks and they retreat. Overall, this change seems to favor Hawks because their aggression ends contests earlier, giving them a higher payoff from saving time and energy.

**Decreasing the Maximum Payoff of Saving Time and Energy (Producing Shorter Contests)**

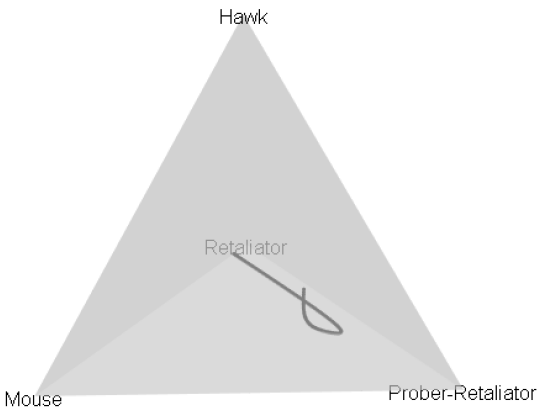
Changing `u_init` from 20 to 5:

| | Mouse | | Hawk | | Retaliator | | Prober-Retaliator | |
|---|---|---|---|---|---|---|---|---|
| Mouse | 30 | **30** | 19.75 | **4.75** | 30 | **30** | 15.3 | **19.1** |
| Hawk | 79.75 | **64.75** | -20.1 | **-27.6** | -15.2 | **18.8** | -14.9 | **-2.1** |
| Retaliator | 30 | **30** | -25.2 | **-50.4** | 30 | **30** | 12.2 | **19.8** |
| Prober-Retaliator | 67.6 | **43.1** | -25.7 | **-37.1** | 14.9 | **27.7** | 3.1 | **18.8** |

$\sigma_0 = (0.25, 0.25, 0.25, 0.25)$




$\sigma_0 = (0.1, 0.1, 0.4, 0.4)$




In this modified simulation with shorter contests and less payoff from saving time and energy, we see that the payoff of using a strike-first strategy (Hawk or Prober-Retaliator) against a Mouse decreases, probably because there is less payoff from ending the contest earlier. Every strategy does better against the Retaliator and Prober-Retaliator. The Retaliator and Prober-Retaliator also do worse against Mouse and Hawk.

When the initial population state is equally weighted on each strategy, the population converges to a Hawk-Mouse mix. If the initial proportion of Retaliator and Prober-Retaliator are high enough, the population converges to a Retaliator-Mouse mix. So although other strategies are more fit against Retaliators and Prober-Retaliators, when they collectively make up enough of the initial population, their fitness against each other and their quality of almost being ESS's allows them to grow while Hawks and Mouse decrease, and eventually the Retaliator becomes the fittest strategy against a population of mostly Retaliators, Prober-Retaliators, and Mice.

## Discussion

In all, these results support Maynard Smith and Price's hypothesis that "limited war" strategies are strategic not only from a group fitness perspective, but also benefit the individual.

One reason for this is because retaliation behavior decreases the fitness of "total war" strategies. Retaliation behavior decreases the fitness of Hawks, since it increases the probability of a Hawk receiving a serious injury during a contest. In the phase diagrams with Maynard Smith and Price's and my simulation's payoffs, the proportion of Hawk always decreases immediately and continues decreasing, and the population becomes dominated either by Retaliators or a mix of Prober-Retaliators and Mice.

If it is better to use "limited war" strategies like Retaliator and Prober-Retaliator, which is better? In most of the results, Retaliator was found to be an NSS, and Prober-Retaliator was not far off from being an NSS, which may make Retaliator the slightly better retaliation strategy. The

exception is that when there are more Mice in the population, it may be beneficial to occasionally be the initiator of more dangerous conflict. Figure 4 suggests that as the presence of Mice in the population increases, the fitness of Prober-Retaliator increases, since they will sometimes initiate a dangerous attack, whereas Retaliators will never have a chance to take advantage of a Mouse. Prober-Retaliator in this case may represent an animal that only employs some Hawk-like behavior, giving them some of Hawks' advantage against Mice but not as much of the disadvantage that Hawks have in every other matchup.

Coding the simulation was easily my favorite part. I thought many times that I could not do it or that it would not be worth the time and effort, but in the end, it was extremely cool to be able to actually run the simulation and observe what happened—live—in response to a change to the parameters of the model. It greatly increased the possibilities and the ease/speed of exploration. Imagine doing this in 1973—I'm just imagining Maynard Smith and Price waiting hours for their punch-card computer to go through 15 matchups, 2,000 trials each. No wonder they didn't go into greater depth in exploring different changes to the model!

I am so thankful for the phase diagrams and the power they gave me to investigate how the pay-off matrices I obtained would actually play out in a population, and how this would change based on the initial population state. The EvolutionaryGames package by Gebele and Staudacher was released only in April of this year, and the rgl package that allowed me to rotate the 3D diagrams was actually only released two weeks ago! Especially if you count the simulation that I coded within the past few days, the amount of very new software/programs that made this project possible is pretty cool.

Overall, I have learned that as one adds more strategies (in this project, I explored from Hawk-Dove to Hawk-Dove-Replicator to Hawk-Mouse-Replicator-ProberReplicator) and more nuances (even with the 7 or so parameters in Maynard Smith and Price's model, there were so many potential adjustments to be explored), the complexity of a model and its dynamics increases exponentially. Many times, when I observed a certain result, I could not explain why it was the way it was, and it would be very difficult to verify that any explanation I could give was truly the explanation (and multiple forces could easily be at play). Even the extent to which I was able to analyze the population dynamics with four strategies was difficult but insightful. The simulations' ability to predict, visualize, and explain the dynamics of animal conflict far surpassed my own.

There are many complexities not captured by my models, many more speculations to be made about the results I have found, and many avenues and applications still unexplored. I hope there is something interesting to you within these pages.

