# Study of Predator-Prey Dynamics

Vincent Edwards, Julia Corrales, and Rachel Gossard

April 13, 2025

## Lotka-Volterra Model (LVM)

The Lotka-Volterra Model (LVM) models the dynamics between a predator species ($y$) and a prey species ($x$) over time ($t$).

$$\frac{dy}{dt} = -ay + bxy \quad \frac{dx}{dt} = +dx - cyx$$

The model depends on four parameters.

- $a$: decay rate of the predators
- $b$: proportionality for how predators grow due to eating prey
- $c$: proportionality for how prey decay due to being eaten by predators
- $d$: growth rate of the prey

Dividing the differential equations by each other yields $\frac{dy}{dx}$.

$$\frac{dy}{dx} = \frac{b}{c} \frac{y\left(x - \frac{a}{b}\right)}{x\left(\frac{d}{c} - y\right)}$$

Since this equation does not depend explicitly on time, it can be used to create phase portraits. The solutions swirl counter-clockwise around $x = \frac{a}{b}$ and $y = \frac{d}{c}$.

## Types of Simulation Outcomes

## Conditions for Good Modeling

```python
parameters = {
    "breed_time": 3,
    "energy_gain": 4,
    "breed_energy": 15,
    "side_length": 80,
    "aspect_ratio": 9/8,
    "initial_fish": 500,
    "initial_sharks": 400,
    "steps": 500,
    "start_energy": 9,
    "use_basic_setup": True,
}

def get_initialization_parameters(params):
    """
    Return a dictionary with just the parameters needed to initialize the game array.
    """
    result = {}
    desired_keys = ["initial_fish", "initial_sharks", "breed_time", "breed_energy"]
    for key in desired_keys:
```

```python
        result[key] = params[key]
    return result

def get_simulation_parameters(params):
    """
    Return a dictionary with just the parameters needed to run the simulation.
    """
    result = {}
    desired_keys = ["steps", "breed_time", "energy_gain", "breed_energy",
"start_energy"]
    for key in desired_keys:
        result[key] = params[key]
    return result
```

```python
import wa_tor
import default_parameters
import matplotlib.pyplot as plt

def test_outcome_chances(target_param, test_values, trials, params=None):
    """
    Vary the target parameter to have the given test values.
    For each value, run the simulation for the specified number of trials and
calculate the chance of each of the possible outcomes.
    - Everything went extinct
    - Fish fill the board
    - Simulation could keep going

    Return a dictionary containing three lists of chances, one list for each outcome.
    Each list contains the chances found using each test value of the target
parameter.
    """
    # Set the parameters to the default if not specified
    if params is None:
        params = default_parameters.parameters.copy()

    # Calculate the board dimensions based on side length and aspect ratio
    side_length = params["side_length"]
    other_side = int(side_length * params["aspect_ratio"])
    dims = [side_length, other_side]

    overall_chances = {
        "everything_extinct": [],
        "fish_fill_board": [],
        "still_going": [],
    }

    for value in test_values:
        # Set the target parameter
        params[target_param] = value
        # Extract the needed parameters for later steps
        init_params = default_parameters.get_initialization_parameters(params)
```

```
        sim_params = default_parameters.get_simulation_parameters(params)
        # Keep track of the counts for the possible outcomes
        everything_extinct_count = 0
        fish_fill_count = 0

        for _ in range(trials):
            # Initialize the game array
            initial_game_array = wa_tor.create_empty_game_array(dims)
            if params["use_basic_setup"]:
                wa_tor.initialize_game_array_randomly(initial_game_array,
**init_params)
            else:
                wa_tor.initialize_game_array_circular(initial_game_array,
**init_params)

            # Run the simulation
            fish_counts, shark_counts =
wa_tor.run_simulation_minimal(initial_game_array, **sim_params)

            # Check whether fish filled the board or if sharks and fish both went
extinct
            # Update the counts for these events
            size = dims[0] * dims[1]
            if fish_counts[-1] + shark_counts[-1] < 0:
                everything_extinct_count += 1
            elif fish_counts[-1] == size:
                fish_fill_count += 1

        # Store the chances of each possible outcome
        still_going_count = trials - everything_extinct_count - fish_fill_count
        overall_chances["everything_extinct"].append(everything_extinct_count /
trials)
        overall_chances["fish_fill_board"].append(fish_fill_count / trials)
        overall_chances["still_going"].append(still_going_count / trials)

    return overall_chances

def plot_and_test_extinction_chances(fname, target_param, test_values, trials,
params=None):
    """
    Run the function test_outcome_chances() with the given arguments, then plot the
results.
    Save the figure at the given file name.
    """
    outcome_chances = test_outcome_chances(target_param, test_values, trials, params)

    fig, ax = plt.subplots()
    ax.plot(test_values, outcome_chances["everything_extinct"], "o", label="Both
Extinct")
    ax.plot(test_values, outcome_chances["fish_fill_board"], "^", label="Sharks
Extinct")
```

```
    ax.plot(test_values, outcome_chances["still_going"], ".", label="Neither Extinct")
    ax.set(xlabel=target_param, ylabel="Chance")
    ax.legend()
    fig.tight_layout()
    fig.savefig(fname)
```

```
import test_outcome_chances as tst

trials = 25
target_parameter = "breed_time"
start_value = 1
end_value = 15

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + 1), trials)
```

