SASTA OLIPHANT SCIENCE AWARDS

SOUTH AUSTRALIAN SCIENCE TEACHERS ASSOCIATION

Prize Winner

Computer Programming, Apps & Robotics

Year 7-8

Luke Mulders

Concordia College

Government of South Australia
Department for Education

SASTA South Australian Science Teachers Association

ROWE SCIENTIFIC PTY. LTD. www.rowe.com.au

Simulating Ecosystems

By Luke Mulders

Scientific purpose

Many ecosystems are dying due to imbalance in biomass. This imbalance can be caused by numerous external factors, many of which are difficult for ecologists to identify. Even once the problem is identified, the solution can take a prolonged time to create and execute. This lengthy process rarely has a positive effect and can even damage the ecosystem further if the solution is not well planned.

Simulating Ecosystems can help predict and identify such external factors, even before the factors have had an effect. Ecologists can also simulate proposed solutions in the sandboxed environment before deploying them. This ensures helpful solutions will be deployed promptly, retaining the ecosystem's balance.

A potential application of *Simulating Ecosystems* is a solution to overgrazing. The program can be used to determine the creatures that are overgrazing, the creatures that are affected, and the acceptable biomass. Once these factors are identified, solutions (such as introducing a primary consumer) can be simulated before being deployed.

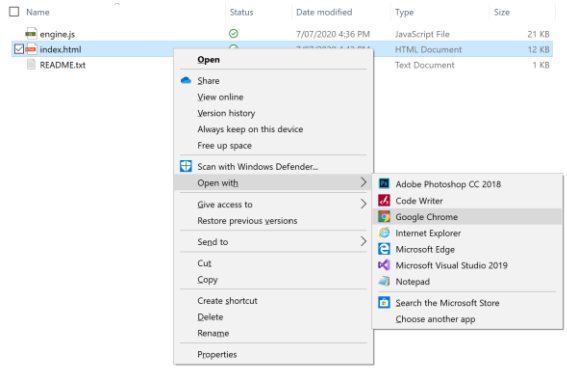
This program displays the first law of ecology. “The environment makes up a huge, enormously complex living machine that forms a thin layer on the earth’s surface.” (Commoner, 1971)

Using the program

Running the program

This program runs on Google Chrome version 71 and above.

1. Ensure the latest version of Google Chrome is installed. If it isn’t, download it from https://www.google.com/intl/en_au/chrome/.
2. Right-click the file named ‘index.html’.
3. Hover over ‘Open with’ in the dropdown menu.
4. Click Google Chrome.
5. Ensure the Google Chrome window has a width of at least 1000 pixels and a height of at least 500 pixels. Otherwise, resize the window by dragging the edges.

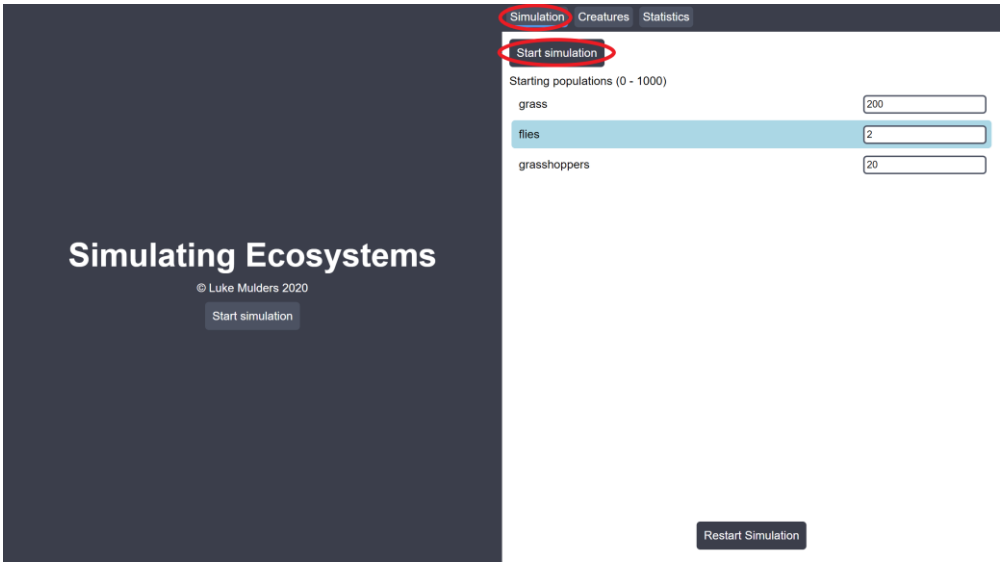


Running the program

Using the program

Running the simulation

1. Click the first tab at the top of the screen that says, ‘Simulation’.
2. Click ‘Start simulation’



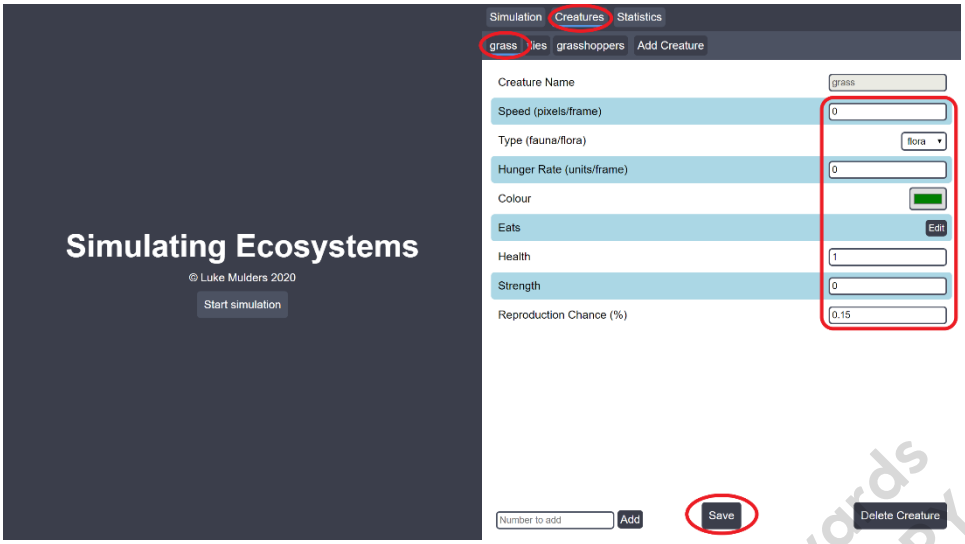
Pausing the simulation

1. Click the tab at the top of the screen that says, ‘Simulation’.
2. Click ‘Pause simulation’ (Note: this button only appears if the simulation is running, otherwise a ‘Start simulation’ button appears.