My code in Python for the simulation and in R for the phase diagrams is in the appendix.

```
import random
from tabulate import tabulate

# global parameters
p_i = 0.10 # probability of injury
p_p = 0.05 # probability of probing
p_r = 1.00 # probability of retaliation

v = 60   # payoff of winning
c = 100  # cost of serious injury
s = 2    # cost of scratch
u_init = 20  # payoff of saving time and energy (counts down every round)
u_minus = 0.5

num_trials = 100000

class Hawk:
    def __init__(self, name):
        self.name = name
        self.payoff = 0
        self.currentAction = "C"
        self.injured = False
        self.attacked = False

    def move(self):
        if self.injured:
            self.currentAction = "R"
        else:
            self.currentAction = "D"
#         print(self.name + ": " + self.currentAction)

class Mouse:
    def __init__(self, name):
        self.name = name
        self.payoff = 0
        self.currentAction = "C"
        self.injured = False
        self.attacked = False

    def move(self, u):
        if self.attacked or u == 0:
            self.currentAction = "R"
        else:
            self.currentAction = "C"
#         print(self.name + ": " + self.currentAction)


class Retaliator:
    def __init__(self, name):
        self.name = name
        self.payoff = 0
        self.currentAction = "C"
        self.injured = False
        self.attacked = False

    def move(self, opponent, u):
        if self.injured or u == 0:
```

```
            self.currentAction = "R"
        elif opponent.currentAction == "D":
            if random.random() <= p_r:
                self.currentAction = "D"
            else:
                self.currentAction = "C"
        elif opponent.currentAction == "C":
            self.currentAction = "C"
#         print("      " + self.name + ": " + self.currentAction)


class ProberRetaliator:
    def __init__(self, name):
        self.name = name
        self.payoff = 0
        self.currentAction = "C"
        self.injured = False
        self.attacked = False
        self.probe = False

    def move(self, opponent, u):
        if self.injured or u == 0:
            self.currentAction = "R"

        elif not self.probe: # if it's not currently probing
            if opponent.currentAction == "D":
                if random.random() <= p_r:
                    self.currentAction = "D"      # retaliates

            elif opponent.currentAction == "C":
                if random.random() <= p_p:
                    self.currentAction = "D"      # probes
                    self.probe = True
                else:
                    self.currentAction = "C"

        else: # if it is currently probing
            if opponent.currentAction == "D": # opponent has retaliated
                self.currentAction = "C"  # reverts back to C
                self.probe = False
            elif opponent.currentAction == "C":
                self.currentAction = "D"  # takes advantage and keeps probing

#         print(self.name + ": " + self.currentAction)


def check(contestant, opponent):
    if contestant.currentAction == "D":
        if not isinstance(opponent, Mouse):
            if random.random() <= p_i: # 1 in 10 chance
                opponent.payoff -= c # serious injury
                opponent.injured = True
            else:
                opponent.payoff -= s # scratch
        else:
            opponent.attacked = True
```

```
    if contestant.currentAction == "R":
        opponent.payoff += v
        return False

    return True


def main():

    """
    Hawk vs. Hawk
    """
    total = 0
    total2= 0

    for _ in range(num_trials):
        p1 = Hawk("p1")
        p2 = Hawk("p2")

        u = u_init  #saved-time payoff starts at u_init

        while True:
            # p1 goes first
            p1.move()

            # check payoffs
            if not check(p1, p2):
                break

            # p2 goes next
            p2.move()

#             u -= u_minus

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of all trials, average of p1 and p2
    print("Hawk vs. Hawk avg payoff: " + str((total+total2)/(2*num_trials)))
    print("\n")
    hh = (total+total2)/(2*num_trials)
```

```
    """
    Hawk vs. Mouse
    """
    total = 0
    total2 = 0
    for _ in range(num_trials):
        p1 = Hawk("p1")
        p2 = Mouse("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move()

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Hawk vs. Mouse avg payoff: " + str(total/num_trials))
    print("   Mouse's payoff: " + str(total2/num_trials))
    print("\n")
    hm1 = total/num_trials
    mh2 = total2/num_trials


    """
    Mouse vs. Hawk
    """
```

```
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Mouse("p1")
        p2 = Hawk("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move()

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Mouse vs. Hawk avg payoff: " + str(total/num_trials))
    print("   Hawk's payoff: " + str(total2/num_trials))
    print("\n")
    mh1 = total/num_trials
    hm2 = total2/num_trials



    """
    Mouse vs. Mouse
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
```

```
        p1 = Mouse("p1")
        p2 = Mouse("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -=u_minus

            # p2 moves
            p2.move(u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Mouse vs. Mouse avgavg payoff: " +
str((total+total2)/(2*num_trials)))
    print("\n")
    mm = (total+total2)/(2*num_trials)



    """
    Retaliator vs. Mouse
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Retaliator("p1")
        p2 = Mouse("p2")

        u = u_init
```

```
        while True:
            # p1 moves
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Retaliator vs. Mouse avg payoff: " + str(total/num_trials))
    print("   Mouse's payoff: "+ str(total2/num_trials))
    print("\n")
    rm1 = total/num_trials
    mr2 = total2/num_trials



    """
    Mouse vs. Retaliator
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Mouse("p1")
        p2 = Retaliator("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(u)
```

```
            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(p1, u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Mouse vs. Retaliator avg payoff: " + str(total/num_trials))
    print("   Retaliator's payoff: " + str(total2/num_trials))
    print("\n")
    mr1 = total/num_trials
    rm2 = total2/num_trials



    """
    Retaliator vs. Hawk
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Retaliator("p1")
        p2 = Hawk("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break
```

```
#             u -= u_minus

            # p2 moves
            p2.move()

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Retaliator vs. Hawk avg payoff: " + str(total/num_trials))
    print("   Hawk's payoff: " + str(total2/num_trials))
    print("\n")
    rh1 = total/num_trials
    hr2 = total2/num_trials



    """
    Hawk vs. Retaliator
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Hawk("p1")
        p2 = Retaliator("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move()

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(p1, u)
```

```
            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Hawk vs. Retaliator avg payoff: " + str(total/num_trials))
    print("   Retaliator's payoff: " + str(total2/num_trials))
    print("\n")
    hr1 = total/num_trials
    rh2 = total2/num_trials



    """
    Retaliator vs. Retaliator
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Retaliator("p1")
        p2 = Retaliator("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(p1, u)

            # check payoffs
            if not check(p2, p1):
                break
```

```
            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Retaliator vs. Retaliator avg payoff: " +
str((total+total2)/(2*num_trials)))
    print("\n")
    rr = (total+total2)/(2*num_trials)



    """
    Prober-Retaliator vs. Mouse
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = ProberRetaliator("p1")
        p2 = Mouse("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
```

```
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Prober-Retaliator vs. Mouse avg payoff: " + str(total/num_trials))
    print("   Mouse's payoff: " + str(total2/num_trials))
    print("\n")
    pm1 = total/num_trials
    mp2 = total2/num_trials



    """
    Mouse vs. Prober-Retaliator
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Mouse("p1")
        p2 = ProberRetaliator("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(p1, u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
```

```
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Mouse vs. Prober-Retaliator avg payoff: " + str(total/num_trials))
    print("   Prober-Retaliator's payoff: " + str(total2/num_trials))
    print("\n")
    mp1 = total/num_trials
    pm2 = total2/num_trials



    """
    Prober-Retaliator vs. Hawk
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = ProberRetaliator("p1")
        p2 = Hawk("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move()

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
```

```
            total += p1.payoff
            total2 += p2.payoff

        # calculate average of num_trials trials
        print("Prober-Retaliator vs. Hawk avg payoff: " + str(total/num_trials))
        print("   Hawk's payoff: " + str(total2/num_trials))
        print("\n")
        ph1 = total/num_trials
        hp2 = total2/num_trials



        """
        Hawk vs. Prober-Retaliator
        """
        total = 0
        total2 = 0

        for _ in range(num_trials):
            p1 = Hawk("p1")
            p2 = ProberRetaliator("p2")

            u = u_init

            while True:
                # p1 moves
                p1.move()

                # check payoffs
                if not check(p1, p2):
                    break

#               u -= u_minus

                # p2 moves
                p2.move(p1, u)

                # check payoffs
                if not check(p2, p1):
                    break

                # decrease time-saved payoff
                u -= u_minus

            # uninjured players get time-saved payoff
            if u < 0:
                u = 0
            if not p1.injured:
                p1.payoff += u
            if not p2.injured:
                p2.payoff += u

            # add final payoffs to totals
            total += p1.payoff
            total2 += p2.payoff

        # calculate average of num_trials trials
```

```
    print("Hawk vs. Prober-Retaliator avg payoff: " + str(total/num_trials))
    print("   Prober-Retaliator's payoff: " + str(total2/num_trials))
    print("\n")
    hp1 = total/num_trials
    ph2 = total2/num_trials



    """
    Prober-Retaliator vs. Retaliator
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = ProberRetaliator("p1")
        p2 = Retaliator("p2")

        u = u_init

        while True:
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(p1, u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals

        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Prober-Retaliator vs. Retaliator avg payoff: " +
str(total/num_trials))
    print("   Retaliator's payoff: " + str(total2/num_trials))
    print("\n")
```

```
    pr1 = total/num_trials
    rp2 = total2/num_trials



    """
    Retaliator vs. Prober-Retaliator
    """
    total = 0
    total2 = 0

    for _ in range(num_trials):
        p1 = Retaliator("p1")
        p2 = ProberRetaliator("p2")

        u = u_init

        while True:
            # p1 moves
            p1.move(p2, u)

            # check payoffs
            if not check(p1, p2):
                break

#             u -= u_minus

            # p2 moves
            p2.move(p1, u)

            # check payoffs
            if not check(p2, p1):
                break

            # decrease time-saved payoff
            u -= u_minus

        # uninjured players get time-saved payoff
        if u < 0:
            u = 0
        if not p1.injured:
            p1.payoff += u
        if not p2.injured:
            p2.payoff += u

        # add final payoffs to totals
        total += p1.payoff
        total2 += p2.payoff

    # calculate average of num_trials trials
    print("Retaliator vs. Prober-Retaliator avg payoff: " +
str(total/num_trials))
    print("   Prober-Retaliator's payoff: " + str(total2/num_trials))
    print("\n")
    rp1 = total/num_trials
    pr2 = total2/num_trials
```

```
"""
Prober-Retaliator vs. Prober-Retaliator
"""
total = 0
total2 = 0

for _ in range(num_trials):
    p1 = ProberRetaliator("p1")
    p2 = ProberRetaliator("p2")

    u = u_init

    while True:
        # p1 moves
        p1.move(p2, u)

        # check payoffs
        if not check(p1, p2):
            break

        # p2 moves
        p2.move(p1, u)

        # check payoffs
        if not check(p2, p1):
            break

        # decrease time-saved payoff
        u -= u_minus

    # uninjured players get time-saved payoff
    if u < 0:
        u = 0
    if not p1.injured:
        p1.payoff += u
    if not p2.injured:
        p2.payoff += u

    # add final payoffs to totals
    total += p1.payoff
    total2 += p2.payoff

# calculate average of num_trials trials
print("PR vs. PR avg payoff: " + str((total+total2)/(2*num_trials)))
print("\n")
pp = (total+total2)/(2*num_trials)



"""
Print out-puts as table
"""
table = [["","Mouse","Hawk","Retaliator","PR"],
         ["Mouse",mm,(mh1+mh2)/2,(mr1+mr2)/2,(mp1+mp2)/2],
         ["Hawk",(hm1+hm2)/2,hh,(hr1+hr2)/2,(hp1+hp2)/2],
```

```
            ["Retaliator",(rm1+rm2)/2,(rh1+rh2)/2,rr,(rp1+rp2)/2],
            ["PR",(pm1+pm2)/2,(ph1+ph2)/2,(pr1+pr2)/2,pp]]
    print(tabulate(table, headers = "firstrow"))


if __name__ == "__main__":
    main()
```

Output used to generate Table 6 (`num_trials = 100000`):

```
Hawk vs. Hawk avg payoff: -20.1133125

Hawk vs. Mouse avg payoff: 80.0
   Mouse's payoff: 20.0

Mouse vs. Hawk avg payoff: 19.5
   Hawk's payoff: 79.5

Mouse vs. Mouse avgavg payoff: 30.0

Retaliator vs. Mouse avg payoff: 0.0
   Mouse's payoff: 60.0

Mouse vs. Retaliator avg payoff: 0.0
   Retaliator's payoff: 60.0

Retaliator vs. Hawk avg payoff: -25.050225
   Hawk's payoff: -15.5846

Hawk vs. Retaliator avg payoff: -14.90955
   Retaliator's payoff: -25.412825

Retaliator vs. Retaliator avg payoff: 30.0

Prober-Retaliator vs. Mouse avg payoff: 63.982175
   Mouse's payoff: 19.426175

Mouse vs. Prober-Retaliator avg payoff: 11.266175
   Prober-Retaliator's payoff: 71.266175

Prober-Retaliator vs. Hawk avg payoff: -26.092925
   Hawk's payoff: -14.32605

Hawk vs. Prober-Retaliator avg payoff: -15.49565
   Prober-Retaliator's payoff: -25.26095

Prober-Retaliator vs. Retaliator avg payoff: -6.2448
   Retaliator's payoff: 33.412425

Retaliator vs. Prober-Retaliator avg payoff: -9.1115
```

```
   Prober-Retaliator's payoff: 35.982575

PR vs. PR avg payoff: 3.0617
```