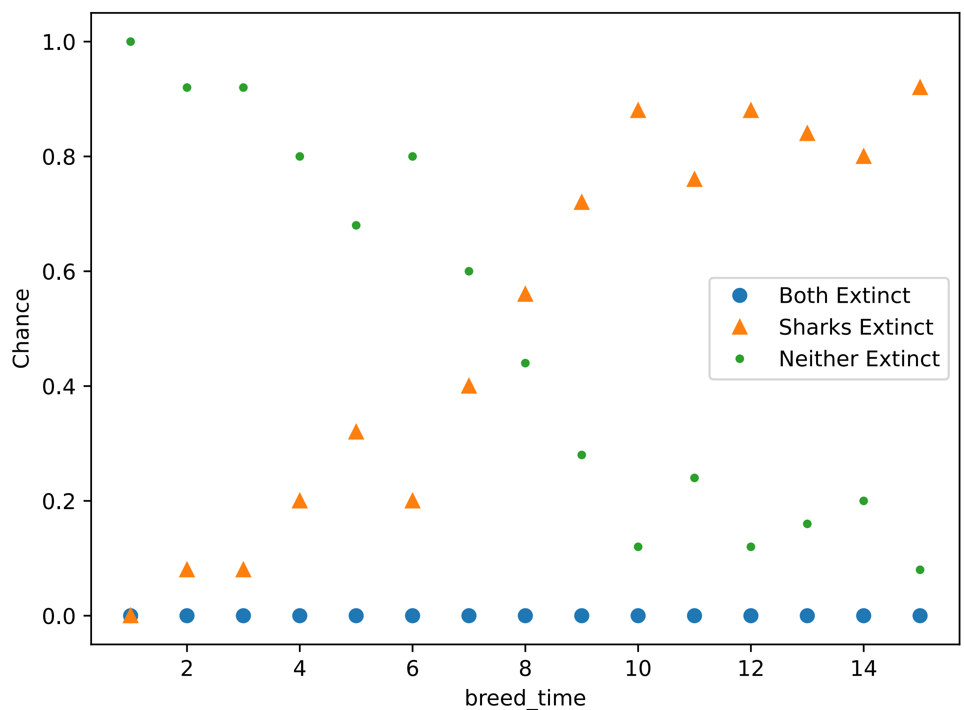


Figure 1: Outcome Chances vs `breed_time`

```
import test_outcome_chances as tst

trials = 25
target_parameter = "energy_gain"
start_value = 2
end_value = 18

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + 1), trials)
```

