Criterion C: Development Stage

1. General Overview

Throughout the development stage I made sure to separate my work into frontend and backend files. The front end focuses on the layout, format, color etc. of my webpage. The backend focuses on the simulation itself, managing the environment and all the elements.

In particular, I focused on organizing my code so that the entire program is easy to understand and runs with a clear flow of operations. To ensure extendability, all variables, functions, classes, and files have intuitive names, and I also have appropriate comments throughout the code to explain the purpose of each section.

Because I was dealing with a large population of similar creatures, object oriented programming was a natural fit. I also implemented modular design (such as separating the 4 different actions a rabbit can take) so it will be easy for future developers to edit a specific part of the code.

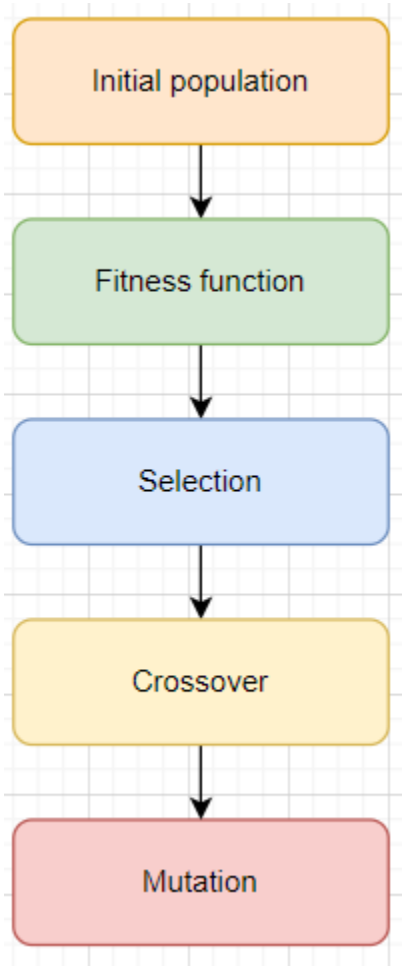
2. Table of Complex Techniques

Program Sections	Complex Techniques
Simulation	Object-oriented programming (OOP) Genetic algorithms List manipulation String slicing Nested loops Modular design
Environment	Tracking time Tracking all items
GUI	Bootstrap Routing JS files Interaction JS CSS
Graph	Chart.js Dataset manipulation Rendering and updating Designating axis scales Color coding

Visualization	P5.js methods Loading images from folder Using sprites Rendering (setup, update framerate)

3. Genetic Algorithm

Every genetic algorithm follows these 5 main steps:



However, since I am simulating a population in real time, these steps do not always occur consecutively in the order shown above. All of these need to be able to happen at any given time, even simultaneously. By using modular design, I separated all of these into individual functions that do not interfere with each other, thus allowing them to run concurrently.

Initial population

Uses a for loop to create “num” amount of rabbits

RandomBinary() generates a random 21bit binary number

```
function addRabbit(num){
    for (let i = 0; i<num;i++){
        rabbits.push(new Rabbit(Math.random() * (970 - 20) + 20,Math.random() * (470 - 20) + 20, RandomBinary()));

    }
}
```

Generates random coordinates within the boundaries of the canvas

Fitness function

If the rabbit’s energy drops below 0, the rabbit dies

```
update(grass){
    if(this.energy<=0){
        this.alive=false
        return
    }
```

The update() function immediately ends if the rabbit dies

Selection

The simulation will only carry out processes if it is running (not paused)

```
if (isrunning){
    rabbits.forEach(function(item,index,object){
        if(item.alive==false){
            object.splice(index,1)
        }
    })
```

Loop through the list of every rabbit

If the rabbit is no longer alive, remove it from the list of rabbits

Crossover

Check if the chosen action is reproduction

The rabbit loses a lot of energy during reproduction

Create a random cutoff point between 0 and 21, which determines where to split the DNAs by.

```
}else if(action[0] == 2){ //reproduce
    this.energy -= 120
    let cutoff = Math.floor(Math.random() * 21)
    let newdna = this.mutate(this.dna.slice(0,cutoff) + action[3].slice(cutoff,24))
    rabbits.push(new Rabbit(action[1],action[2],newdna));
```

mutate() randomly flips a certain number of bits

create a new rabbit in between its parents with the new DNA

Use string slicing to combine the DNA of the two rabbits, joined together at the cutoff point

Mutation

Fetches the mutation rate from the slider

Initializes an empty string to be added to

```
//takes in a binary number and randomly flips a certian number of bits based on the mutation rate
    mutate(dna) {
        let mut_dna = "";
        let mut_rate = document.getElementById("mutation_rate").value;

        for (let i = 0; i < dna.length; i++) {
          if (Math.random() < 0.2 || mut_rate <= 0) {
            mut_dna += dna[i];
          } else {
            mut_dna += dna[i] === "0" ? "1" : "0";
            mut_rate--;
          }
        }

        return mut_dna;
      }
```