Sample table output (with `p_i = 0.90` and `num_trials = 100000`):

| | Mouse | Hawk | Retaliator | Prober-Retaliator |
|---|---|---|---|---|
| Mouse | 30 | 19.75 | 30 | 15.3448 |
| Hawk | 79.75 | -10.136 | 63.5529 | 59.6246 |
| Retaliator | 30 | -84.0495 | 30 | -69.4299 |
| Prober-Retaliator | 67.6996 | -80.1159 | 53.8537 | -11.6767 |

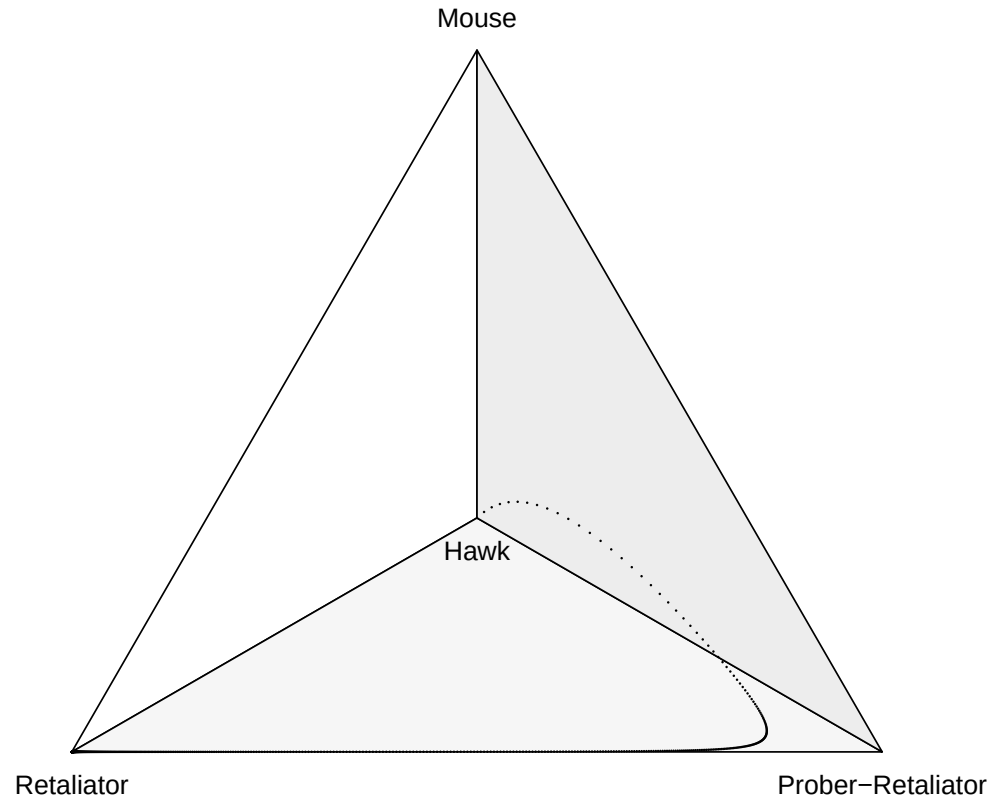
# Appendix B: Replicator Dynamic Phase Diagram Generator (R)

(and some bonus diagrams)

```
# packages used
library(EvolutionaryGames)
library(rgl)
```

**Mouse, Hawk, Retaliator, Prober-Retaliator (MS&P's payoffs)**

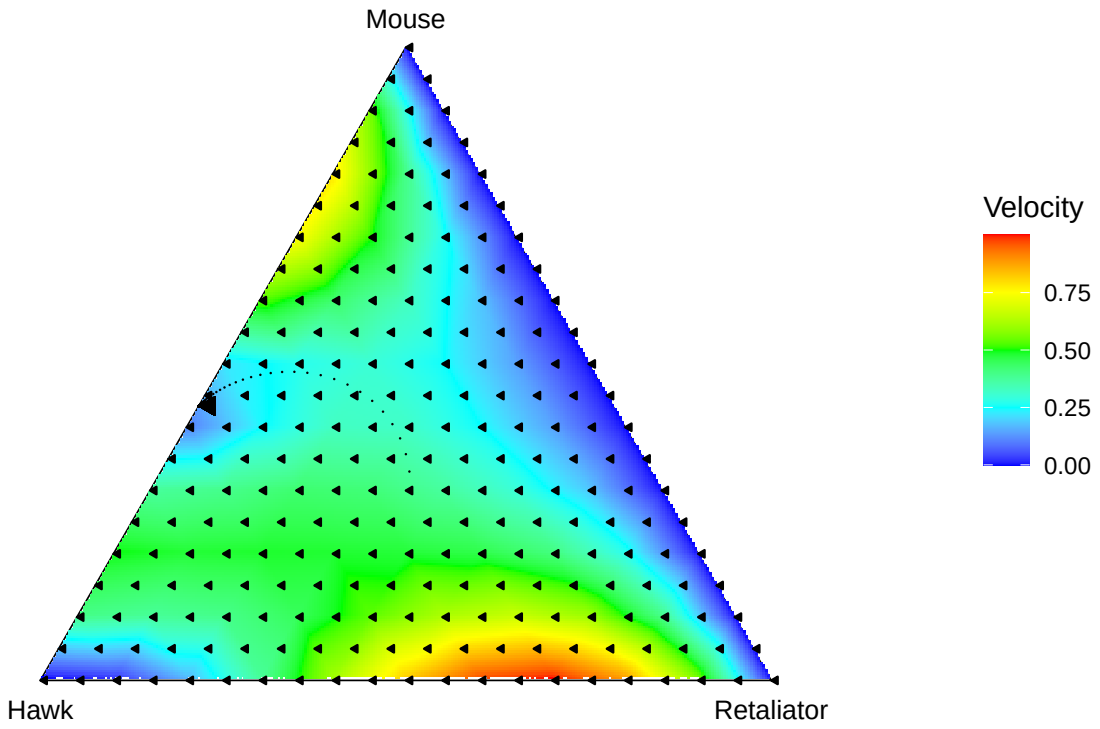
```
# create the payoff matrix for the game
A <- matrix(c(29, 19.5, 29, 17.2,
              80, -19.5, -18.1, -18.9,
              29, -22.3, 29, 23.1,
              56.7, -20.1, 26.9, 21.9),
            4, byrow=TRUE)

# specify the names of the strategies, in order of column
strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

# specify the initial population state
state <- c(.25, .25, .25, .25)

# plot phase diagram (3D for 4 strategies)
phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



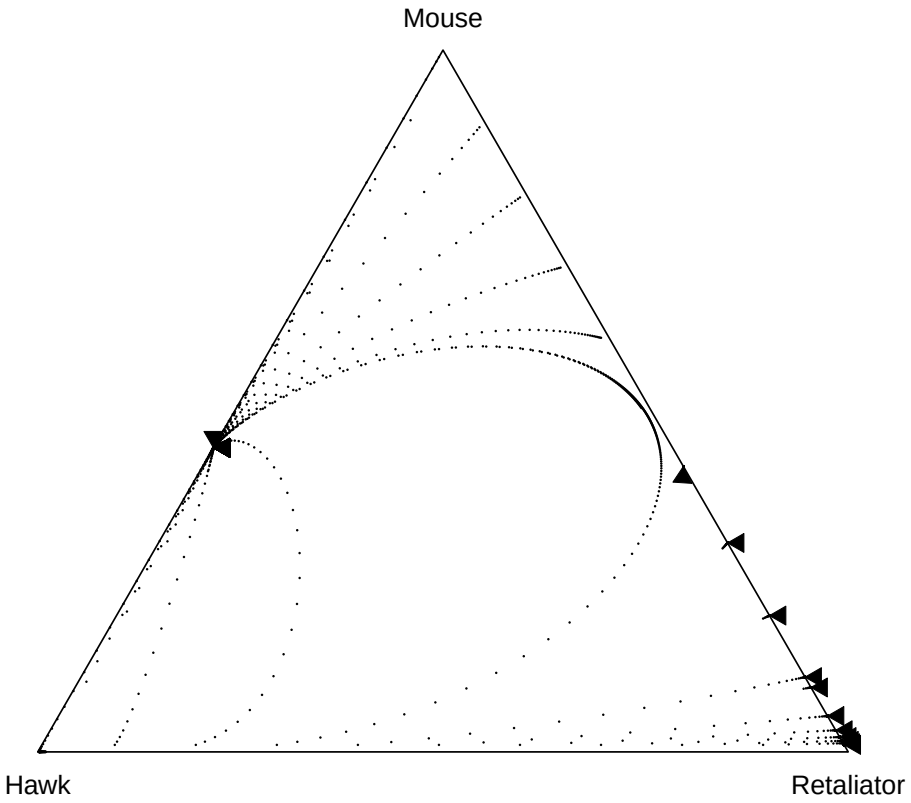

```
# if you run with noRGL = F, the rgl package will create an interactive 3D diagram
# that you can rotate!
```

**Mouse, Hawk, Retaliator (MS&P)**

