## SUPPLEMENTARY MATERIALS:
# Predator-Prey Oscillations in a Cellular Automaton of Huffakers Mite Experiment

Haley Zsoldos*† and Isabelle Stepler*†

*Project advisors: Jessica M. Conway† and Timothy Reluga‡*

## SM1. Extinction Trial Data.

**Table SM1**

*24x20 grid size, no posts, -1 predator energy per timestep*

| | Both Ext Imm | Both Ext | Prey Ext | Pred Ext Imm | Pred Ext | No Ext |
|---|---|---|---|---|---|---|
| **1,1,1** | 100 | 0 | 0 | 0 | 0 | 0 |
| **1,1,10** | 89 | 9 | 0 | 0 | 0 | 2 |
| **1,10,1** | 4 | 0 | 0 | 93 | 3 | 0 |
| **1,10,10** | 0 | 10 | 0 | 51 | 6 | 33 |
| **10,1,1** | 0 | 0 | 100 | 0 | 0 | 0 |
| **10,1,10** | 72 | 0 | 28 | 0 | 0 | 0 |
| **10,10,1** | 0 | 0 | 100 | 0 | 0 | 0 |
| **10,10,10** | 36 | 1 | 0 | 62 | 1 | 0 |

†Department of Mathematics, The Pennsylvania State University, State College, PA. (haz5040@psu.edu, ifs5097@psu.edu, jmc90@psu.edu)

‡Department of Mathematics and Department of Biology, The Pennsylvania State University, State College, PA. (tcr2@psu.edu)

*Both authors contributed equally

## SM2. Oscillation Trial Data without Posts.

**Table SM2**

*36x30 grid size, no posts, -1 predator energy per timestep*

| | **Both Ext Imm** | **Both Ext** | **Prey Ext** | **Pred Ext Imm** | **Pred Ext** | **No Ext** |
|---|---|---|---|---|---|---|
| **1,1,1** | 98 | 2 | 0 | 0 | 0 | 0 |
| **1,1,10** | 25 | 11 | 0 | 0 | 0 | 64 |
| **1,10,1** | 0 | 0 | 0 | 100 | 0 | 0 |
| **1,10,10** | 0 | 0 | 0 | 95 | 0 | 5 |
| **10,1,1** | 0 | 0 | 100 | 0 | 0 | 0 |
| **10,1,10** | 86 | 0 | 14 | 0 | 0 | 0 |
| **10,10,1** | 0 | 0 | 100 | 0 | 0 | 0 |
| **10,10,10** | 21 | 6 | 0 | 40 | 6 | 27 |

## SM3. Oscillation Trial Data with Posts.

**Table SM3**

*36x30 grid size, with posts, -1 predator energy per timestep.*

| | Both Ext Imm | Both Ext | Prey Ext | Pred Ext Imm | Pred Ext | No Ext |
|---|---|---|---|---|---|---|
| **1,1,1** | 2 | 0 | 0 | 0 | 0 | 98 |
| **1,1,10** | 49 | 46 | 0 | 0 | 0 | 5 |
| **1,10,1** | 0 | 0 | 0 | 74 | 22 | 4 |
| **1,10,10** | 61 | 0 | 0 | 39 | 0 | 0 |
| **10,1,1** | 0 | 0 | 100 | 0 | 0 | 0 |
| **10,1,10** | 57 | 0 | 43 | 0 | 0 | 0 |
| **10,10,1** | 0 | 0 | 100 | 0 | 0 | 0 |
| **10,10,10** | 60 | 0 | 0 | 40 | 0 | 0 |

**Table SM4**

*36x30 grid size, with posts, -2 predator energy per timestep.* ***Total oscillations: 85***

| | Both Ext Imm | Both Ext | Prey Ext | Pred Ext Imm | Pred Ext | No Ext |
|---|---|---|---|---|---|---|
| **2,2,20** | 15 | 20 | 0 | 0 | 0 | 65 |