Loops for the length of the DNA

decrement the mut_rate so only a specific number of bits are flipped

return the new dna

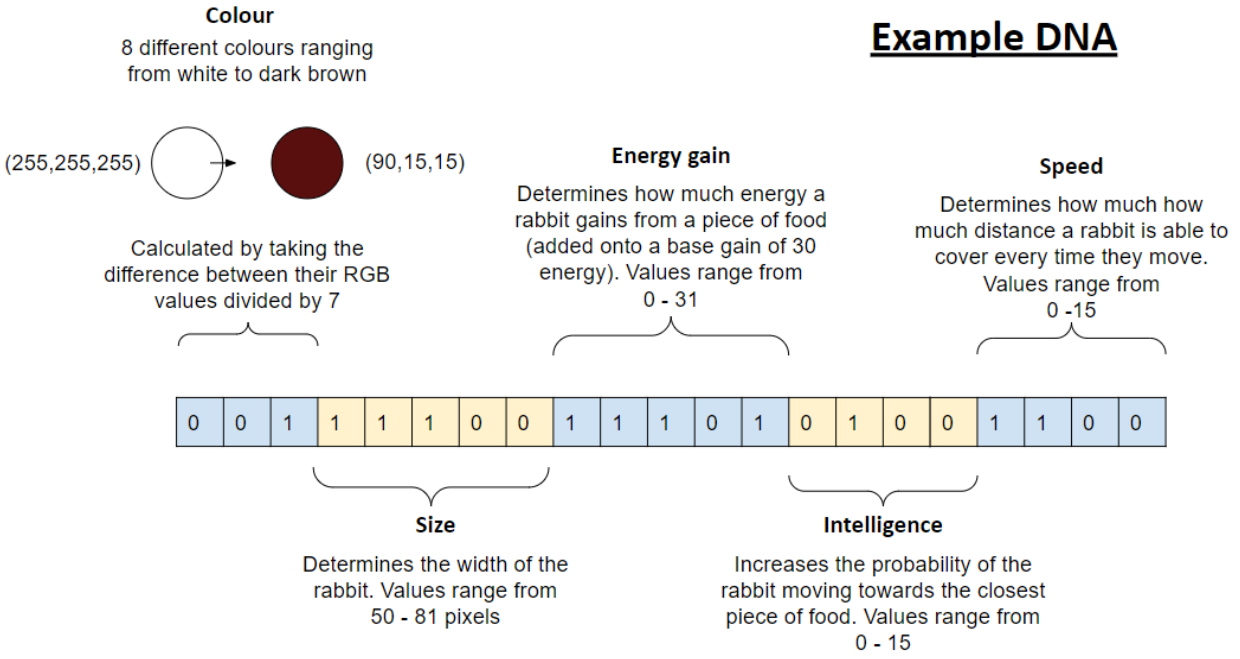
flips the bit

Adds either the original bit or the flipped bit to the string based on random chance

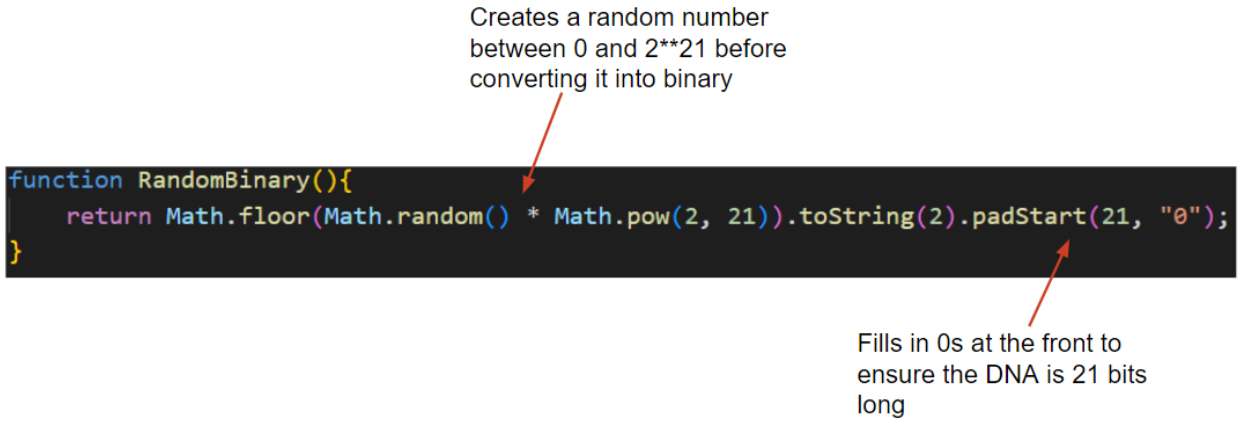
4. Rabbit

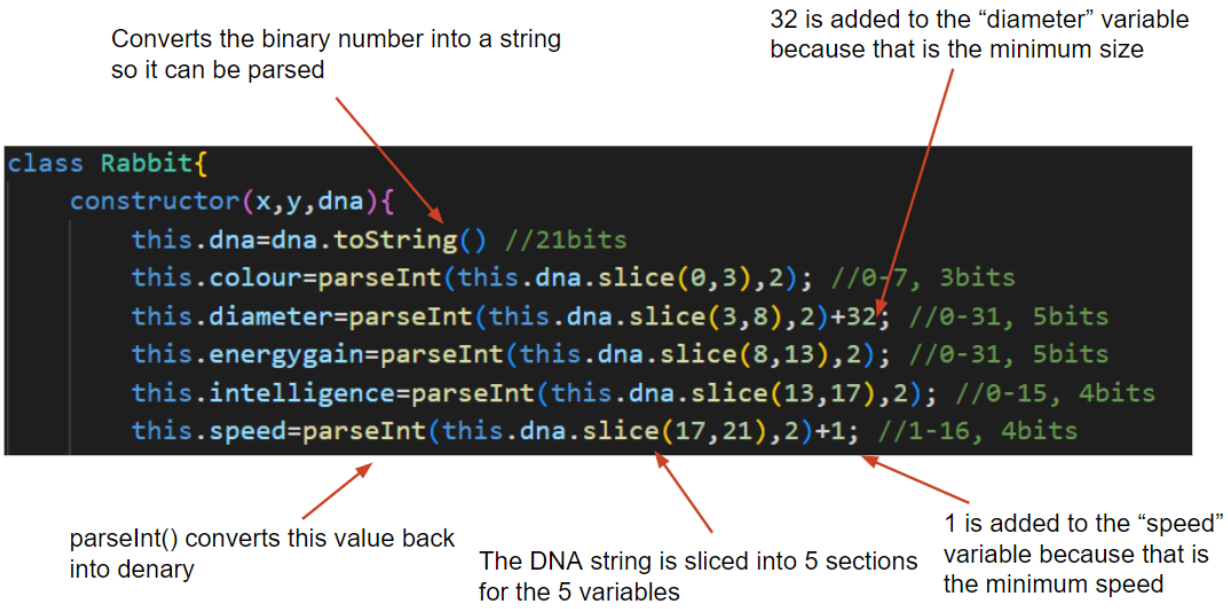
- **DNA**

The DNA of the rabbit is a crucial part that governs how the genetic algorithm works. It will be the centerpiece of how traits are passed on from generation to generation and how families are formed later on. I am also able to identify each rabbit by their DNA.