```
A <- matrix(c(29, 19.5, 29,
              80, -19.5, -18.1,
              29, -22.3, 29),
            3, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator")

state <- matrix(c(.33, .33, .34), 1, 3, byrow = T)

# this one is 3D for 2 strategies
# it also has a contour map showing the speed of change
# and a vector field showing the direction
phaseDiagram3S(A, dynamic = Replicator, params = NULL, trajectories = state,
               contour = T, vectorField = T, strategies)
```



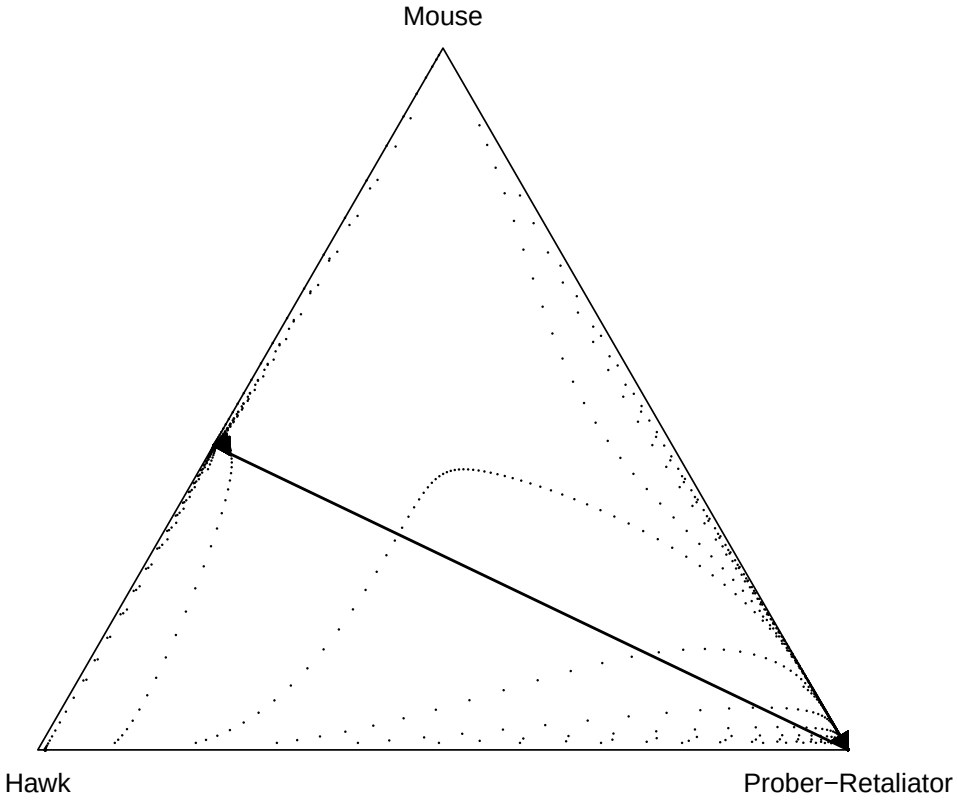

```
# you can also find the set of ESSs and graph their paths
ESset(A, strategies)
```




```
##           [,1]      [,2] [,3]
## [1,] 0.4333333 0.5666667    0
```

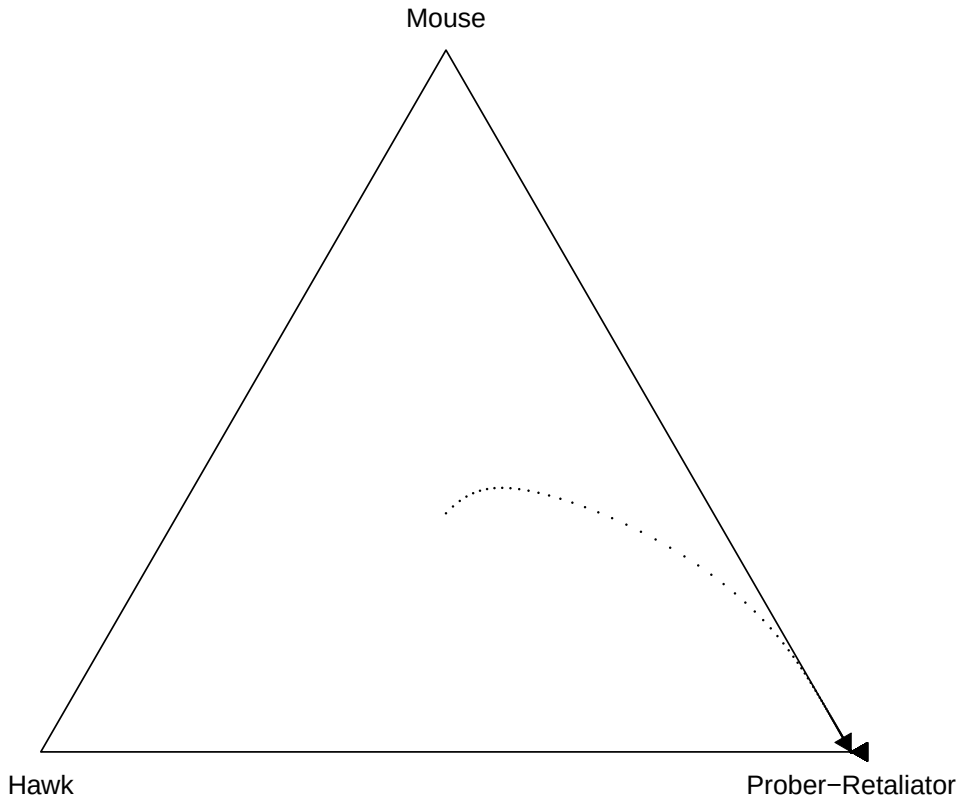
**Mouse, Hawk, Prober-Retaliator (MS&P)**

```
A <- matrix(c(29, 19.5, 17.2,
              80, -19.5, -18.9,
              56.7, -20.1, 21.9),
            3, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Prober-Retaliator")

ESset(A, strategies)
```




```
##           [,1]      [,2] [,3]
## [1,] 0.0000000 0.0000000    1
## [2,] 0.4333333 0.5666667    0
```

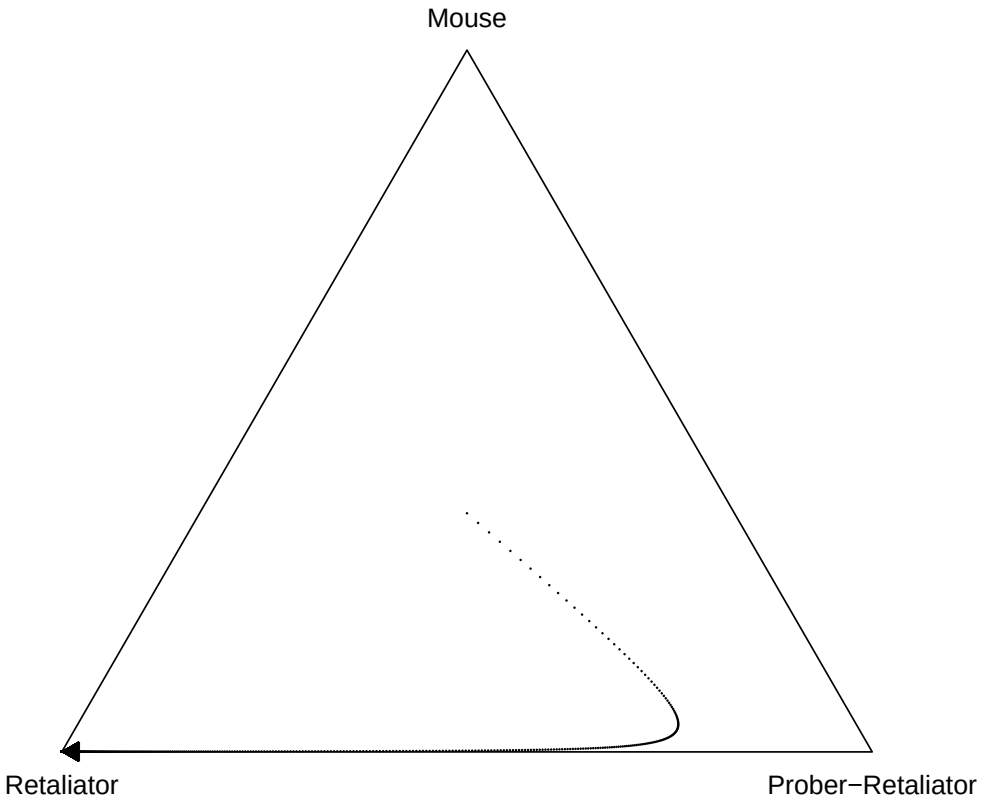
```
state <- matrix(c(.34, .33, .33), 1, 3, byrow = T)