Editing creatures

1. Click the tab at the top of the screen that says, ‘Creatures’.
2. Click the tab in the second bar that has the name of the creature that is to be edited.
3. Click the input box next to the property to be edited.
4. If the box has a dropdown, select an item from the dropdown. If the box says ‘Edit’ and shows another window, tick the boxes to modify and click ‘Enter’. If the box shows a window that says, ‘Choose colour’, follow the prompts in that window. If the box has a number, input a number between 0 and 1000.
5. Click ‘Save’.

6. All descendants of that creature will inherit the new properties, similar to a genetic mutation. (Note: Only the descendant's properties will be changed, the already existing creatures will not be changed)

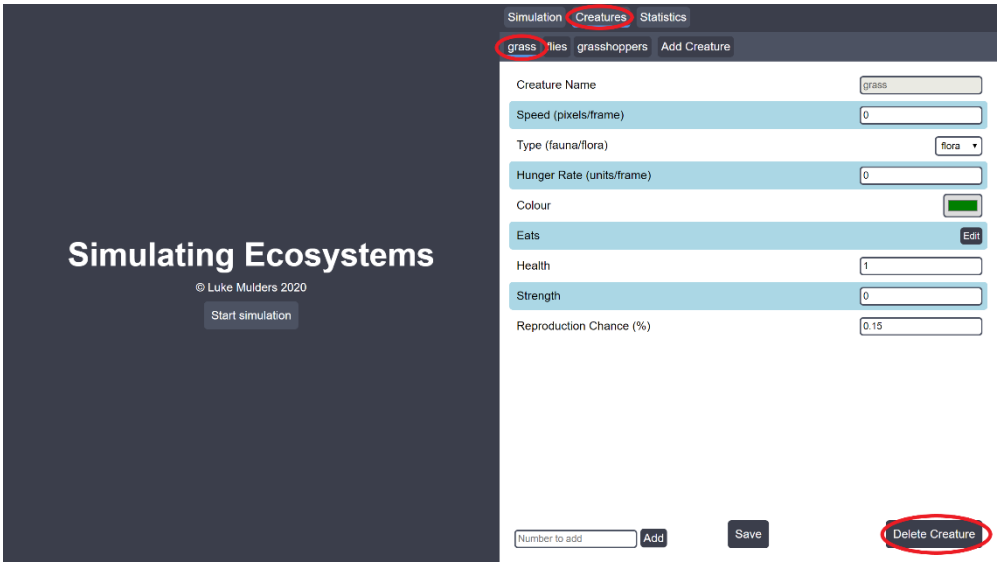


Adding creatures

1. Click the tab at the top of the screen that says, 'Creatures'.
2. Click the button that says 'Add Creature' on the lower bar.
3. Click the white input box.
4. Enter a creature name.
5. Click enter.

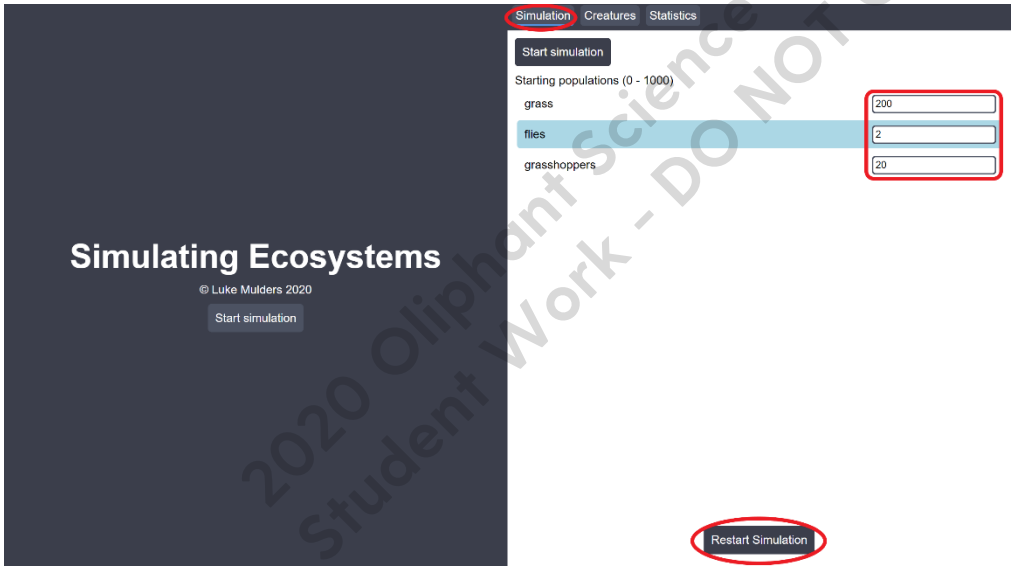
Removing creatures

1. Click the tab at the top of the screen that says, 'Creatures'.
2. Click the tab in the second bar that has the name of the creature that is to be deleted.
3. Click the 'Delete Creature' button in the bottom right corner of the screen.