When initially creating the environment, I need to be able to create random DNA for all the rabbits spawning in to simulate a biodiverse population.





It is important to acknowledge that there are many aspects of my simulation which simplifies how DNA works in the real world. However, the intention of this product is not to perfectly resemble biology, it is to be used as a teaching material to help children grasp the concept of DNA. As long as the offspring’s DNA is a combination of its parents’ DNA, and it has the ability to randomly mutate, that is sufficient for traits to be selected for. A more complicated system would likely prove to be excessive and unnecessarily create sources for bugs to arise.

- **Choose Action**

In my simulation, the rabbit has to be able to carry out several different actions (eating, moving, reproduction). I have written a function named chooseaction() to decide on which action for the rabbit to take, before carrying out the chosen action. The rationale of using modular design here is mainly to reduce the amount of unnecessary processing and improve the extensibility of the product. By separating every action into individual parts, future developers will have an easier time when trying to adjust one specific function. Additionally, this also makes it easier for myself when debugging and running tests.

The 4 possible actions a rabbit can take, in order of priority.

Initialize temporary variables within the function, because you cannot access the rabbit's own variables when under the forEach() loop

```
chooseaction(grass){
    //eat, reproduce, move, choose direction
    let tempx = this.x
    let tempy = this.y
    let tempe = this.energy
    let tempd = this.diameter
    let tempdna = this.dna
    let action = [4]
```

This is the variable the function will return at the end. It is a list because sometimes I may need to return other information alongside the chosen action. It is initialized at 4 so that action 4 (choose direction) is the default action if no other action is chosen.

First, we check if there is food available to eat in the vicinity.

Loop through each piece of grass

Calculate the distance between the rabbit and the grass with the x and y coordinates by using the pythagorean theorem

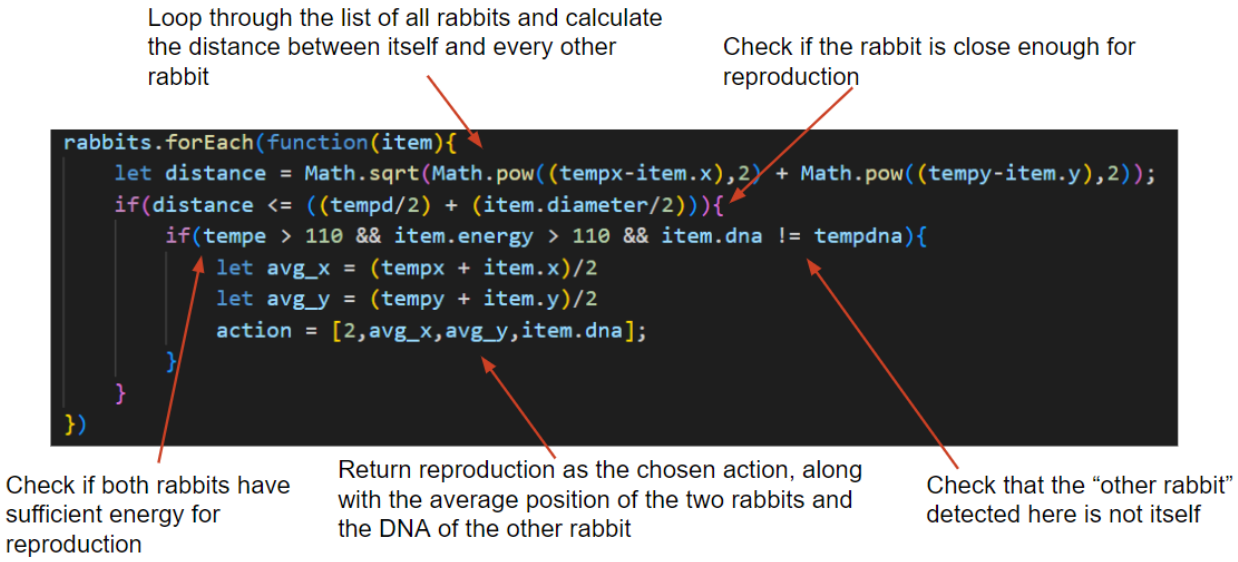
```
grass.forEach(function(item,index,object){
    let distance = Math.sqrt(Math.pow((tempx-item.x),2) + Math.pow((tempy-item.y),2));
    if(distance <= (tempd/2+18)){
        object.splice(index,1);
        action = [1];

    }
})
```