phaseDiagram3S(A, dynamic = Replicator, params = NULL, trajectories = state,
               contour = F, vectorField = F, strategies)
```




**Mouse, Retaliator, Prober-Retaliator (MS&P)**

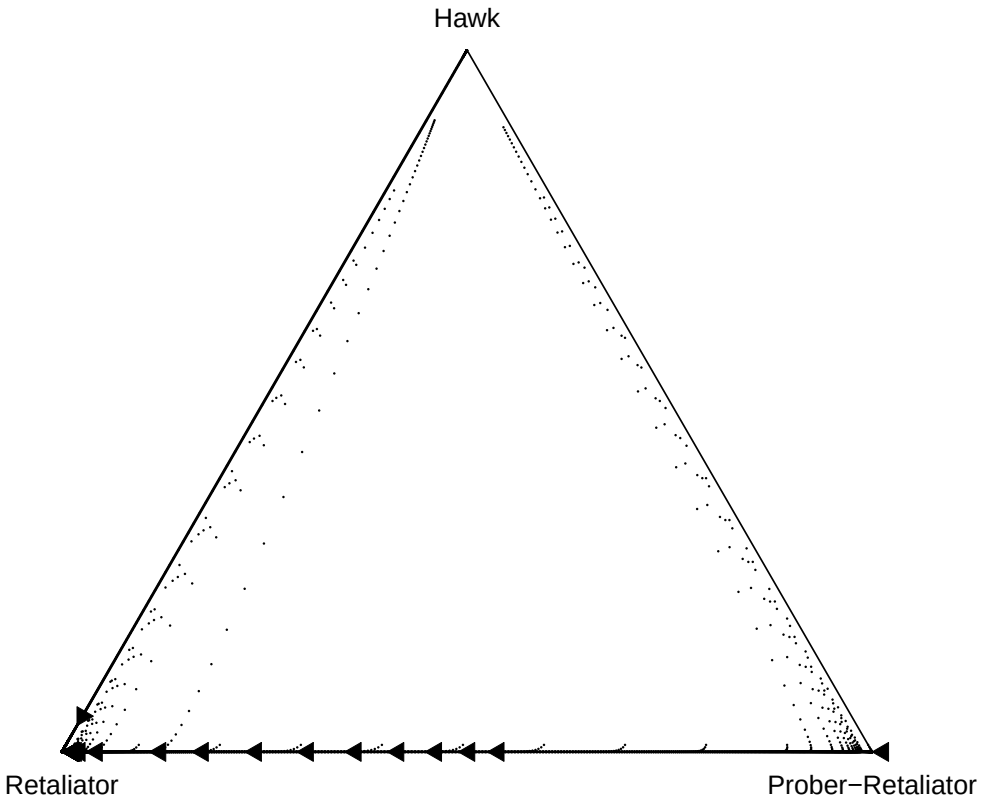
This one has no ESS.

```
A <- matrix(c(29, 29, 17.2,
              29, 29, 23.1,
              56.7, 26.9, 21.9),
            3, byrow=TRUE)

strategies <- c("Mouse", "Retaliator", "Prober-Retaliator")

state <- matrix(c(.34, .33, .33), 1, 3, byrow = T)

phaseDiagram3S(A, dynamic = Replicator, params = NULL, trajectories = state,
               contour = F, vectorField = F, strategies)
```




**Hawk, Retaliator, Prober-Retaliator (MS&P)**

```
A <- matrix(c(-19.5, -18.1, -18.9,
              -22.3, 29, 23.1,
              -20.1, 26.9, 21.9),
            3, byrow=TRUE)

strategies <- c("Hawk", "Retaliator", "Prober-Retaliator")

ESset(A, strategies)
```

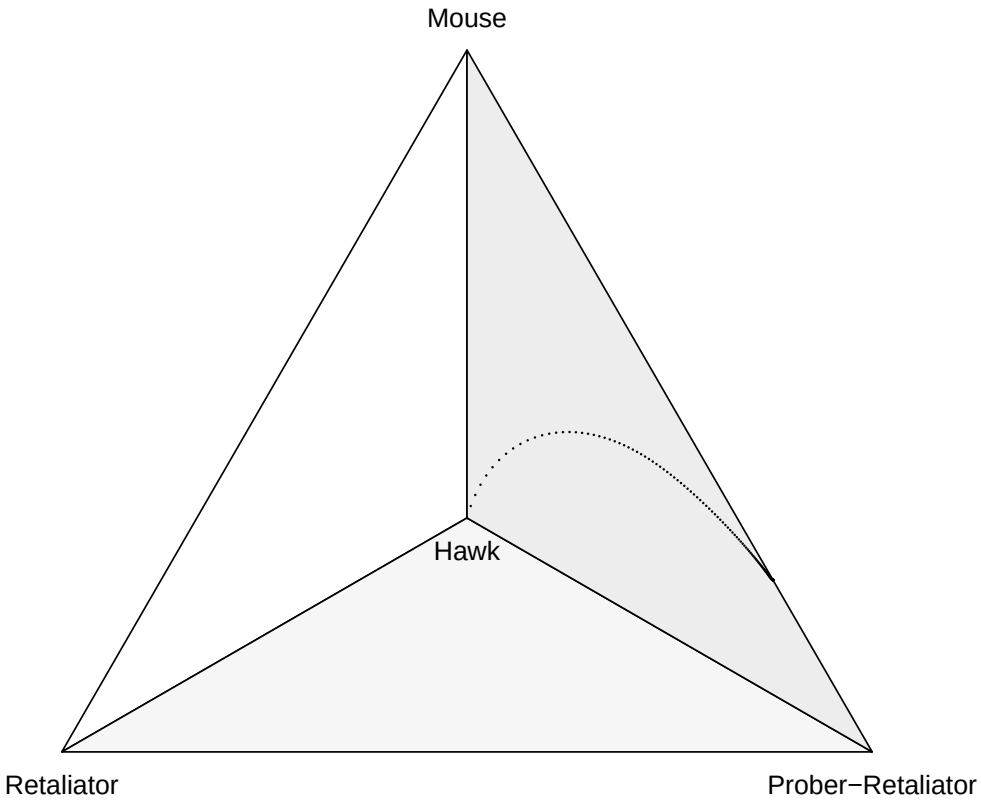



```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
```

## Phase Diagrams from My Computer Simulation

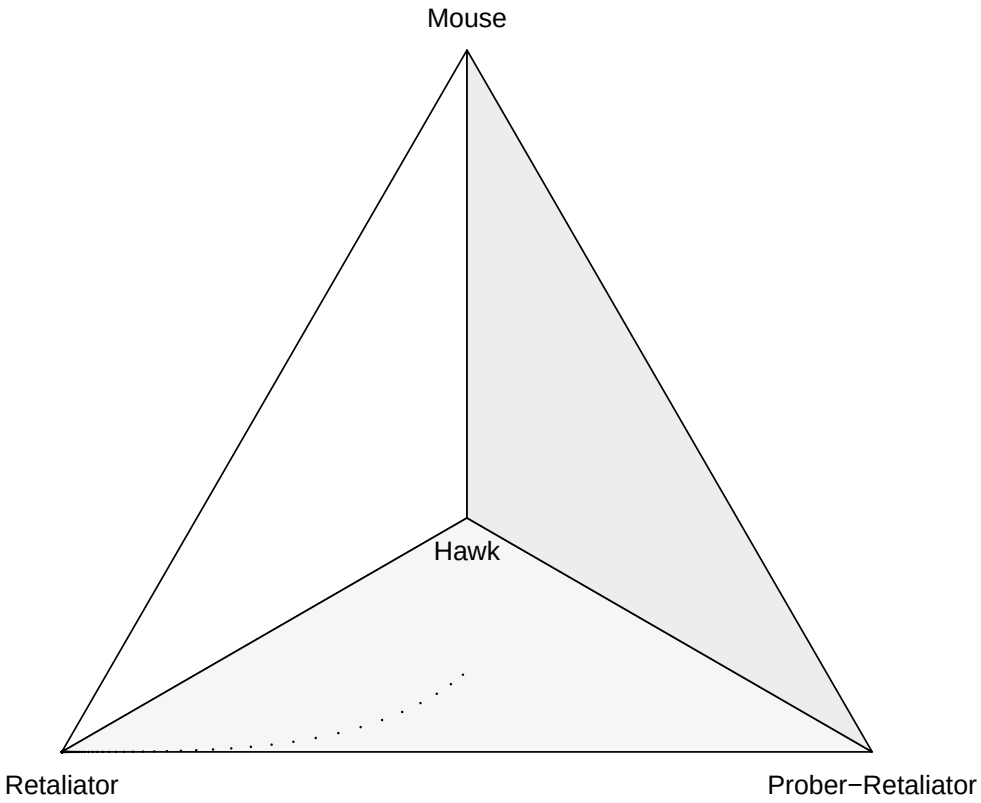
### Mouse, Hawk, Retaliator, Prober-Retaliator (my payoffs)

```
A <- matrix(c(30, 19.75, 30, 15.3,
              79.75, -20.1, -15.2, -14.9,
              30, -25.3, 30, 12.2,
              67.6, -25.7, 14.9, 3.1),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