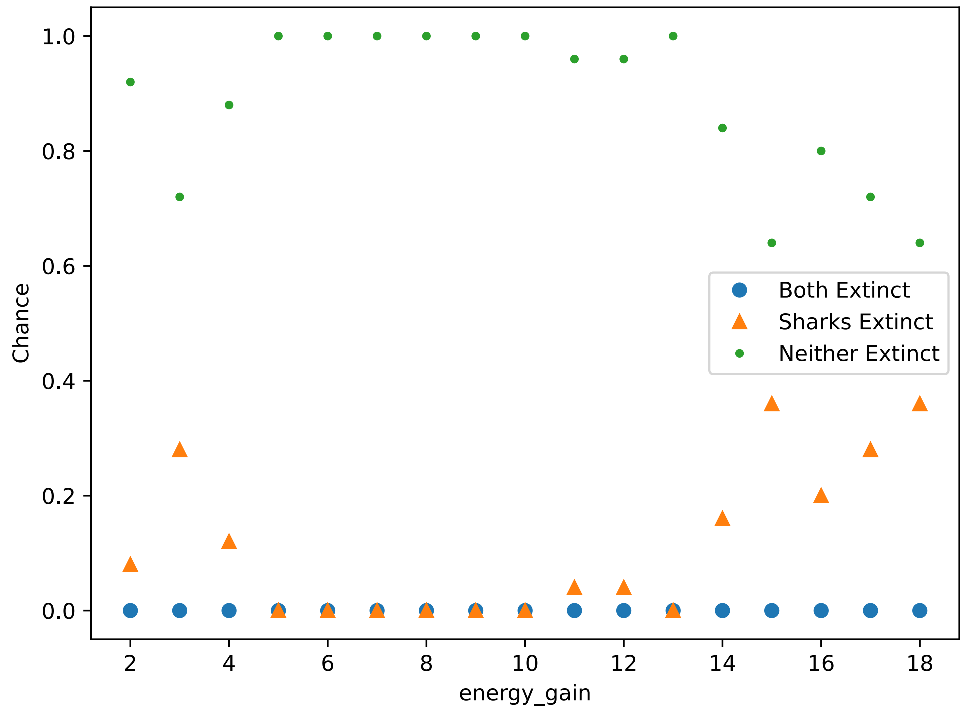


Figure 2: Outcome Chances vs `energy_gain`

```
import test_outcome_chances as tst
import default_parameters

trials = 25
target_parameter = "breed_energy"
start_value = default_parameters.parameters["start_energy"] + 1
end_value = 25

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + 1), trials)
```

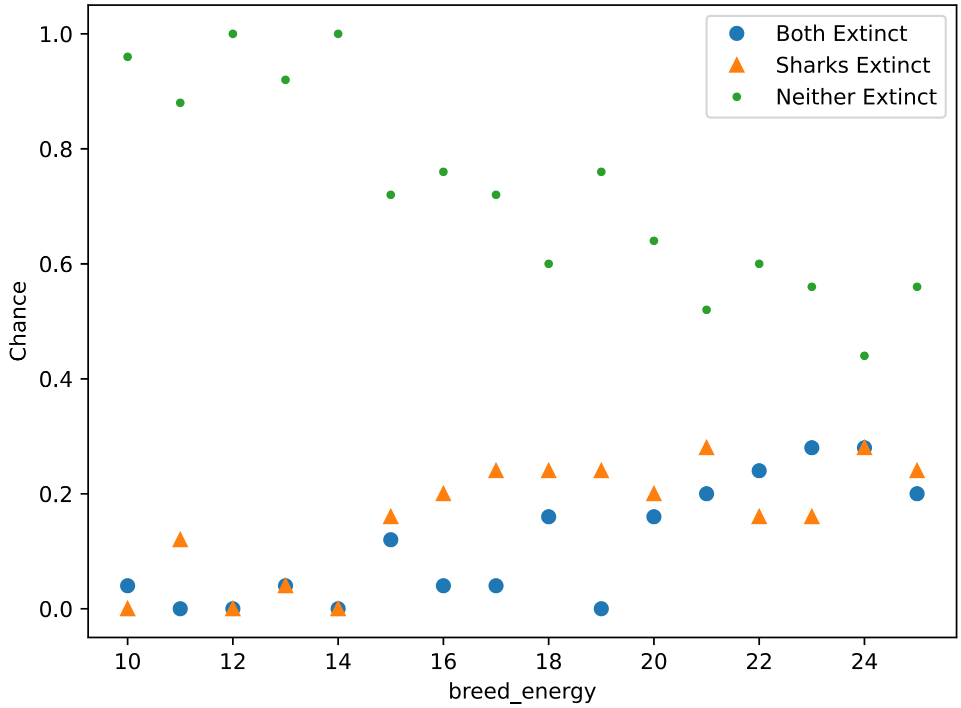


Figure 3: Outcome Chances vs `breed_energy`

```
import test_outcome_chances as tst

trials = 25
target_parameter = "side_length"
step_size = 10
start_value = 40
end_value = 120

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + step_size, step_size), trials)
```