## SM4. Python Code.

```
### Modified from http://scipython.com/blog/wa-tor-world/
/Users/haleyzsoldos/PycharmProjects/fishshark/main.py
import random
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
from matplotlib import animation
from matplotlib import colors
import numpy as np

EMPTY = 0
PREY = 1
PREDATOR = 2

# Color the cells for the above states in this order:
colors = ['sandybrown', 'mediumseagreen', 'crimson']
n_bin = 3
cm = LinearSegmentedColormap.from_list(
    'wator_cmap', colors, N=n_bin)

# # Run the simulation for MAX_CHRONONS chronons (time intervals).
MAX_CHRONONS = 500

# parameters
# NOTE: make sure parameters match those in line 300
# NOTE: PREY INITIAL ENERGY IS ARBITRARY
initial_energies = {PREY: 10, PREDATOR: 1}
fertility_thresholds = {PREY: 1, PREDATOR: 10}


class Mite:
    def __init__(self, id, x, y, init_energy, fertility_threshold):
        """Initialize the mite.

        id is an integer identifying the mite.
        x, y is the mite's position in the world grid.
        init_energy is the mite's initial energy: this decreases by 1
            each time the mite moves and if it reaches 0 the mite dies.
        fertility_threshold: each chronon, the mite's fertility increases
            by 1. When it reaches fertility_threshold, the mite reproduces.

        """

        self.id = id
        self.x, self.y = x, y
        self.energy = init_energy
        self.fertility_threshold = fertility_threshold
```

```
        self.fertility = 0
        self.dead = False


class World:
    def __init__(self, width=24, height=20):
        """Initialize (but don't populate) the Wa-Tor world."""

        self.width, self.height = width, height
        self.ncells = width * height
        self.grid = [[EMPTY] * width for y in range(height)]
        self.mites = []

    def spawn_mite(self, mite_id, x, y):
        """Spawn a mite of type ID mite_id at location x,y."""

        mite = Mite(mite_id, x, y,
                        initial_energies[mite_id],
                        fertility_thresholds[mite_id])
        self.mites.append(mite)
        self.grid[y][x] = mite

    def populate_world(self, nprey=120, npredators=27):
        """Populate the Wa-Tor world with prey and predators."""

        self.nprey, self.npredators = nprey, npredators

        def place_mites(nmites, mite_id):
            """Place nmites of type ID mite_id in the Wa-Tor world."""

            for i in range(nmites):
                while True:
                    x, y = divmod(random.randrange(self.ncells), self.height)
                    if not self.grid[y][x]:
                        self.spawn_mite(mite_id, x, y)
                        break
        place_mites(self.nprey, PREY)
        place_mites(self.npredators, PREDATOR)

    def get_world_image_array(self):
        """Return a 2D array of mite type IDs from the world grid."""
        return [[self.grid[y][x].id if self.grid[y][x] else 0
                 for x in range(self.width)] for y in range(self.height)]

    def get_world_image(self):
        """Create a Matplotlib figure plotting the world."""

        im = self.get_world_image_array()
        fig = plt.figure(figsize=(8.3333, 6.25), dpi=72)
```

```
        ax = fig.add_subplot(111)
        ax.imshow(im, interpolation='nearest', cmap=cm)
        # Remove ticks, border, axis frame, etc
        ax.set_xticks([])
        ax.set_yticks([])
        ax.axis('off')
        return fig

    def show_world(self):
        """Show the world as a Matplotlib image."""

        fig = self.get_world_image()
        plt.show()
        # plt.close(fig)

    def save_world(self, filename):
        """Save a Matplotlib image of the world as filename."""

        fig = self.get_world_image()
        # NB Ensure there's no padding around the image plot
        plt.savefig(filename, bbox_inches='tight', dpi=72, pad_inches=0)
        plt.close(fig)

    def get_neighbours_prey(self, x, y):
        """Return a dictionary of the contents of cells neighbouring (x,y).

        The dictionary is keyed by the neighbour cell's position and contains
        either EMPTY or the instance of the prey mite occupying that cell.

        """
        neighbours = {}
        # post_pos = [(4, 7), (4, 16), (4, 25), (7, 4), (7, 13), (7, 22),
        #             (16, 7), (16, 16), (16, 25), (19, 4), (19, 13), (19, 22),
        #             (28, 7), (28, 16), (28, 25), (31, 4), (31, 13), (31, 22)]

        for dx, dy in ((0, -1), (1, 0), (0, 1), (-1, 0)):
            xp, yp = (x + dx) % self.width, (y + dy) % self.height
            # If the mite is on a boundary and trying to move to the
            # other side of the grid, prevent this from happening
            if (x == 0) and (xp == self.width - 1):
                continue
            elif (xp == 0) and (x == self.width - 1):
                continue
            elif (y == 0) and (yp == self.height - 1):
                continue
            elif (yp == 0) and (y == self.height - 1):
                continue
            else:
                neighbours[xp, yp] = self.grid[yp][xp]
```

```
        # Posts
        # if (x, y) in post_pos:
        #     temp = post_pos[:]
        #     temp.remove((x, y))
        #     for (x_pos, y_pos) in temp:
        #         neighbours[x_pos, y_pos] = self.grid[y_pos][x_pos]
        # else:
        #     for (x_pos, y_pos) in post_pos:
        #         neighbours[x_pos, y_pos] = self.grid[y_pos][x_pos]
        return neighbours

    def get_neighbours_predator(self, x, y):
        """Return a dictionary of the contents of cells neighbouring (x,y).

        The dictionary is keyed by the neighbour cell's position and contains
        either EMPTY or the instance of the predator mite occupying that cell.

        """
        neighbours = {}