If the grass is close enough to the rabbit, the rabbit will eat the grass

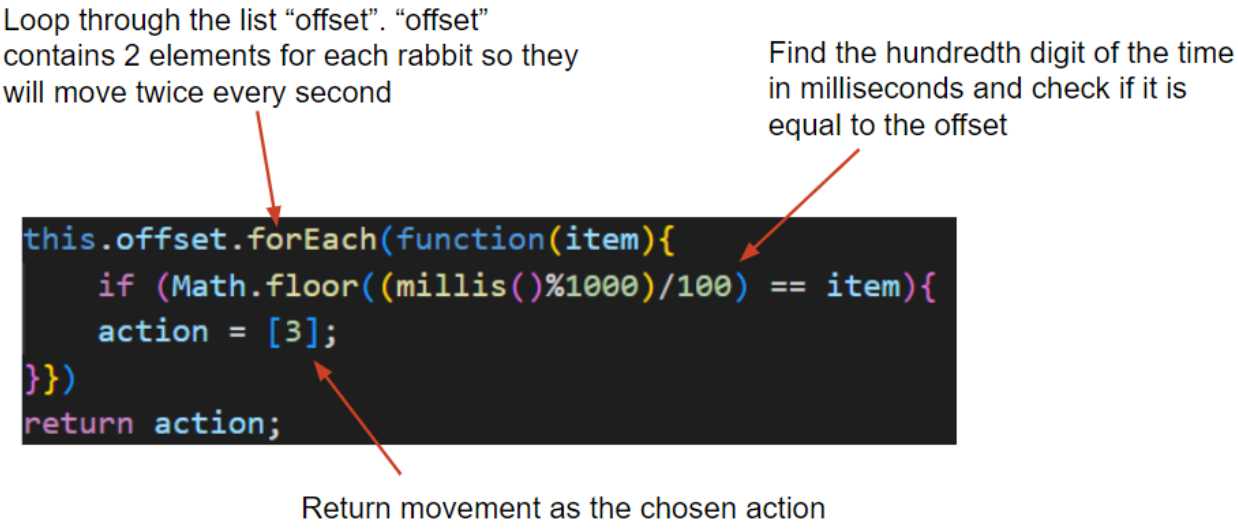
Remove the grass because it has now been eaten

Set action = [1] to indicate that the chosen action is to eat

Then, if there is no grass available to eat, the rabbit will look for any other rabbits ready to mate.



Finally if there is no grass to eat and no partners to mate with, the rabbit will decide whether to move or not.

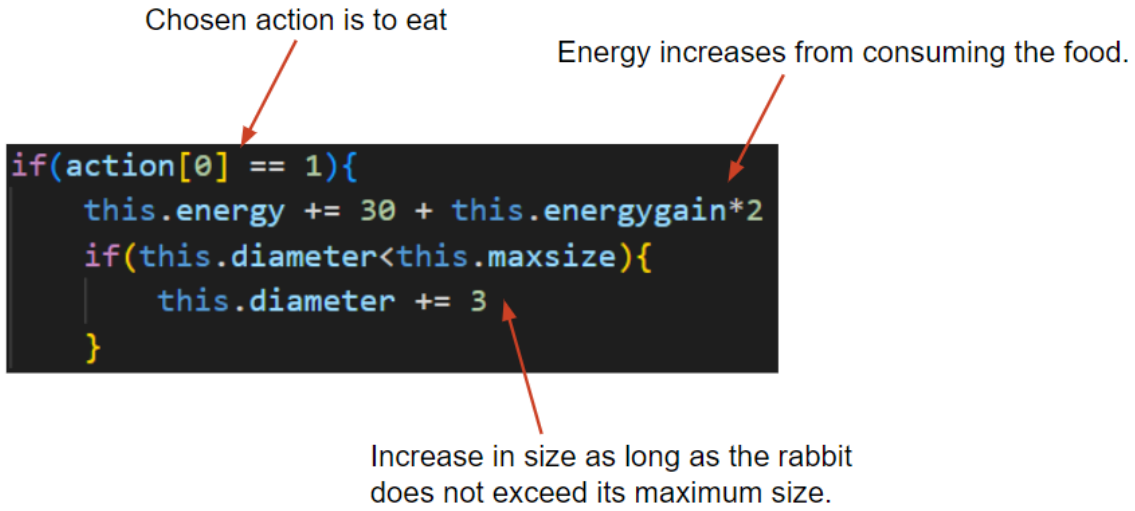


Since the target audience of this product is young students, interactiveness and visual appeal are factors I also have to consider. I decided to implement movement this way to mimic the sporadic movement of real rabbits. While this is not needed for demonstrating natural selection, I felt that this makes the simulation look a lot more lively rather than having every rabbit moving in tandem. Future developers can increase the frequency of the rabbit's movement by simply adding more elements to the "offset" array.

- **Actions**

The update() function carries out the chosen action based on what chooseaction() returns.

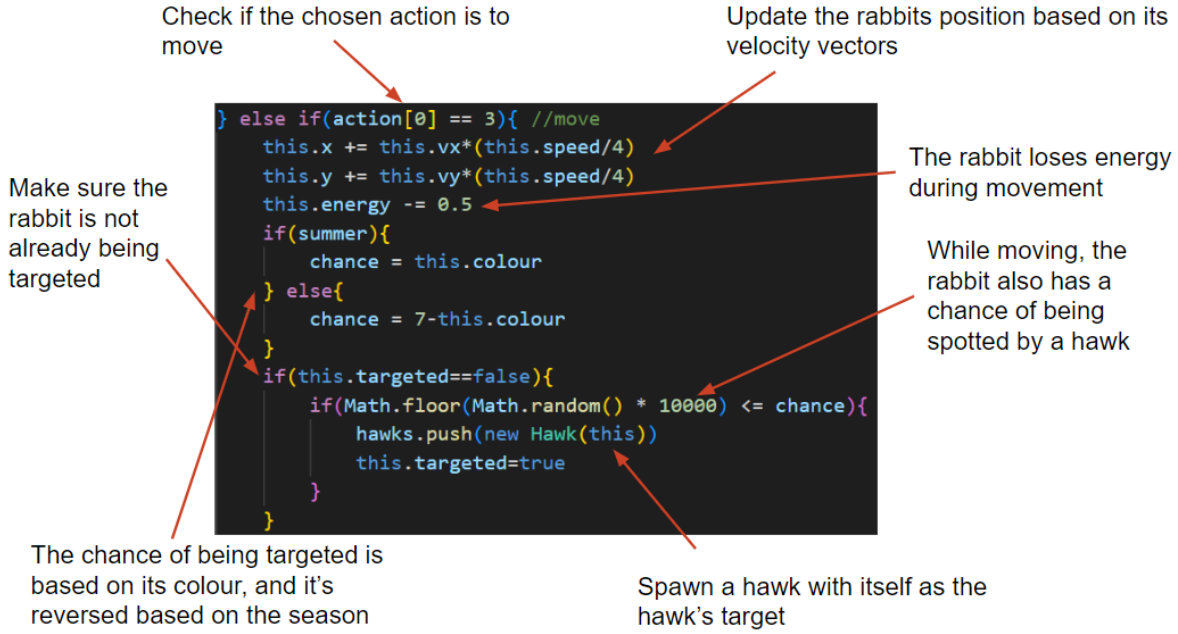
Action 1: Eat



Action 2: Reproduction

(see “crossover” section above)