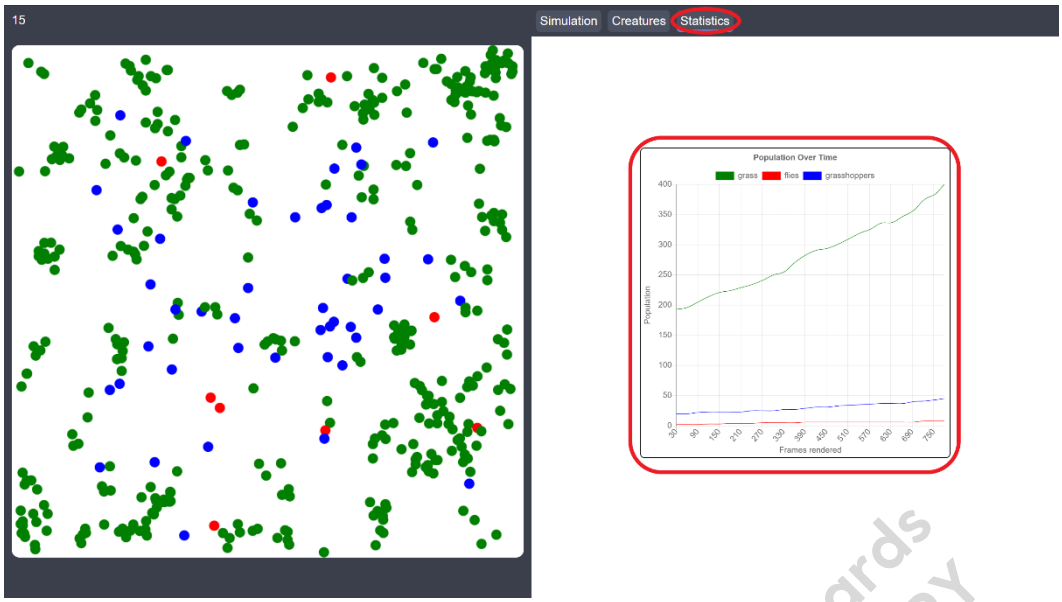
Editing biomass

1. Click the tab at the top of the screen that says, ‘Simulation’.
2. Click the input box next to the name of the creature whose biomass is to be edited.
3. Click ‘Restart Simulation’.



Viewing statistics

1. Click the tab at the top of the screen that says, ‘Statistics’.



Code explanation

Files

Simulating Ecosystems uses three files: index.html, engine.js and chart.js. Index.html contains the markup code responsible for the user interface. Engine.js contains the backend JavaScript code responsible for making the user interface interactive and calculating the creature's behaviour. Chart.js contains the JavaScript code for graphing the data in the 'Statistics' tab. The chart.js code is an external library licenced under MIT available at https://www.chartjs.org/.

Sections

There are four main sets of functions in the engine.js code: initialisation, user interface interaction, creature modification and rendering/calculations.

The initialisation section defines global variables and set initial values for variables.

The user interface section contains functions such as changeTab, setCanvasStyle, showTabCre. The changeTab function swaps between the Simulation, Creatures and Statistics tabs. The setCanvasStyle function resizes the area where the dots are drawn when the window is resized. The showTabCre function switches the creature that is being edited when in the 'Creatures' tab.

The creature modification section has five main functions: addCreature, updateCreature, deleteCreature, deleteCreatureCode and restartSimulation. The addCreature function creates a creature and initialises the creature's properties. The updateCreature function changes the creature properties. The deleteCreature function tells the program to pause next frame and call the deleteCreatureCode function, this is because removing creatures while the main render function is executing can cause confusing semantics. The deleteCreatureCode function hides the creature from the editing lists and deletes all copies of it in the simulation but saves its properties, so the creature can be recovered using addCreature.

The rendering/calculations section has one long function called 'main'. This function draws and calculates the behaviour of the creatures. The function draws the dots on the left half of the screen,

remembers how many of each type of creature there is, moves the creatures, applies hunger, simulates fights between creatures, simulates reproduction and updates the chart.

Annotations

Annotations are formatted in parenthesis and underlined.