        for dx, dy in ((0, -1), (1, 0), (0, 1), (-1, 0)):
            xp, yp = (x + dx) % self.width, (y + dy) % self.height
            # If the mite is on a boundary and trying to move to the
            # other side of the grid, prevent this from happening
            if (x == 0) and (xp == self.width - 1):
                continue
            elif (xp == 0) and (x == self.width - 1):
                continue
            elif (y == 0) and (yp == self.height - 1):
                continue
            elif (yp == 0) and (y == self.height - 1):
                continue
            else:
                neighbours[xp, yp] = self.grid[yp][xp]
        return neighbours

    def evolve_mite(self, mite):
        """Evolve a given mite forward in time by one chronon."""
        if mite.id == PREY:
            neighbours = self.get_neighbours_prey(mite.x, mite.y)
        elif mite.id == PREDATOR:
            neighbours = self.get_neighbours_predator(mite.x, mite.y)

        mite.fertility += 1
        moved = False
        if mite.id == PREDATOR:
            try:
                # Try to pick a random prey to eat.
                xp, yp = random.choice([pos
                                        for pos in neighbours if neighbours[pos] !=
```

```
                                        EMPTY
                                    and neighbours[pos].id == PREY])
                # Eat the prey. Yum yum.
                mite.energy += 1
                self.grid[yp][xp].dead = True
                self.grid[yp][xp] = EMPTY
                moved = True
            except IndexError:
                # No prey to eat: just move to a vacant cell if possible.
                pass

        if not moved:
            # Try to move to a vacant cell
            try:
                xp, yp = random.choice([pos
                                    for pos in neighbours if neighbours[pos] ==
                                        EMPTY])
                if mite.id != PREY:
                    # The predator's energy decreases when it moves.
                    mite.energy -= 1
                moved = True
            except IndexError:
                # Surrounding cells are all full: no movement.
                xp, yp = mite.x, mite.y

        if mite.energy <= 0:
            # Mite dies.
            mite.dead = True
            self.grid[mite.y][mite.x] = EMPTY
        elif moved:
            # Remember the mite's old position.
            x, y = mite.x, mite.y
            # Set new position
            mite.x, mite.y = xp, yp
            self.grid[yp][xp] = mite
            if mite.fertility >= mite.fertility_threshold:
                # Spawn a new mite and reset fertility.
                mite.fertility = 0
                self.spawn_mite(mite.id, x, y)
            else:
                # Leave the old cell vacant.
                self.grid[y][x] = EMPTY

    def evolve_world(self):
        """Evolve the world forward in time by one chronon."""

        # Shuffle the mites grid so that we don't always evolve the same
        # mites first.
        random.shuffle(self.mites)
```

```

        # NB The self.mites list is going to grow as new mites are
        # spawned, so loop over indices into the list as it stands now.
        nmites = len(self.mites)
        for i in range(nmites):
            mite = self.mites[i]
            if mite.dead:
                # This mite has been eaten so skip it.
                continue
            self.evolve_mite(mite)

        # Remove the dead mites
        self.mites = [mite for mite in self.mites
                      if not mite.dead]


# advance the world 1 time step
tots = []


def advance():
    world.evolve_world()
    X = world.get_world_image_array()
    tots.append([sum([i.count(j) for i in X]) for j in range(3)])
    return X

# %% Comment out below to run without animation
# The animation function: called to produce a frame for each generation.
def animate(i):
    if i <= MAX_CHRONONS:
        im.set_data(animate.X)
        animate.X = advance()


# Initialize
world = World()
world.populate_world()

# Animation time. First plot IC
X = world.get_world_image_array()
fig = plt.figure(figsize=(25 / 3, 6.25))
ax = fig.add_subplot(111)
ax.set_axis_off()
im = ax.imshow(X, cmap=cm) # , interpolation='nearest'

# Bind our grid to the identifier X in the animate function's namespace.
animate.X = X

# Interval between frames (ms).
```

```
interval = 100
anim = animation.FuncAnimation(fig, animate, frames=MAX_CHRONONS, interval=interval)
# plt.show()

anim.save('WaTor.gif', fps=15)

# %%

# Run the simulation for MAX_CHRONONS chronons (time intervals) - INTEGER
MAX_CHRONONS = 500

# parameters
# NOTE: PREY INITIAL ENERGY IS ARBITRARY
initial_energies = {PREY: 10, PREDATOR: 1}
fertility_thresholds = {PREY: 1, PREDATOR: 10}

# For loop intended for multiple test runs, in this case 100
# Comment out for loop and i variable if intent is to only run once
# for i in range(1, 101):
world = World()
world.populate_world()
X = world.get_world_image_array()
tots = []

# and run it
for jj in range(0, MAX_CHRONONS):
    advance()

# %%#####################################################
# now plot the predator-prey dynamics
atots = np.array(tots)
fig = plt.figure(figsize=(25 / 3, 6.25))
plt.plot(atots[:, 1], 'darkorange', linewidth=3, label='Prey')
plt.plot(atots[:, 2], '--k', linewidth=3, label='Predators')

plt.xlabel('Number of Timesteps', fontsize=24)
plt.ylabel('Pop. sizes', fontsize=24)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.axis([0, 500, 0, 1200])
plt.legend(loc=1, fontsize=20)
plt.tight_layout()
# Change file path to desired file you would like the graphs saved to
plt.savefig('test2.png', dpi=300) # comment for for loop
# plt.savefig('test_10_10_10_10/TimeSeries%d.png' % i) # uncomment for for loop
plt.close(fig)
# plt.show()
```