Action 3: Move



Action 4: Choose direction

Check if the chosen action is to choose a direction

If the rabbit has sufficient energy already, it will prioritise reproduction and search for the nearest partner ready to mate

```
} else if(action[0] == 4){
    if(this.energy>150){
        rabbits.forEach(function(item){
            let distance = Math.sqrt(Math.pow((tempx-item.x),2) + Math.pow((tempy-item.y),2))
            if(distance<closest && item.dna != tempdna && item.energy > 150){
                closest = distance
                if(item.x > tempx){
                    tempvx = 1
                }else if(item.x < tempx){
                    tempvx = -1
                }else{
                    tempvx = 0
                }
                if(item.y > tempy){
                    tempvy = 1
                }else if(item.y < tempy){
                    tempvy = -1
                }else{
                    tempvy = 0
                }
```

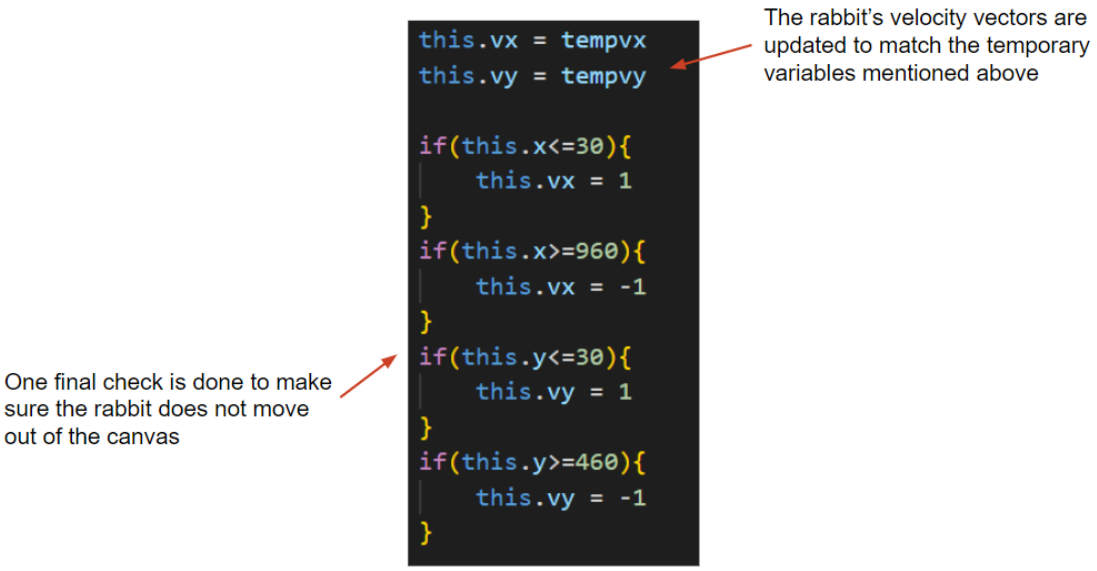
Temporary variables are then used to hold the direction of the nearest mating partner

The higher a rabbit's intelligence, the more chance it has to be able to search for food

If the rabbit doesn't have energy, it will prioritise searching for food instead

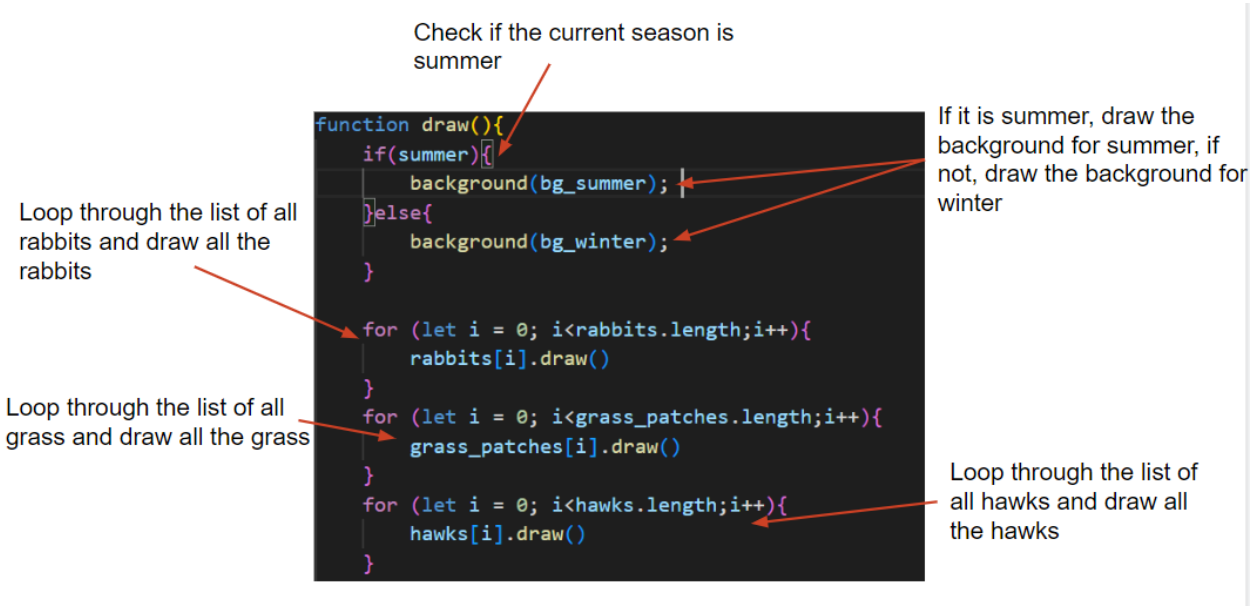
```
}else if(Math.floor(Math.random() * 15) <= this.intelligence){
    grass.forEach(function(item){
        let distance = Math.sqrt(Math.pow((tempx-item.x),2) + Math.pow((tempy-item.y),2))
        if(distance<closest){
            closest = distance
            if(item.x > tempx){
                tempvx = 1
            }else if(item.x < tempx){
                tempvx = -1
            }else{
                tempvx = 0
            }
            if(item.y > tempy){
                tempvy = 1
            }else if(item.y < tempy){
                tempvy = -1
            }else{
                tempvy = 0
            }
```