Index.html

```
<!DOCTYPE html> (tells the browser it is a HTML document)
<html lang="en">
<head>
        <title>Simulating Ecosystems</title> (tells the browser the title)
  <style> (defines the colours and layout)
   #c {(style for the left half of the screen where the creatures move)
      height: auto;
      width: calc(100% - 20px);
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
      border-radius: 10px;
   }
   canvas {(style for the left half of the screen where the creatures move)
      background-color: white !important;
   }
   body, html {(change the background colour and font)
      margin: 0px;
      font-family: sans-serif;
      height: 100%;
      background-color: #383c4a;
   }
   #fpsCount {(show how many frames per second are rendered in the top corner of the screen)
      top: 10px;
      left: 10px;
      position: absolute;
      color: white;
      z-index: 10;
   }
   #populationChart {(show the population graph)
     height: 400px !important;
     width: 400px !important;
   }
   #rightBar {(show the bar for creature editing on the right side of the screen)
      top: 0px;
      right: 0px;
      width: 50%;
      position: absolute;
      height: 100%;
      overflow: auto;
      background-color: white;
   }
   #rightBar canvas { (center the graph in the statistics tab)
```

```
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    position: absolute;
    display: inherit !important;
    border-radius: 5px;
    border: 1px solid black;
}
#tabs {(style the creature, simulation and statistics tabs)
    background-color: #383c4a;
    padding: 4px;
}
.tab {{(style the creature, simulation and statistics tabs)
    background-color: #4b5162;
    color: white;
    display: inline-block;
    padding: 5px;
    margin: 2px;
    border-radius: 5px;
    cursor: pointer;
}
.selected {(have an underline for the current tab)
    border-bottom: 3px solid #5294e2;
    padding-bottom: 2px !important;
}
.item { (hide tabs that aren't open)
    display: none;
    width: 100%;
    height: calc(100% - 45px);
}
#secondBar { (style the bar that has the creature list)
    background-color: #4b5162;
    padding: 4px;
    overflow: auto;
}
#creatureSelect > span, #addCreature { (style the add creature button)
    background-color: #383c4a;
    color: white;
    border-radius: 5px;
    margin: 2px;
    display: inline-block;
    padding: 5px;
    cursor: pointer;
}
input, select { (style the input boxes)
    border-radius: 5px;
    outline: 0px !important;
    border: 2px solid #4b5162;
    padding: 3px;
}
.list > div { (style the lists of properties)
```

```
    margin: 3px;
    padding: 10px;
    border-radius: 5px;
}
.list input, .list select { (style the lists of properties)
    position: absolute;
    right: 20px;
    transform: translateY(-3px);
}
.list input[type=checkbox] { (override input styling for the checkboxes)
    transform: none;
}
.list > div:nth-child(even) { (make every second line in lists lightblue)
  background-color: lightblue;
}
.list { (more list styling)
    padding: 10px;
}
.button { (button styling)
    position: fixed;
    bottom: 20px;
    left: 75%;
    transform: translateX(-50%);
}
.buttonInline, .button { (button styling)
    display: inline-block;
    border-radius: 5px;
    color: white;
    background-color: #383c4a;
    padding: 10px;
    cursor: pointer;
}
#creatureColor { (colour input styling)
    transform: translateY(-7px) !important;
}
#addCreature, #creatureSelect {(add creature button styling)
    display: inline-block;
}
#addWrap {(styling for the add x creatures button)
    position: fixed;
    left: calc(50% + 20px);
    bottom: 20px;
}
#addBtn { (styling for the button to add x creatures)
    border-radius: 5px;
    color: white;
    background-color: #383c4a;
    padding: 4px;
    cursor: pointer;
}
#pause {(styling for the pause button)
```

```
    display: none;
}
#startScreen { (styling for the title screen)
    height: 100%;
    width: 50%;
    top: 0px;
    position: absolute;
    z-index: 40;
    background-color: #383c4a;
}
#startScreen * { (styling for the title screen)
   display: block;
   text-align: center;
   width: 100%;
}
h1 { (styling for the title screen)
   font-size: 45px;
   margin: 0px 0px 10px 0px;
}
.center { (styling for centring things)
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    color: white;
    text-align: center;
}
#delBtn {(styling for delete button)
    position: fixed;
    bottom: 20px;
    right: 20px;
}
#nameBox, #eatsBox { (styling for the box that pops up when creating a creature)
    background-color: #383c4a;
    width: 500px;
    height: 300px;
    border-radius: 10px;
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    z-index: 50;
    padding: 10px;
    display: none;
    color: white;
}
.heading { (style for headings)
    text-align: center;
    font-weight: bold;
    font-size: 20px;
}
#cWrap { (style for canvas)
```

```
        overflow: auto;
        width: 50%;
        height: calc(100% - 4px);
        position: relative;
        display: inline-block;
        background-color: #383c4a;
    }
    #nameInError { (error message styling)
        display: none;
        color: red;
        margin-bottom: 5px;
    }
    #nameWrap, #eatsWrap { (centring the name and eats box)
        text-align: center;
        position: absolute;
        left: 50%;
        top: 50%;
        transform: translate(-50%, -50%);
    }
    #eatsWrap div:nth-child(even) { (making every second box a different colour)
        background-color: #4b5162;
    }
    #eatsWrap { (setting the dimensions of the eatsWrap box)
        overflow: auto;
        height: 170px;
        width: 424px;
    }
    #creatureEats { (setting the dimensions of the eatsWrap box)
        position: absolute;
        right: 20px;
        padding: 4px;
        font-size: 13px;
        transform: translateY(-2px);
    }
    </style>
</head>
<body>
    <div id="fpsCount"></div> (fps count container)
    <div id="cWrap">
        <canvas id="c" width="1000" height="1000"></canvas>(area for drawing dots)
    </div>
    <div id="nameBox"> (creature name input)
    <div class="heading">Creature name</div>
        <div id="nameWrap">
            <div id="nameInError">Please enter a name</div>
            <input id="nameIn" />
        </div>
        <span class="button" style="background-color: #4b5162; left: 50%;"
onclick="addCreature()">Enter</span>
    </div>
    <div id="eatsBox">(what the creature eats input)
```

```
      <div class="heading">Eats</div>
      <div id="eatsWrap" class="list" style="text-align: left;">
         <div>grass <input data="grass" class="eatsCheck" type="checkbox" /></div>
         <div>flies <input data="flies" class="eatsCheck" type="checkbox" /></div>
         <div>grasshoppers <input data="grasshoppers" class="eatsCheck" type="checkbox" /></div>
      </div>
      <span class="button" style="background-color: #4b5162; left: 50%;"
onclick="this.parentElement.style.display='none'">Enter</span>
   </div>
   <div id="rightBar">
      <div id="tabs">(tabs)
         <div class="tab  selected" onclick="changeTab(0)">Simulation</div>
         <div class="tab" onclick="changeTab(1)">Creatures</div>
         <div class="tab" onclick="changeTab(2)">Statistics</div>
      </div>
      <div class="item" style="display: inline-block">
         <div style="margin: 10px 0px 0px 10px">
            <span class="buttonInline" id="start"
onmouseup="document.getElementById('restartBtn').onclick = function(){restartSimulation()};
restartSimulation()" onclick="this.onmouseup=null;
document.getElementById('startScreen').style.display='none'; running=true; main();
this.style.display='none'; document.getElementById('pause').style.display='inline-block';">Start
simulation</span>
            <span class="buttonInline" id="pause" onclick="running=false; this.style.display='none';
document.getElementById('start').style.display='inline-block';">Pause simulation</span>
         </div>
         <div class="list" id="startPopList">(starting populations/biomass)
            Starting populations (0 - 1000)
            <div>grass <input type="number" class="popInput" data="grass" value="200" /></div>
            <div>flies <input type="number" class="popInput" data="flies" value="2" /></div>
            <div>grasshoppers <input type="number" class="popInput" data="grasshoppers"
value="20"/></div>
         </div>
         <div class="button" id="restartBtn" onclick="restartSimulation(true);">Restart
Simulation</div>(restart simulation)
      </div>
      <div class="item" id="creatureEdit">
         <div id="secondBar">
            <div id="creatureSelect"></div>
            <div id="addCreature" onclick="showNameBox();">Add Creature</div>
         </div>
         <div class="list" id="creEditList">(creature editing)
            <div>Creature Name <input id="creatureName" disabled /></div>
            <div>Speed (pixels/frame) <input class="needsValue" id="creatureSpeed" type="number"
min="0" max="1000" step="any"/></div>
            <div>Type (fauna/flora) <select id="creatureType">
               <option value="flora">flora</option>
               <option value="fauna">fauna</option>
            </select></div>
            <div>Hunger Rate (units/frame) <input class="needsValue" id="creatureHunger"
type="number" min="0" max="1000" step="any"/></div>
```

```
            <div>Colour <input id="creatureColor" type="color" /></div>
            <div>Eats <span onclick="document.getElementById('eatsBox').style.display='block'"
min="0" max="1000" class="buttonInline" id="creatureEats">Edit</span></div>
            <div>Health <input class="needsValue" id="creatureHealth" type="number" step="any"
min="0" max="1000" /></div>
            <div>Strength <input class="needsValue" id="creatureStrength" type="number" step="any"
min="0" max="1000" /></div>
            <div>Reproduction Chance (%) <input class="needsValue"
id="creatureReproductionChance" type="number" min="0" max="1000" step="any"/></div>
        </div>
        <span id="saveBtn" class="button" onclick="updateCreature()">Save</span>
        <div id="addWrap">
            <input id="addNumber" placeholder="Number to add" />
            <span id="addBtn" onclick="for(g=0;
g<parseInt(document.getElementById('addNumber').value); g++) {
            createCreature(creTabName, {x: Math.random()*c.width, y: Math.random()*c.height, food:
10});}">Add
            </span>
        </div>
        <div class="buttonInline" id="delBtn" onclick="deleteCreature()">Delete Creature</div>
      </div>
      <div class="item">
        <canvas id="populationChart" width="400" height="400"></canvas>
      </div>
    </div>
    <div id="startScreen">
      <div class="center">
        <h1>Simulating Ecosystems</h1>
        <h7 style="margin-bottom: 10px">&amp;copy; Luke Mulders 2020</h7>
        <div class="wrap">
            <span class="buttonInline" id="start" style="width: auto; display: inline-block; background-
color: #4b5162" onclick="document.getElementById('restartBtn').onclick =
function(){restartSimulation()}; document.getElementById('start').onmouseup=null;
restartSimulation(); running=true; main(); document.getElementById('start').style.display='none';
document.getElementById('startScreen').style.display='none';
document.getElementById('pause').style.display='inline-block';">Start simulation</span>
        </div>
      </div>
    </div>
    <script src="./chart.js"></script>
    <script src="./engine.js"></script>
</body>
</html>
```