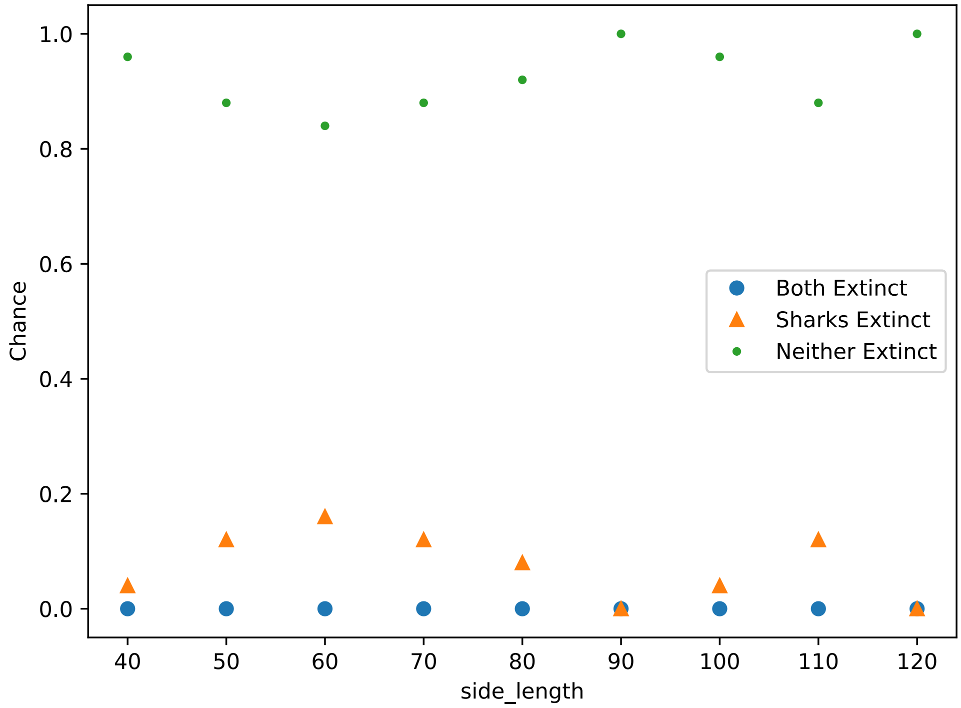


Figure 4: Outcome Chances vs `side_length`

```
import test_outcome_chances as tst

trials = 25
target_parameter = "aspect_ratio"
test_values = [i/8 for i in range(8, 16 + 1)]

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, test_values, trials)
```