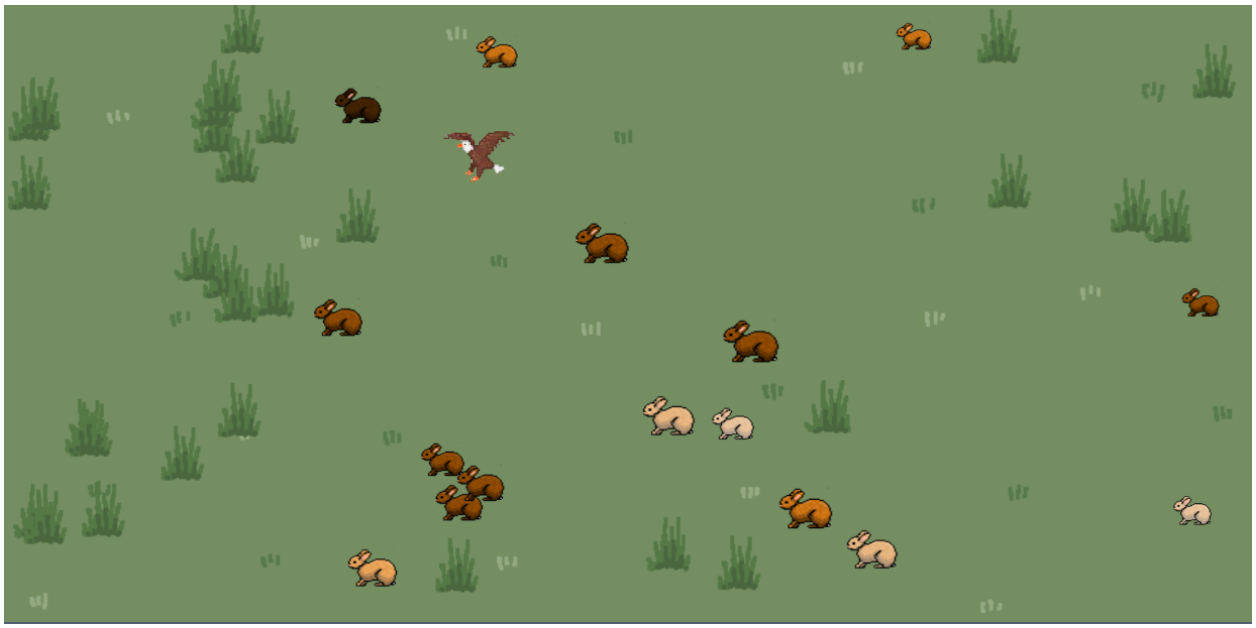
Once again, temporary variables are then used to hold the direction of the nearest piece of food



5. Environment

The environment governs everything that happens within the simulation. This is what allows the user to start, pause, or reset the simulation.





(what the environment with all the elements drawn looks like)

Check if the simulation is running

```
if (isrunning){
    rabbits.forEach(function(item,index,object){
        if(item.alive==false){
            object.splice(index,1)
        }
    })
    hawks.forEach(function(item,index,object){
        if(item.x<=-40 || item.y<=-40){
            object.splice(index,1)
        }
    })

    for (let i = 0; i<rabbits.length;i++){
        rabbits[i].update(grass_patches)
    }
    for (let i = 0; i<hawks.length;i++){
        hawks[i].update()
    }
    if(counter%250 == 0){
        addGrass(35-rabbits.length)
    }

    counter+=1
}
}
```

Loop through every rabbit to check if they're alive, removing them if not

Loop through every hawk to check if they're still on the canvas, removing them if not

Update the rabbits and hawks

Spawn new grass

Increment "counter" to keep track of the passage of time in the simulation

6. GUI

The generated user interface for this product is broken up into large sections to make it as intuitive for the user as possible. All the buttons, sliders and graphs also have clear labels to

indicate their purpose. The design is simple and clean so as to not draw attention away from the main focus of the product - the simulation.

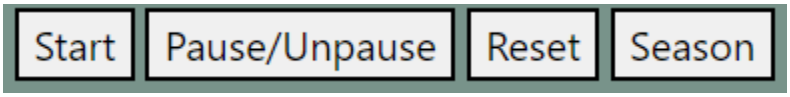
Organize the layout of the web page into columns and rows

```
{% extends "base.html" %}{%block content%}
<div class="container-fluid">
    <div class="row">
        <div class="col-8"><main></main></div>
        <div class="col-4">{%include 'right_graphs.html'%}</div>
    <div class="row">
        <style>
            .custom-bg-color {
              background-color: #78938A;
            }
        </style>
        <div class="col-7 custom-bg-color">
            <div class="row"><div id="flagText"></div></div>
            <button onclick="start()" id="start">Start</button>
            <button onclick="halt()" id="halt">Pause/Unpause</button>
            <button onclick="clearance()" id="clearance">Reset</button>
            <button onclick="updateText()">Season</button>
        </div>
        <div class="col-5 custom-bg-color">
            {%include 'right_sliders.html'%}
        </div>
    </div>
</div>
{%endblock%}
```