Engine.js

(initialise variables)

```
const c = document.getElementById("c");
const ctx = c.getContext("2d");
var creatureRef = {
    "flies": {name:"flies", hunger: 0.005, speed: 2, color: "#ff0000", eat: ["grasshoppers"], health: 30,
strength: 10, type: "fauna", reproductionChance: 0.12, food: Math.floor(Math.random()*5) + 10},
```

```
   "grasshoppers": {name:"grasshoppers", hunger: 0.005, speed: 2, color: "#0000ff", eat: ["grass"],
health: 30, strength: 5, type: "fauna", reproductionChance: 0.15, food:
Math.floor(Math.random()*5) + 10},
   "grass": {name:"grass", hunger: 0, speed: 0, color: "#008000", eat: [], health: 1, strength: 0, type:
"flora", reproductionChance: 0.15}//0.1575
};
(creature object required properties: name, x, y, speed, type, hunger, colour, eat, health, strength,
food)
var creatures = [];
var i;
var f;
var g;
var h;
var prevCreatures;
var newCreature;
var closest;
var frame = 0;
var newCreatureLoc;
var lastLoop = performance.now();
var thisLoop;
var defaultHealth;
var fps;
var fpsCount = document.getElementById("fpsCount");
var firstAttack;
var creatureList = ["grass", "flies", "grasshoppers"];//HERE
var oldTabCre = 0;
var creTabName = "grass";
var a;
var running = false;
var startingPopulations = {
   grass: 200,
   grasshoppers: 20,
   flies: 2
};
(chartjs setup)
Chart.defaults.global.defaultFontSize = 18;
populationChart = new Chart(document.getElementById("populationChart").getContext("2d"), {
   type: "line",
   data: {
      datasets: [{
         label: "grass",
         data: [],
         fill: false,
         borderWidth: 1,
         pointRadius: 0,
         borderColor: "green"
      }, {
         label: "flies",
         data: [],
         fill: false,
         borderWidth: 1,
```

```
            pointRadius: 0,
            borderColor: "red"
        }, {
            label: "grasshoppers",
            data: [],
            fill: false,
            borderWidth: 1,
            pointRadius: 0,
            borderColor: "blue"
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: true,
        aspectRatio: 1,
        title: {
            display: true,
            text: "Population Over Time"
        },
        scales: {
            xAxes: [{
                scaleLabel: {
                    display: true,
                    labelString: "Frames rendered"
                }
            }],
            yAxes: [{
                scaleLabel: {
                    display: true,
                    labelString: "Population"
                },
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
var oldTab = 0;
function changeTab(tab) {
    (hide old tab)
    document.getElementsByClassName("item")[oldTab].style.display = "none";
    document.getElementsByClassName("tab")[oldTab].className = "tab";
    (show new tab)
    document.getElementsByClassName("item")[tab].style.display = "inline-block";
    document.getElementsByClassName("tab")[tab].className = "tab selected";
    oldTab = tab;
}
(set canvas style)
function setCanvasStyle() {
    if (window.innerWidth/2 < window.innerHeight) {
```

```
      c.style.width = "calc(100% - 20px)";
      c.style.height = "auto";
   } else {
      c.style.width = "auto";
      c.style.height = "calc(100% - 20px)";
   }
}
setCanvasStyle();
window.onresize = setCanvasStyle;
(second bar for creatures)
function showTabCre(tab) {
   (hide old tab)
   document.getElementsByClassName("creTab")[oldTabCre].className = "creTab";
   (show new tab)
   document.getElementsByClassName("creTab")[tab].className = "creTab selected";
   (put value into inputs)
   oldTabCre = tab;
   creTabName = document.getElementsByClassName("creTab")[tab].innerText;
   let eatsCheck = document.getElementsByClassName("eatsCheck");
   for (a=0; a<eatsCheck.length; a++) {
      if (creatureRef[creTabName].eat.indexOf(eatsCheck[a].getAttribute("data")) !== -1) {
         (set checked value)
         eatsCheck[a].checked = true;
      } else {
         eatsCheck[a].checked = false;
      }
   }
(set input values)
   document.getElementById("creatureName").value = creTabName;
   document.getElementById("creatureSpeed").value = creatureRef[creTabName].speed;
   document.getElementById("creatureType").value = creatureRef[creTabName].type;
   document.getElementById("creatureHunger").value = creatureRef[creTabName].hunger;
   document.getElementById("creatureColor").value = creatureRef[creTabName].color;
   document.getElementById("creatureHealth").value = creatureRef[creTabName].health;
   document.getElementById("creatureStrength").value = creatureRef[creTabName].strength;
   document.getElementById("creatureReproductionChance").value =
creatureRef[creTabName].reproductionChance;
}
for (i=0; i<creatureList.length; i++) {
   document.getElementById("creatureSelect").innerHTML += "<span class='creTab'
onclick='showTabCre(" + i + ")'>" + creatureList[i] + "</span>";
}
showTabCre(0);
var creatureCount = {};
(create new creature when a creature reproduces)
function createCreature(name, parent) {
   newCreature = {};
   Object.assign(newCreature, creatureRef[name]);
   newCreature.x = parent.x;
   newCreature.y = parent.y;
   newCreature.food = parent.food;
```

```
  (set x location)
  newCreatureLoc = Math.floor(Math.random() * 30);
  if (Math.random() >= 0.5) newCreatureLoc *= -1;
  newCreature.x += newCreatureLoc;
  (set y location)
  newCreatureLoc = Math.floor(Math.random() * 30);
  if (Math.random() >= 0.5) newCreatureLoc *= -1;
  newCreature.y += newCreatureLoc;
  (add a "no spawn zone" 10px around screen)
  if (newCreature.x - 10 < 0 || newCreature.x + 10 > c.width || newCreature.y - 10 < 0 ||
newCreature.y + 10 > c.height) return;
  (set where the creature is moving to)
  newCreature.moveTo = undefined;
  (vary starting amount of food)
  newCreature.food -= Math.random();
  (push the new creature into list)
  creatures.push(newCreature);
}
var creatureCountNumb = 4;
function showNameBox() {
  document.getElementById("nameBox").style.display = "inline-block";
}
(add creature when a user clicks the button)
function addCreature() {
  let nameElm = document.getElementById("nameIn");
  document.getElementById("nameInError").style.display = "none";
  nameElm.style.borderColor = "#383c4a";
  if (nameElm.value === "" || nameElm.value === undefined || nameElm.value.length === 0) {
    document.getElementById("nameInError").style.display = "block";
    nameElm.style.borderColor = "red";
    return;
  }
  document.getElementById("nameBox").style.display = "none";