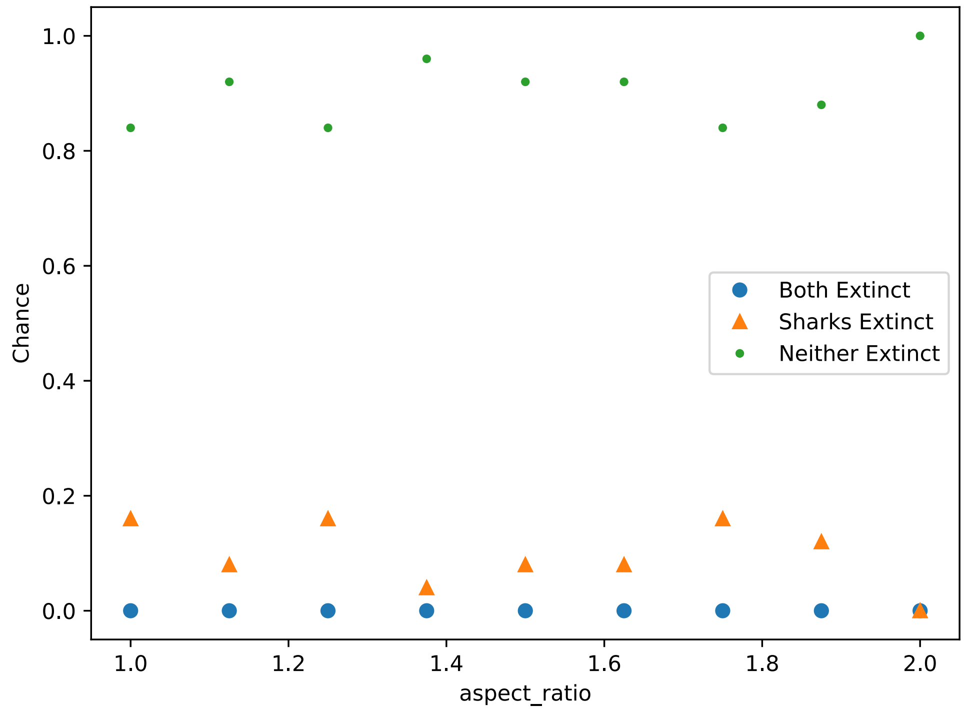


Figure 5: Outcome Chances vs `aspect_ratio`

```
import test_outcome_chances as tst

trials = 25
target_parameter = "initial_fish"
step_size = 50
start_value = 200
end_value = 1000

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + step_size, step_size), trials)
```

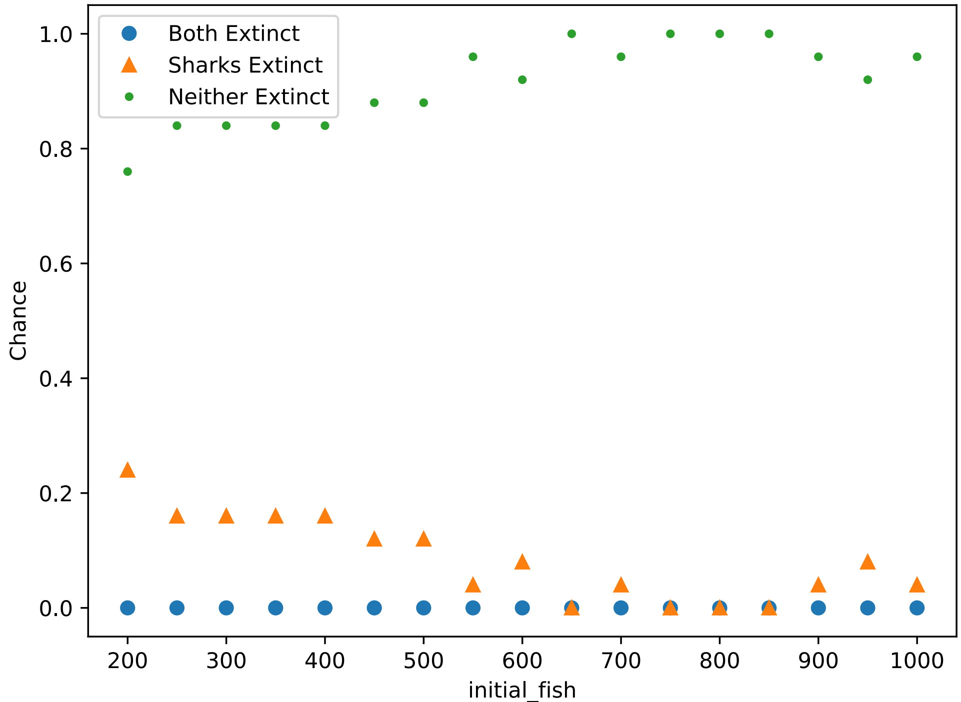


Figure 6: Outcome Chances vs `initial_fish`

```
import test_outcome_chances as tst

trials = 25
target_parameter = "initial_sharks"
step_size = 50
start_value = 200
end_value = 1000

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + step_size, step_size), trials)
```