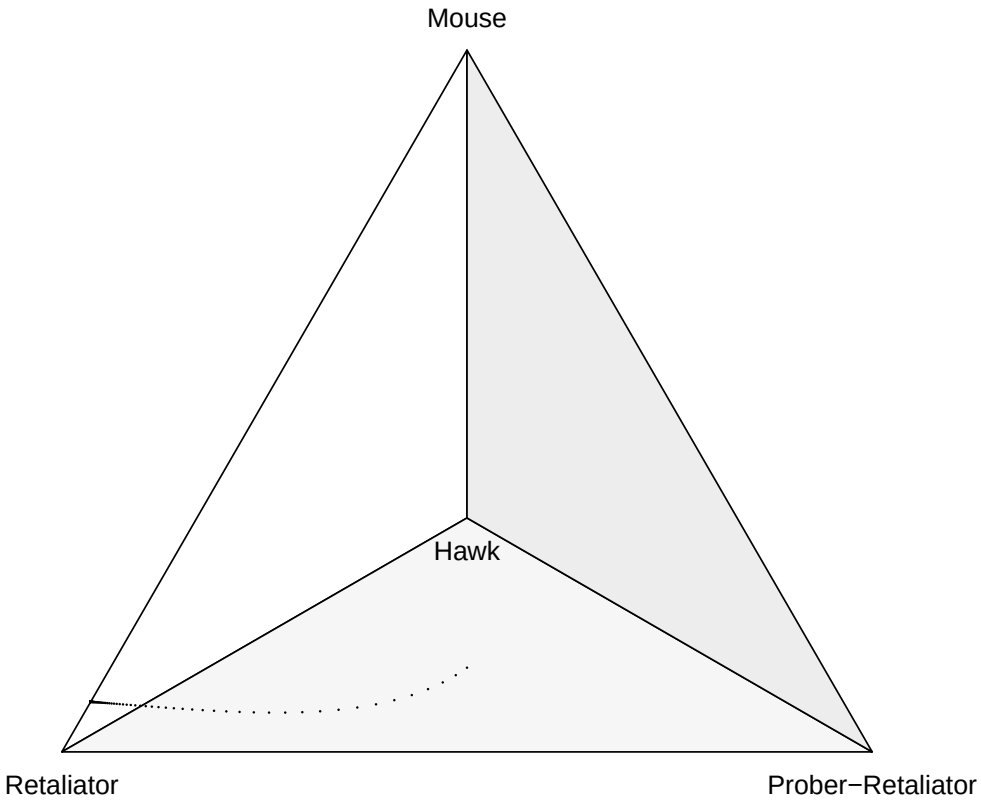

**Mouse, Hawk, Retaliator, Prober-Retaliator (my, Mouse starts at 0)**

```
state <- c(0, 0.33, 0.33, 0.33)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



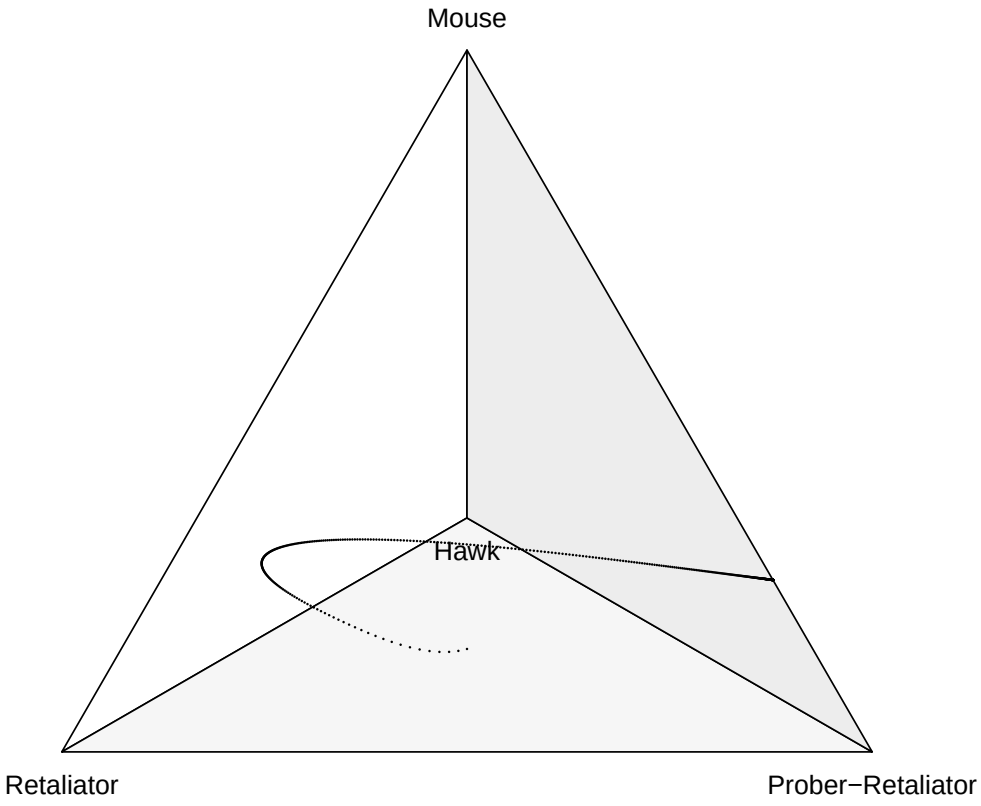

**Mouse, Hawk, Retaliator, Prober-Retaliator (my, Mouse starts at 0.01)**

```
state <- c(0.01, 0.33, 0.33, 0.33)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```




**Mouse, Hawk, Retaliator, Prober-Retaliator (my, Mouse starts at 0.04)**

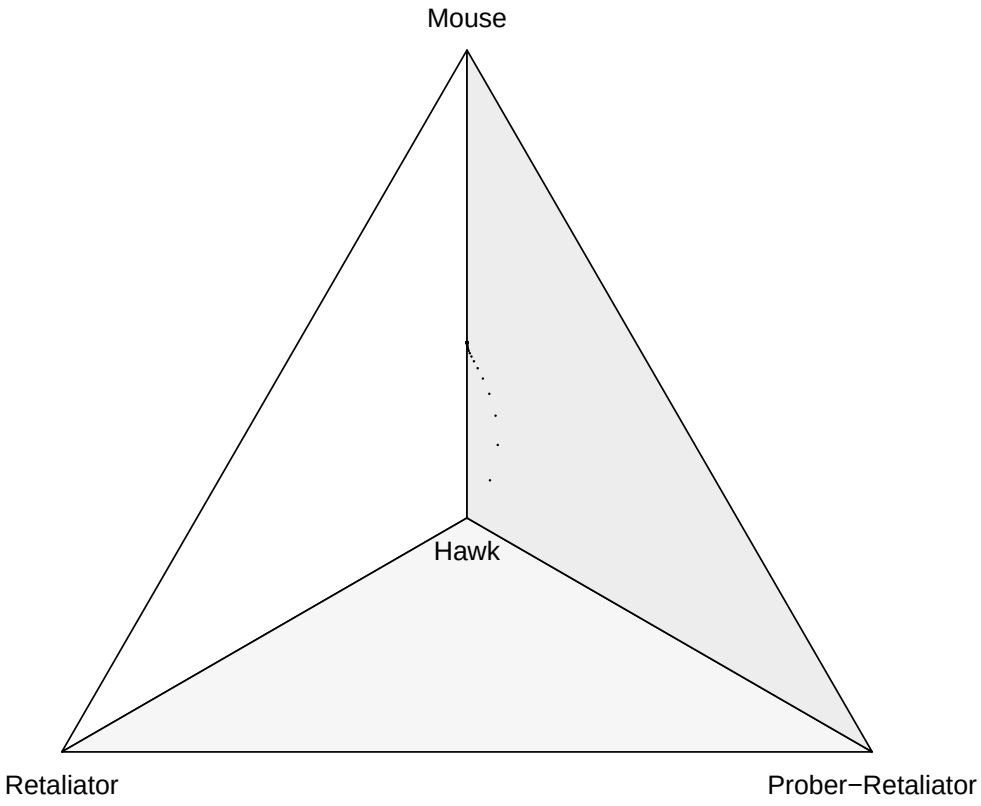
```
state <- c(0.04, 0.32, 0.32, 0.32)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```




## Phase Diagrams from Changing Model Parameters

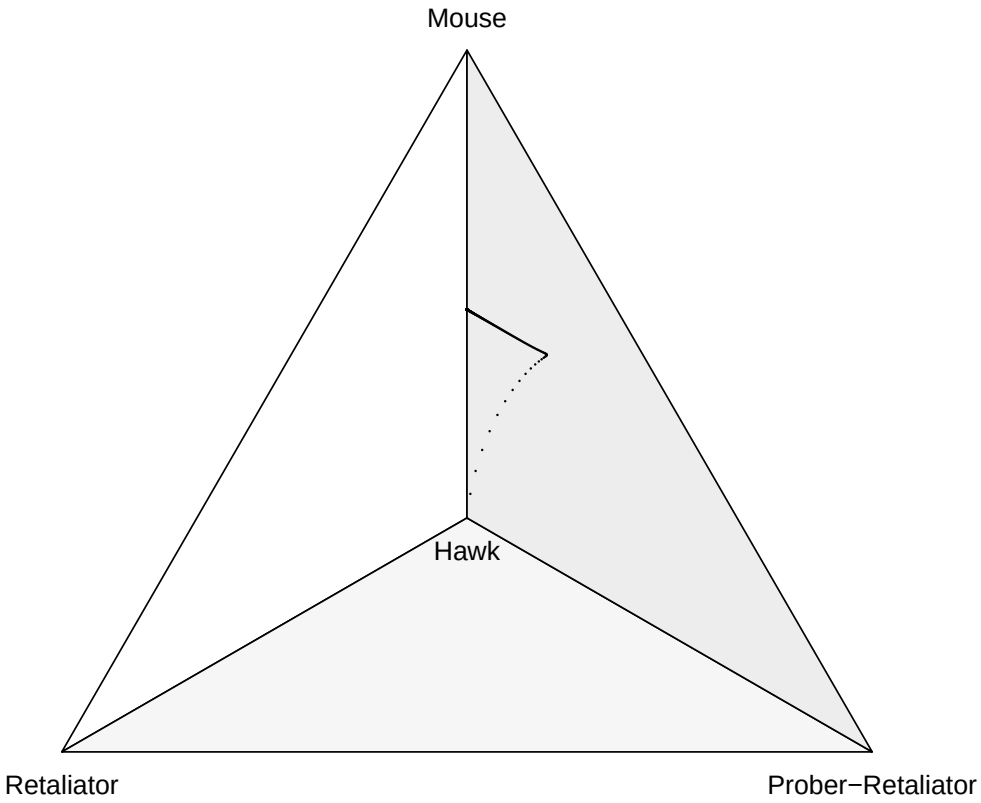
**Increasing the probability of serious injury (p_i = 0.90)**