  document.getElementById("creEditList").style.display = "block";
  document.getElementById("saveBtn").style.display = "block";
  document.getElementById("delBtn").style.display = "block";
  document.getElementById("addWrap").style.display = "block";
  let elms = document.getElementsByClassName("creTab");
  if (creatureList.indexOf(nameElm.value) !== -1) {
    for (a = 0; a<elms.length; a++) {
      if (elms[a].innerText === nameElm.value) {
        elms[a].style.display = "inline-block";
        document.getElementsByClassName("popInput")[a].parentElement.style.display = "block";
        document.getElementsByClassName("eatsCheck")[a].parentElement.style.display = "block";
        showTabCre(a);
        nameElm.value = "";
        return;
      }
    }
  }
  creatureRef[nameElm.value] = {
```

```
      name: nameElm.value,
      speed: 2,
      type: "fauna",
      hunger: 0.005,
      color: "#383c4a",
      eat: [],
      health: 10,
      strength: 10,
      reproductionChance: 0.1
   };
   creatureList.push(nameElm.value);
   (need seperate var because using '-' in onclick doesnt work)
   let corVar = creatureCountNumb-1;
   document.getElementById("creatureSelect").innerHTML += "<span class='creTab'
onclick='showTabCre(" + corVar + ")'>" + nameElm.value  + "</span>";
   document.getElementById("startPopList").innerHTML += "<div>" + nameElm.value + "<input
type='number' class='popInput' value='1' data='" + nameElm.value  + "' /></div>";
   document.getElementById("eatsWrap").innerHTML += "<div>" + nameElm.value + "<input
type='checkbox' class='eatsCheck' data='" + nameElm.value  + "' /></div>";
   (add to chart)
   populationChart.data.datasets.push({
      label: nameElm.value,
      data: [{x: frame, y: 0}],
      fill: false,
      borderWidth: 1,
      pointRadius: 0,
      borderColor: "#383c4a"
   });
   populationChart.update();
   creatureCountNumb++;
   nameElm.value = "";
}
var toDelete = false;
function deleteCreature() {
   (wait until it stops if already running to avoid endless loops in main())
   if (running) {
      running = false;
      toDelete = true;
   } else {
      deleteCreatureCode();
   }
}
(delete creature)
function deleteCreatureCode() {
   let selectedOne = false;
   for (i=0; i<creatures.length; i++) {
      if (creatures[i].name === creTabName) {
         creatures.splice(i, 1);
         i--;
      }
   }
```

```
    let popInput = document.getElementsByClassName("popInput")[oldTabCre];
    popInput.value = 0;
    popInput.parentElement.style.display = "none";
    document.getElementsByClassName("eatsCheck")[oldTabCre].parentElement.style.display =
"none";
    let tabs = document.getElementsByClassName("creTab");
    tabs[oldTabCre].style.display = "none";
    for (i = 0; i<tabs.length; i++) {
        if (tabs[i].style.display !== "none") {
            selectedOne = true;
            showTabCre(i);
            break;
        }
    }
    if (!selectedOne) {
        document.getElementById("creEditList").style.display = "none";
        document.getElementById("saveBtn").style.display = "none";
        document.getElementById("delBtn").style.display = "none";
        document.getElementById("addWrap").style.display = "none";
    }
}
function restartSimulation(hideStartScreen) {
    let popInput = document.getElementsByClassName("popInput");
    for(h=0;h<popInput.length;h++) {
        popInput[h].value = Math.round(popInput[h].value);
        if (popInput[h].value === "" || popInput[h].value === undefined || isNaN(popInput[h].value) ||
            popInput[h].value.length === 0 || popInput[h].value < 0) {
            popInput[h].value = 0;
        } else if (popInput[h].value > 1000) {
            popInput[h].value = 1000;
        }
        startingPopulations[popInput[h].getAttribute("data")] = parseInt(popInput[h].value);
    }
    creatures = [];
    populationChart.data.labels = [];
    for(h=0;h<populationChart.data.datasets.length; h++) {
        populationChart.data.datasets[h].data=[];
    }
    populationChart.data.labels = [];
    frame = 0;
    for(h=0; h<creatureList.length; h++) {
        for (g = 0; g< startingPopulations[creatureList[h]]; g++) {
            createCreature(creatureList[h], {x: Math.random()*c.width, y: Math.random()*c.height, food:
10});
        }
    }
    if (hideStartScreen) {
        document.getElementById("restartBtn").onclick = function(){restartSimulation();};
        document.getElementById("start").style.display = "none";
        document.getElementById("startScreen").style.display = "none";
        document.getElementById("pause").style.display = "inline-block";
```

```
      running = true;
      main();
   }
}
restartSimulation();
setInterval(function () {
   (update fps counter)
   fpsCount.innerText = fps;
}, 500);
function updateCreature() {
   if (document.getElementById("creatureType").value !== "fauna" &&
document.getElementById("creatureType").value !== "flora") {
      document.getElementById("creatureType").value = creatureRef[creTabName].type;
   }
   let needsValue = document.getElementsByClassName("needsValue");
   for (a=0; a<needsValue.length; a++) {
      if (!needsValue[a].value || parseFloat(needsValue[a].value) < 0) {
         needsValue[a].value = 0;
      } else if (parseFloat(needsValue[a].value) > 1000) {
         needsValue[a].value = 1000;
      }
   }
   let eatsCheck = document.getElementsByClassName("eatsCheck");
   let eats = [];
   for (a=0; a<eatsCheck.length; a++) {
      if (eatsCheck[a].checked) {
         eats.push(eatsCheck[a].getAttribute("data"));
      }
   }
   creatureRef[creTabName] = {
      name: creTabName,
      speed: parseFloat(document.getElementById("creatureSpeed").value),
      type: document.getElementById("creatureType").value,
      hunger: parseFloat(document.getElementById("creatureHunger").value),
      color: document.getElementById("creatureColor").value,
      eat: eats,
      health: parseFloat(document.getElementById("creatureHealth").value),
      strength: parseFloat(document.getElementById("creatureStrength").value),
      reproductionChance:
parseFloat(document.getElementById("creatureReproductionChance").value)
   };
   populationChart.data.datasets[creatureList.indexOf(creTabName)].borderColor =
document.getElementById("creatureColor").value;
   document.getElementsByClassName("creTab")[oldTabCre].innerText =
document.getElementById("creatureName").value;
}
function main() {
   (frame counter)
   frame++;
   (fps counter)
   thisLoop = performance.now();
```

```