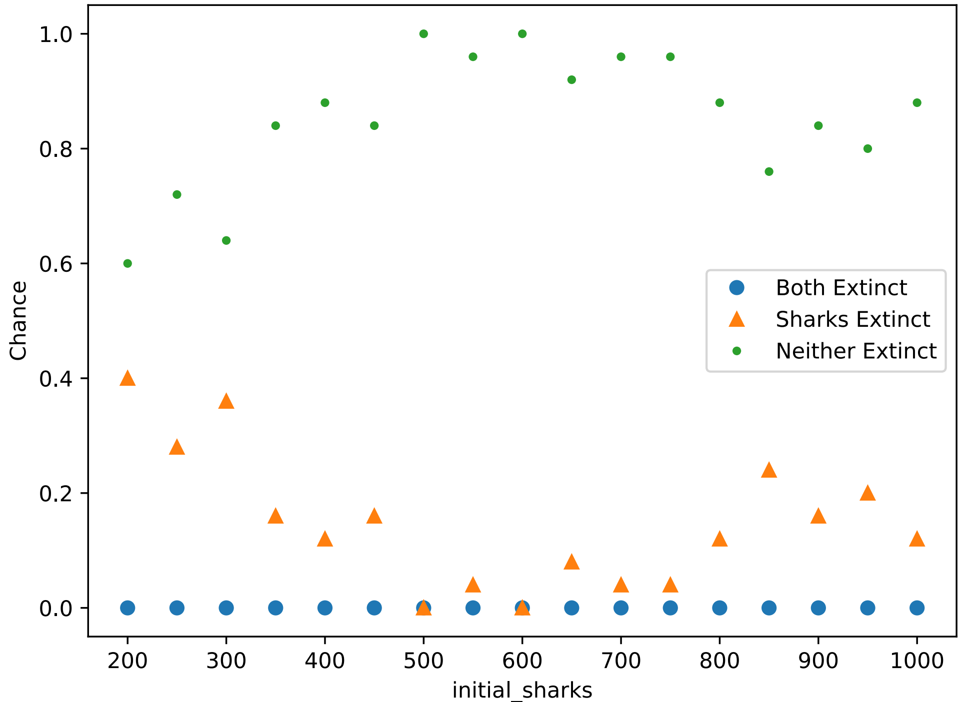


Figure 7: Outcome Chances vs `initial_sharks`

```
import test_outcome_chances as tst
import default_parameters

trials = 25
target_parameter = "start_energy"
start_value = 1
end_value = default_parameters.parameters["breed_energy"] - 1

tst.plot_and_test_extinction_chances(f"media/outcome_chances_{target_parameter}.svg",
target_parameter, range(start_value, end_value + 1), trials)
```

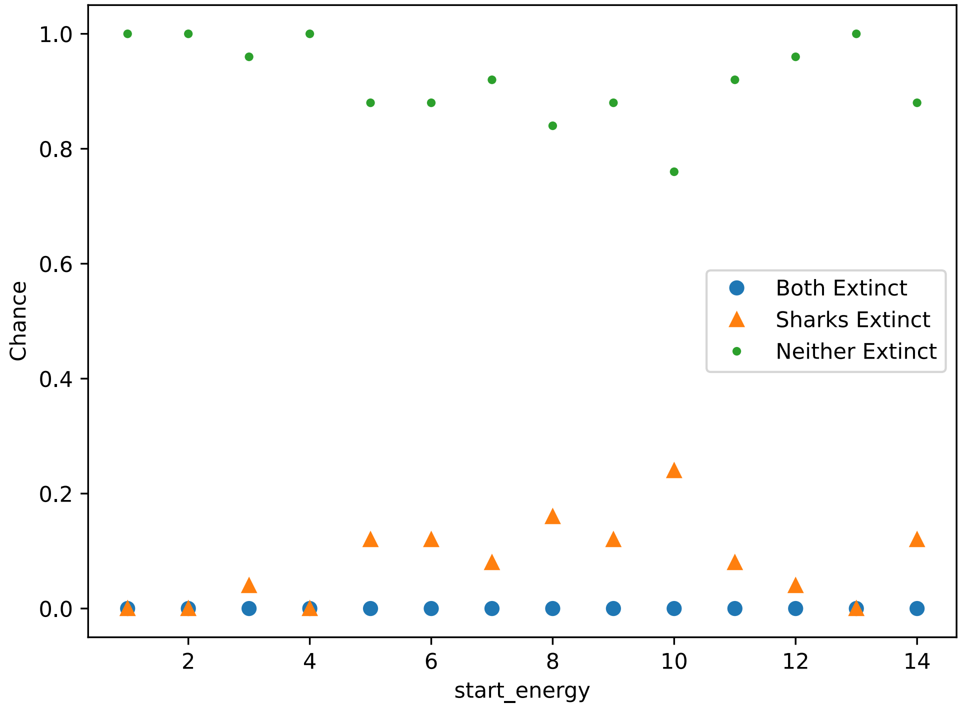


Figure 8: Outcome Chances vs `start_energy`

## Main Simulation Parameters

### `breed_time`

### `energy_gain`

### `breed_energy`

## Circular Initialization

## Extension