Includes the graphs which are written on a separate file

Separate the buttons by line for sake of clarity and extensibility

Includes the sliders which are written on a separate file



(buttons created by the code above)

Linking a bootstrap template and a stylesheet for formatting purposes

```
<!DOCTYPE html>
<html>
    <head><!-- CSS only -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="sty
        <link href="{{url_for('static', filename='style.css')}}", rel="stylesheet"/>
        <script src="https://cdn.jsdelivr.net/npm/p5@1.4.1/lib/p5.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.7.0/chart.min.js"></script>
        <script src="{{url_for('static', filename='program.js')}}"></script>
        <script src="{{url_for('static', filename='sketch.js')}}"></script>
    </head>
    <body>
        <div class="container-fluid">{%block content%}{%endblock%} </div>
        <!-- JavaScript Bundle with Popper -->
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" int
        <script>
          var popoverTriggerList = [].slice.call(document.querySelectorAll('[data-bs-toggle="popover"
          var popoverList = popoverTriggerList.map(function (popoverTriggerEl) {
            return new bootstrap.Popover(popoverTriggerEl)
          })
        </script>
    <script src="{{url_for('static', filename='graphs.js')}}"></script>
    </body>
    </html>
```

Accessing the p5.js and chart.js libraries

Connecting to the javascript files which contain the simulation

7. Graphs

The graphs on the top right of the webpage need to be able to fetch data from the simulation and update in real time. Due to its complexity, I wrote the code for the graphs on a separate html file for sake of clarity and ease of development.

- **Bar Chart**

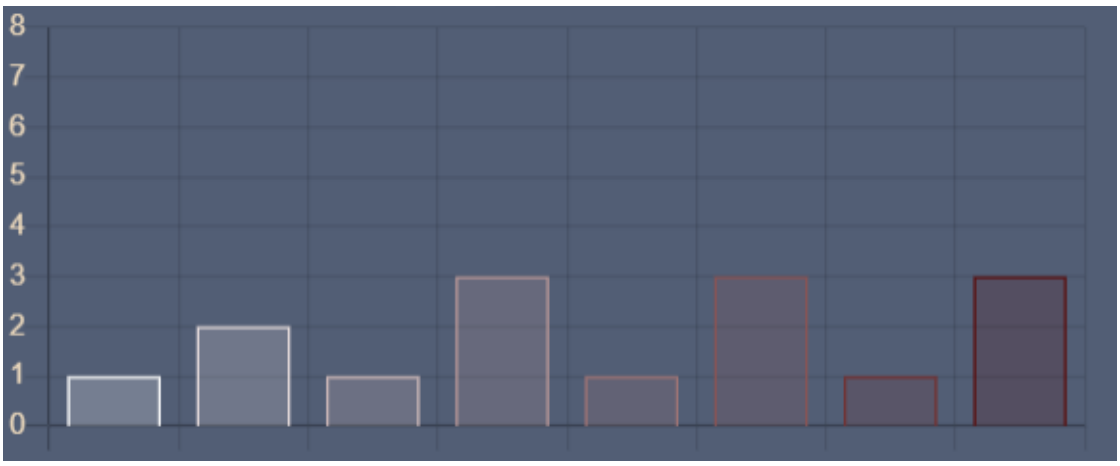
Creates a 2d bar chart using chart.js

```
<script>
  // Bar Chart
  const ctx1 = document.getElementById("barChart").getContext("2d");
  const myBarChart = new Chart(ctx1, {
    type: "bar",
    data: {
      labels: ['','','','','','','',''],
      datasets: [
        {
          label: "Variable Value",
          data: [0,0,0,0,0,0,0,0],
          backgroundColor: [
            "rgba(90, 15, 15, 0.2)",
            "rgba(115, 49, 49, 0.2)",
            "rgba(138, 84, 84, 0.2)",
            "rgba(161, 117, 117, 0.2)",
            "rgba(185, 152, 152, 0.2)",
            "rgba(208, 186, 186, 0.2)",
            "rgba(232, 221, 221, 0.2)",
            "rgba(255, 255, 255, 0.2)",
          ],
```

Load in 8 empty labels for the x axis

Initialize all the data at 0

Colour coding the bars to match the colours of the rabbits



- **Line Chart**

Creates a 2d line chart using chart.js

```
// Line Chart
var ctx2 = document.getElementById('lineChart').getContext('2d');
var myLineChart = new Chart(ctx2, {
  type: 'line',
  data: {
    datasets: [{
      label: 'Rabbit Population',
      data: [], // Update this with the list of numbers
      backgroundColor: 'rgba(255, 99, 132, 0.2)',
      borderColor: 'rgba(255, 99, 132, 1)',
      borderWidth: 0
    }]
```