```
A <- matrix(c(30, 19.75, 30, 15.4,
              79.75, -10.1, 63.4, 59.8,
              30, -83.9, 30, -69.5,
              67.6, -80.3, 53.8, -11.7),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



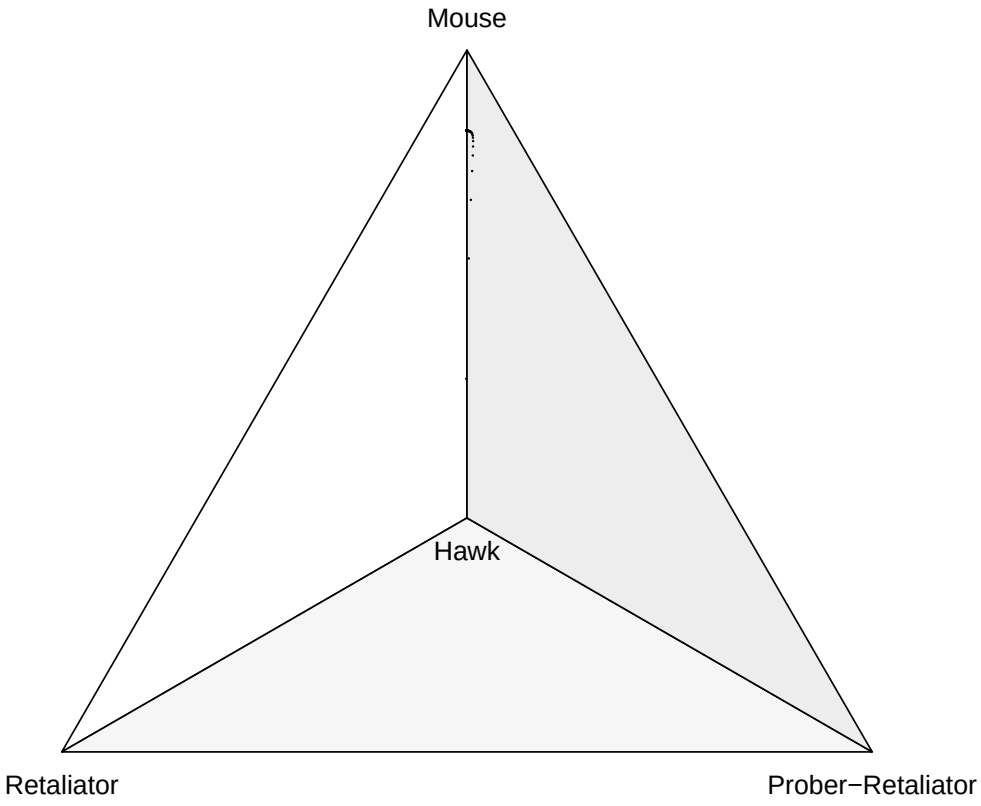

**Increasing the probability of probing behavior (p_p = 0.5)**

```
A <- matrix(c(30, 19.75, 30, 19.2,
              79.75, -20.2, -14.8, -15.3,
              30, -25.6, 30, -23.5,
              79.2, -25.1, -13.8, -20.4),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



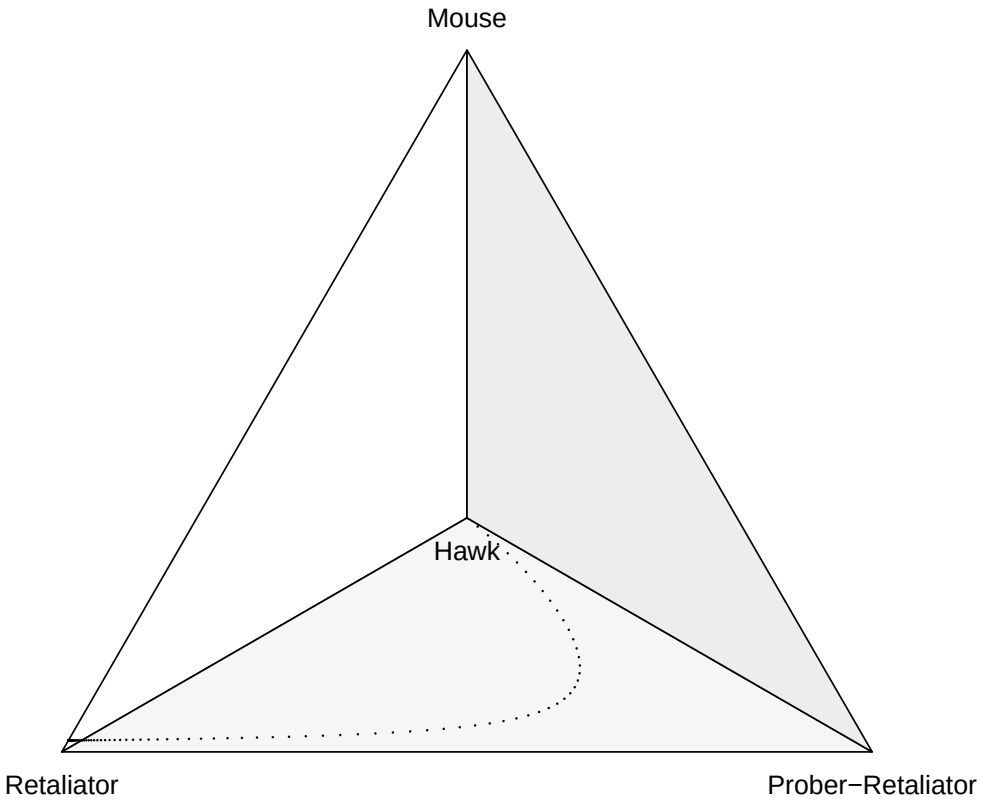

**Increasing the cost of serious injury (c = 500)**

```
A <- matrix(c(30, 19.75, 30, 15.4,
              79.75, -220.2, -205.8, -205.3,
              30, -234.6, 30, -53.4,
              67.6, -235.1, -42.5, -97.2),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.1, 0.7, 0.1, 0.1)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