  fps = Math.round(1000 / (thisLoop - lastLoop));
  lastLoop = thisLoop;
  (animations)
  if (running) {
    window.requestAnimationFrame(main);
  } else if (toDelete) {
    deleteCreatureCode();
    toDelete = false;
    running = true;
    main();
  }
  (reset creature count for chart)
  creatureCount = {};
  (clear canvas)
  ctx.clearRect(0, 0, c.width, c.height);
  for (i = 0; i< creatures.length; i++) {
    ctx.beginPath();
    ctx.fillStyle = creatures[i].color;
    ctx.arc(creatures[i].x, creatures[i].y, 10, 0, 2 * Math.PI, false);
    ctx.fill();
    ctx.closePath();
    (LOGIC)
    (creature count)
    if (!creatureCount[creatures[i].name]) {
      creatureCount[creatures[i].name] = 0;
    }
    creatureCount[creatures[i].name]++;
    (calculate creature movement)
    if (creatures[i].type === "fauna") {(if statement for optimisation)
      if (creatures[i].food < 8 && creatures[i].moveTo === undefined) {
        (look for food)
        closest = Infinity;
        for (f = 0; f< creatures.length; f++) {
          //dont look at self, check if will eat, check if closest
          if (f !== i && creatures[i].eat.some(x => x === creatures[f].name) &&
        Math.abs(creatures[f].x - creatures[i].x + creatures[f].y - creatures[i].y) < closest &&
        creatures[i].food < 10) {
            closest = Math.abs(creatures[f].x - creatures[i].x + creatures[f].y - creatures[i].y);
            creatures[i].moveTo = creatures[f];
          }
        }
      }
      (move creatures on display)
      if (creatures[i].moveTo !== undefined) {
        (move x)
        if (creatures[i].moveTo.x - creatures[i].x > creatures[i].speed) {
          creatures[i].x += creatures[i].speed;
        } else if (creatures[i].moveTo.x - creatures[i].x < -creatures[i].speed) {
          creatures[i].x -= creatures[i].speed;
        } else {
          creatures[i].x = creatures[i].moveTo.x;
```

```
            }
            (move on y axis)
            if (creatures[i].moveTo.y - creatures[i].y > creatures[i].speed) {
               creatures[i].y += creatures[i].speed;
            } else if (creatures[i].moveTo.y - creatures[i].y < -creatures[i].speed) {
               creatures[i].y -= creatures[i].speed;
            } else {
               creatures[i].y = creatures[i].moveTo.y;
            }
            (detect if the creature reached desired location when wandering)
            if (creatures[i].moveTo && Math.abs(creatures[i].x - creatures[i].moveTo.x) < 4 &&
Math.abs(creatures[i].moveTo.y - creatures[i].moveTo.y) < 4) {
               creatures[i].moveTo = undefined;
            }
         }
         (make the creature wander if it can't see food)
         if (creatures[i].moveTo === undefined) {
            creatures[i].moveTo = {
               x: Math.floor(Math.random() * c.width), y: Math.floor(Math.random() * c.height)
            };
         }
      }
   }
   (uses seperate loop for each death calculation to ensure a creature doesn't loop twice)
   (hunger)
   for (i = 0; i< creatures.length; i++) {
      if (creatures[i].type !== "fauna") continue;
      creatures[i].food -= creatures[i].hunger;
      if (creatures[i].food <= 0) {
         creatures.splice(i, 1);
         i--;
      }
   }
   for (i = 0; i< creatures.length; i++) {
      (attack)
      fightLoop2: for (f = 0; f< creatures.length; f++) {
         if (Math.abs(creatures[i].x - creatures[f].x) < 4 && Math.abs(creatures[i].y - creatures[f].y) < 4
&& i !== f &&
            (creatures[i].eat.some(x => x === creatures[f].name) || creatures[f].eat.some(x => x ===
creatures[i].name))) {
            defaultHealth = [creatures[i].health, creatures[f].health];
            firstAttack = true;
            while (creatures[i] && creatures[f]) {
               (creature 1 attack)
               creatures[f].health -= Math.floor(Math.random() * creatures[i].strength / 3) +
creatures[i].strength / 3 * 2;
               if (creatures[f].health <= 0) {
                  if (creatures[i].eat.some(x => x === creatures[f].name) && creatures[i].food < 8) {
                     creatures[i].food += 3;
                  }
                  creatures[i].moveTo = undefined;
```

```
                creatures[i].health = defaultHealth[0];
                creatures.splice(f, 1);
                break fightLoop2;
              }
              if (Math.random() > 0.5 && firstAttack === true) {
                  (randomly decide who goes first, reset the health of creature 1 if creature 2 is going
                  first)
                creatures[f].health = defaultHealth[1];
              }
              firstAttack = false;
              (creature 2 attack)
              creatures[i].health -= Math.floor(Math.random() * creatures[f].strength / 3) +
creatures[f].strength / 3 * 2;
              if (creatures[i].health <= 0) {
                if (creatures[f].eat.some(x => x === creatures[i].name) && creatures[f].food < 8) {
                  creatures[f].food += 3;
                }
                creatures[f].moveTo = undefined;
                creatures[f].health = defaultHealth[1];
                creatures.splice(i, 1);
                i--;
                break fightLoop2;
              }
            }
          }
        }
    }
    (separate loop to ensure loop finishes)
    prevCreatures = creatures.length;
    for (i = 0; i< prevCreatures; i++) {
      (if a fauna of same species are close, and no overpopulation (50 or more cratures in 100px),
have a chance to reproduce)
      let closeCreatures = 0;
      for(f = 0; f < creatures.length; f++){
        if (Math.abs(creatures[i].x - creatures[f].x) < 100 && Math.abs(creatures[i].y - creatures[f].y) <
100) {
          closeCreatures++;
        }
      }
      if (creatures.some(x => x.name === creatures[i].name && Math.abs(creatures[i].x - x.x) < 100
&& Math.abs(creatures[i].y - x.y) < 100)) {
        if (Math.random() * 100 <= creatures[i].reproductionChance && closeCreatures < 50) {
          (create copy not reference)
          createCreature(creatures[i].name, creatures[i]);
        }
      }
    }
    (chart)
    if (frame % 30 === 0) {
      for (i=0; i<creatureList.length; i++) {
        if (!creatureCount[creatureList[i]]) {
```

```
            creatureCount[creatureList[i]] = 0;
          }
          populationChart.data.datasets[i].data.push({x: frame, y: creatureCount[creatureList[i]]});
        }
        populationChart.data.labels.push(frame);
        populationChart.update();
    }
}
(call main initially)
main();
```