Initialize an empty list so data can be added over time

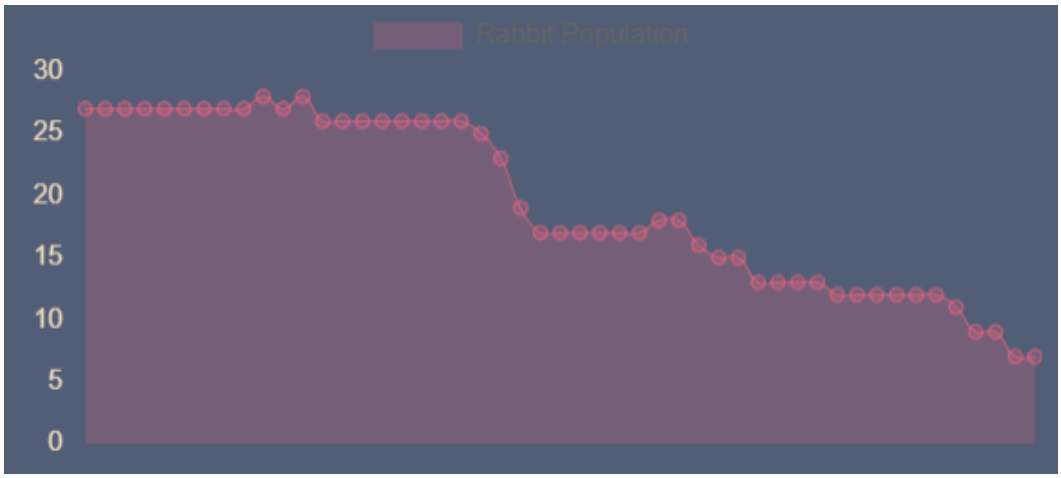
Appropriate title and clearly visible colours for the graph

```
options: {
  scales: {
    xAxes: [{
        gridLines: {
            display:false
    }
}],
    yAxes: [{
        gridLines: {
            display:false
        },
        ticks: {
            beginAtZero: true,
            max: 30,
            fontColor: '#F1DDBF',
      }
    }]
  }
}
```

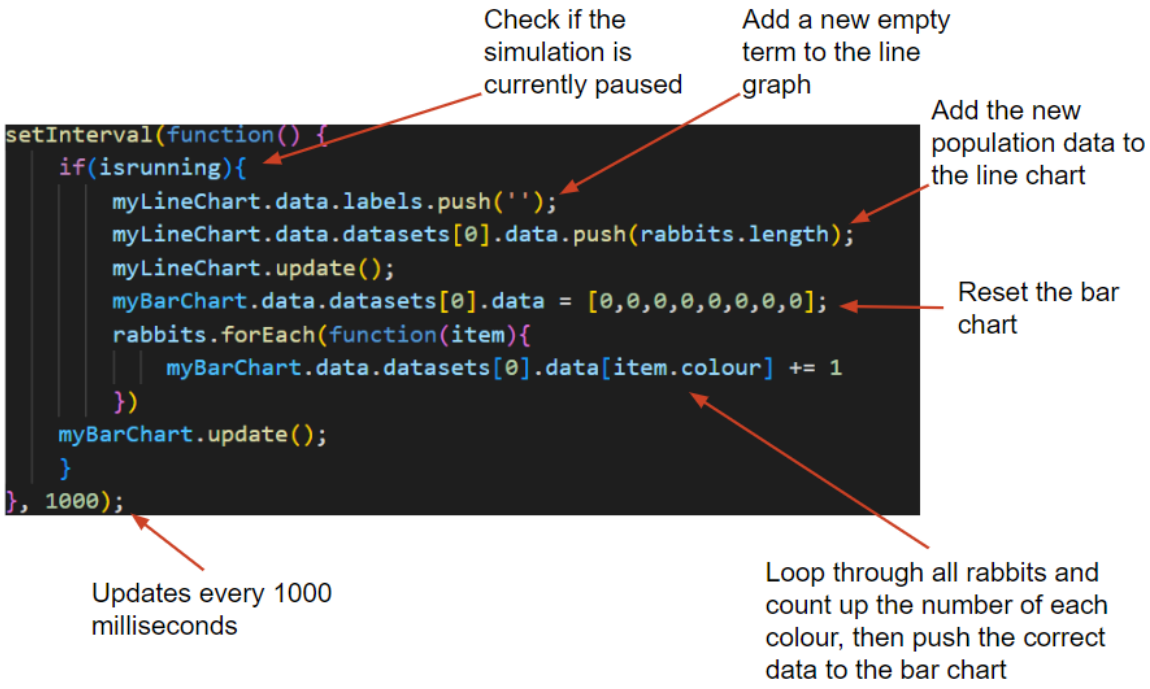
Hides the grid lines

Sets the y-axis to range from 0 to 30

Sets the y-axis to range from 0 to 30



I wrote a separate function named setInterval() for updating both graphs so they update at the same rate, which can be adjusted by future developers.




```
setInterval(function() {
    if(isrunning){
        myLineChart.data.labels.push('');
        myLineChart.data.datasets[0].data.push(rabbits.length);
        myLineChart.update();
        myBarChart.data.datasets[0].data = [0,0,0,0,0,0,0,0];
        rabbits.forEach(function(item){
            myBarChart.data.datasets[0].data[item.colour] += 1
        })
    myBarChart.update();
    }
}, 1000);
```

Bibliography

"Python virtualenv" Stack Overflow forums. Accessed 27/05/2022
https://stackoverflow.com/questions/35017160/how-to-use-virtualenv-with-python

"Navbar Templates" Bootstrap templates. Accessed 27/05/2022
https://getbootstrap.com/docs/4.0/examples/navbars/

“createCanvas()” p5.js Reference Guide. Accessed 13/06/2022
https://p5js.org/reference/#/p5/createCanvas

“JavaScript Class constructor” W3 Schools guide. Accessed 13/06/2022
https://www.w3schools.com/jsref/jsref_constructor_class.asp

“Range Sliders” W3 Schools guide. Accessed 25/06/2022
https://www.w3schools.com/howto/howto_js_rangeslider.asp

“onclick Event” W3 Schools guide. Accessed 25/06/2022
https://www.w3schools.com/jsref/event_onclick.asp

“genetic-algorithm-js” Github open resource. Accessed 18/09/2022
https://github.com/lodenrogue/genetic-algorithm-js
“Line Chart” Chart.js Reference Guide. Accessed 20/01/2022
https://www.chartjs.org/docs/latest/charts/line.html

“Bar Chart” Chart.js Reference Guide. Accessed 22/01/2022
https://www.chartjs.org/docs/latest/charts/bar.html

“image()” p5.js Reference Guide. Accessed 27/01/2022
https://p5js.org/reference/#/p5/image