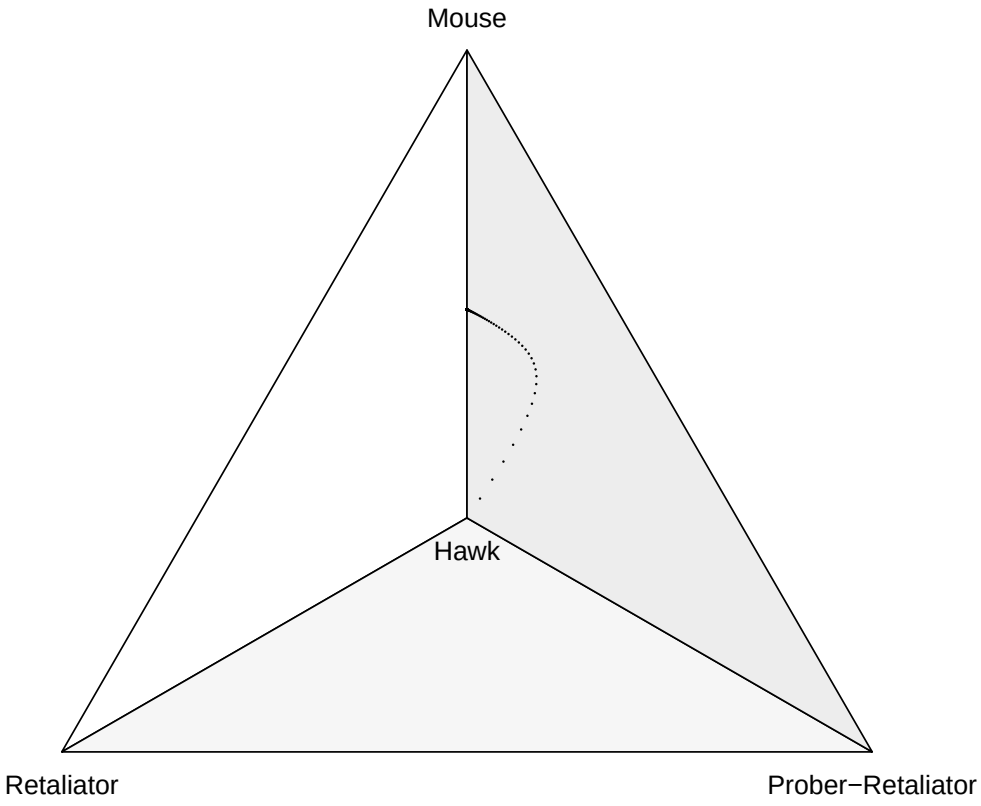

**Increasing the payoff of winning (v = 120)**

```
A <- matrix(c(60, 19.75, 60, 19.2,
              139.75, 10, 15, 14,
              60, 5.5, 60, 41.3,
              123.9, 5.4, 46.2, 34),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



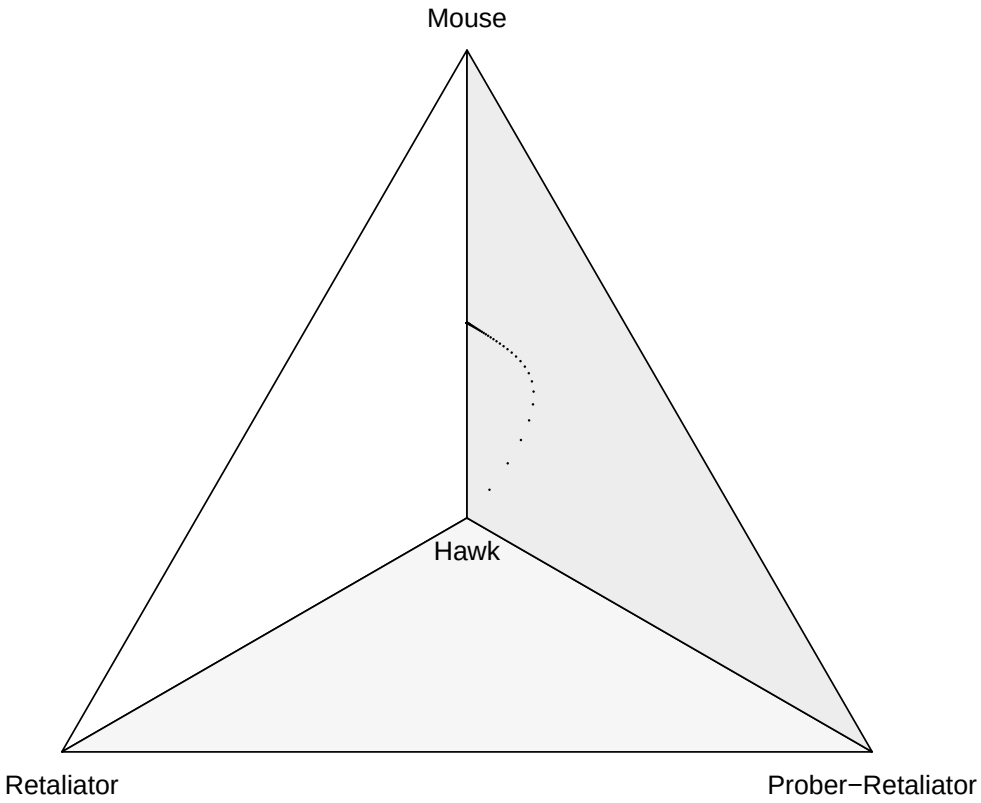

**Decreasing the probability of retaliation (p_r = 0.5)**

```
A <- matrix(c(30, 19.75, 30, 15.4,
              79.75, -20.2, 14.5, -8.6,
              30, -54.4, 30, -3.2,
              67.6, -32.4, 21.5, -2),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```



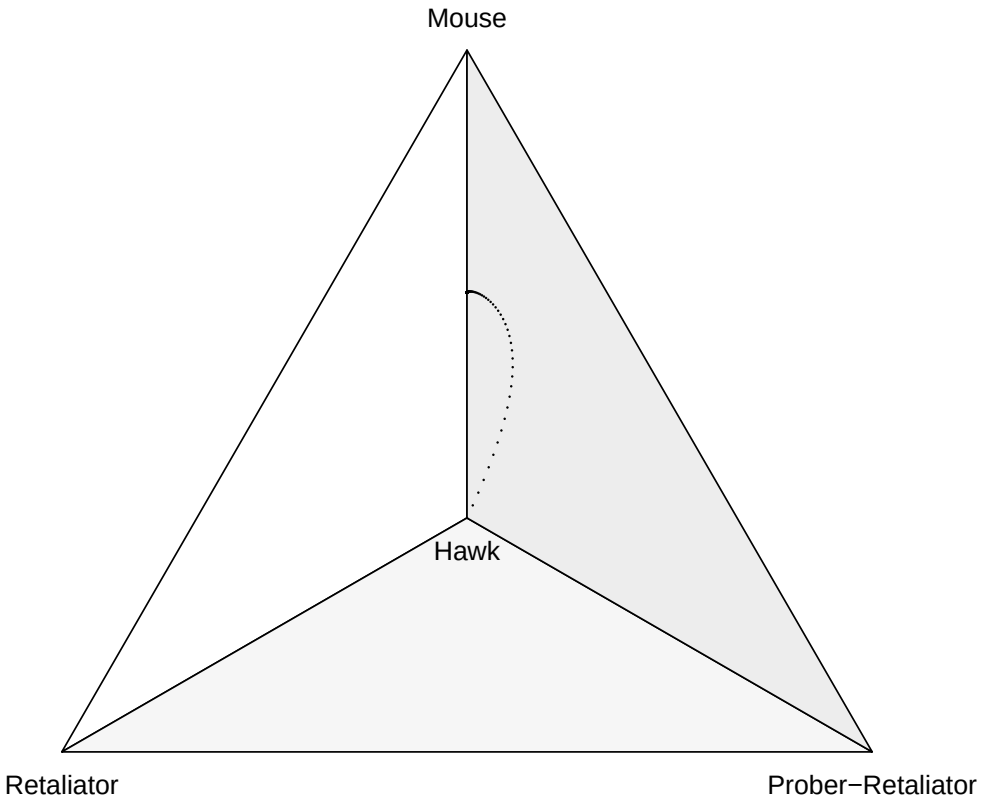

**Increasing the maximum payoff of saving time and energy, producing longer contests (u_init = 40)**

```
A <- matrix(c(30, 39.75, 30, 30.9,
              99.75, -10.2, 29.9, 4.2,
              30, -51.2, 30, -23.6,
              89.9, -24.9, 24.2, -8.8),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```

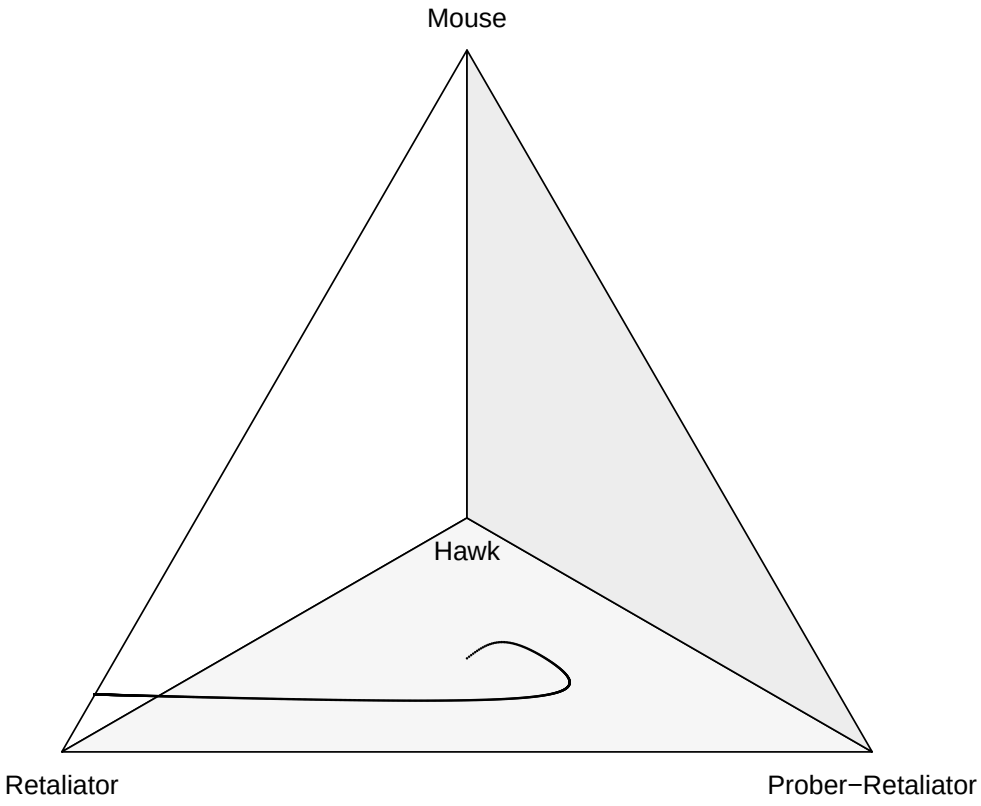



**Decreasing the maximum payoff of saving time and energy, producing shorter contests (u_init = 5)**

```
A <- matrix(c(30, 4.75, 30, 19.1,
              64.75, -27.6, 18.8, -2.1,
              30, -50.4, 30, 19.8,
              43.1, -37.1, 27.7, 18.8),
            4, byrow=TRUE)

strategies <- c("Mouse", "Hawk", "Retaliator", "Prober-Retaliator")

state <- c(0.25, 0.25, 0.25, 0.25)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```




```
state <- c(0.1, 0.1, 0.4, 0.4)

phaseDiagram4S(A, Replicator, NULL, state, noRGL = T, strategies)
```


Mouse
Hawk
Retaliator
Prober−Retaliator

**Works Cited**

F., R. (2020, November 19). *Two Big Mule Deer Bucks Fighting During The Rut* [Photograph]. Flickr. https://www.flickr.com/photos/fetherbrain/51023089168

Gebele, D., & Staudacher, J. (2022). EvolutionaryGames: Important Concepts of Evolutionary Game Theory. R package version 0.1.1. https://cran.r-project.org/package=EvolutionaryGames

Maynard Smith, J., & Price, G. R. (1973). The Logic of Animal Conflict. *Nature*, *246*(5427), 15–18. https://doi.org/10.1038/246015a0

Murdoch, D., & Adler, D. (2022). rgl: 3D Visualization Using OpenGL. R package version 0.108.3.2. https://cran.r-project.org/package=rgl

R Core Team (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.