Acknowledgement

Apart from the following files, the program was coded entirely by Luke Mulders.

- Chart.js: ChartJS (Copyright (c) 2018 Chart.js Contributors, licenced under MIT).

Video demonstration

This video shows a standard simulation using this program. This simulation depicts a common ecosystem, where flies eat grasshoppers and grasshoppers eat flies. The flies are secondary consumers, the grasshoppers are primary consumers, and the grass is the primary producer.

During the video, I also created a creature called 'test' to demonstrate creating and deleting creatures.

At the start of the video, I set up the properties. During the simulation, I modified the properties for the grasshoppers to make them stronger and increased their reproduction rate. This meant the flies almost became extinct as they couldn't attack the grasshoppers to get food. To counteract this, I made the flies stronger. The grasshopper population also dramatically increased due to the increased reproduction rate. The grasshopper population started dropping due to overgrazing of grass. While the grasshoppers were weakened, the flies finished the grasshoppers off before they could eat all the grass. This meant the grass recovered, but the flies were left with no food, leading to their extinction. The ecosystem was left with only a small pocket of grass in the corner.

I could have continued to make small adjustments to the simulation to allow the ecosystem to survive for longer, but the video was already over seven minutes long.

- URL: https://vimeo.com/436323952
- Password: Oliphant20

Bibliography

Commoner, B., 1971. *The Closing Circle: Nature, Man, and Technology.* 1st ed. New York City: Penguin Random House.

Conserve Energy Future Editors, 2020. *What is Overgrazing?.* [Online]
Available at: https://www.conserve-energy-future.com/causes-effects-solutions-overgrazing.php
[Accessed 8 July 2020].

The Editors of Encyclopaedia Britannica, 2020. *Biomass.* [Online]
Available at: https://www.britannica.com/science/biomass
[Accessed 8 July 2020].