

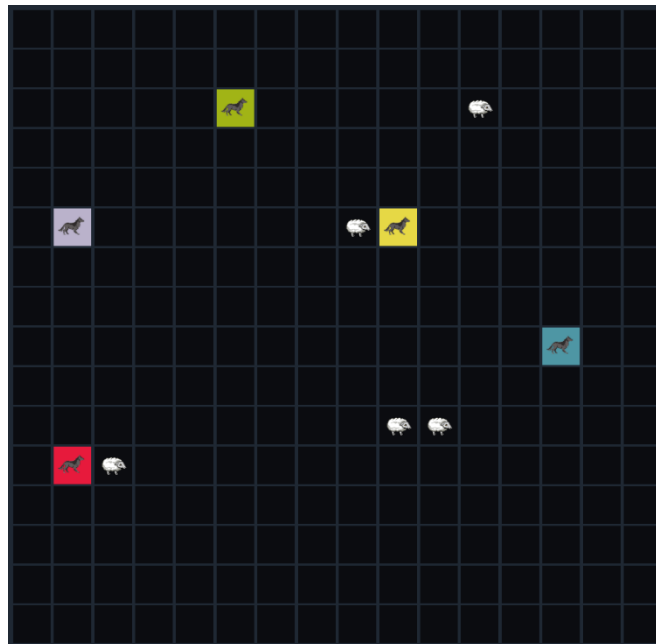
NOVA

IMS

Information  
Management  
School

# ARTIFICIAL LIFE

## PREDATOR MODELING



Artificial Intelligence 2020/21

Vitor Santos

Ana Lisboa Cotovio [r20181138]

Serhiy Bondarenko [r20181175]

| Patrik Pop [r20181157]

| Carlos Rincón [e20201158]

# INDEX

---

- 1. Introduction.....2
- 2. Rules.....4
- 3. Environment Design.....5
- 4. Prey modeling.....7
- 5. Predator modeling.....8
- 6. Conclusions.....14

# 1. INTRODUCTION

---

The aim of this project is to create a game that contains an environment in where artificial life is created and is capable of interacting with its surroundings. Our team has chosen the option of designing the Predator model, as we thought it was more interesting due to the nature of the goal, which is developing a hunter's way of thinking. Hunting over being hunted.

These predators or “**Xizoids**” are the ones being capable of reacting to the virtual environment. Their objective is to survive and last as long as they can in the game. Each one of them has an initial energy level, that is preselected by the player before starting the simulation. In each cycle or movement that occurs, predators lose vital energy and to keep it high, they should look for their preys and try to ‘eat’ them. We have decided to represent them as wolves, following its hunter spirit, for easier identification.



These preys, also called “**Ypsiloids**”, do not react to the environment and act as they had no brain or instinct. They are modeled to move around randomly, waiting to be eaten by the Xizoids. The number of Ypsiloids always remains the same, if one is eaten, a new one appears in some random point of the virtual world. The best representation for them was the sheep, as they are hunted by wolves, our predators.



Apart from the ‘hunting’ game, other things can happen in this environment, as it is intended to seem as real as possible. For instance, if two Xizoids are in the field of view of at least one of each other, there is a possibility of reproduction. If this is the case, another predator will be born in a random location and its characteristics would be a combination of his parents’ genes. There is a chance of mutation occurring, that will alter the child’s genome. Logically, every Xizoid has a different genome, which means different characteristics and different behavior towards his surroundings.


Firstly, the world or environment is created according to a minimum of 500 cells and knowing that in each cell can only contain one item at any given moment.



Here it is where the predators and preys will move and interact with each other. In the case some of them reach the limit of the cells, they will reappear at the opposite place as if the area was a

sphere. Same thing would happen if they were moving from the upper or bottom part of the virtual world.

The second task to be done is to set up the settings in where you can change the different parameters that define the game. For example, this menu includes the initial energy of each new individual, the mutation rate, the likelihood of reproduction occurrence, the initial number of Xizoids and Ypsiloids, the amount of energy gained by consuming the prey, the amount spent on reproduction and the maximum number of Xizoids. These are metrics required to get the game as close to reality as we can, for instance, it is obvious that reproducing does not come for free and some energy is needed to perform the task. They are also important to personalize the simulation and make it more amusing, as it may be very interesting to observe the changes in the animals' behavior and life cycle.



The image shows a dark-themed 'Initial Configuration' menu. It contains several input fields with numerical values: 'Initial energy of each new individual:' (100), 'Mutation rate:' (10), 'Likelihood of reproduction occurrence:' (50), 'Initial number of Xizoids:' (10), 'Initial number of Ypsiloids:' (10), 'Amount of energy gained from consumption:' (10), 'Amount of energy spent on reproduction:' (10), and 'Maximum number of Xizoids:' (30). Below these is a 'SET' button. Further down is a 'Year:' label with the value '42' and 'RUN' and 'PAUSE' buttons. At the bottom, there is a small horizontal bar chart with several green and grey dots.

Parameter	Value
Initial energy of each new individual:	100
Mutation rate:	10
Likelihood of reproduction occurrence:	50
Initial number of Xizoids:	10
Initial number of Ypsiloids:	10
Amount of energy gained from consumption:	10
Amount of energy spent on reproduction:	10
Maximum number of Xizoids:	30

Year: 42

RUN PAUSE

The most challenging part of this project has been the modeling of the predator's brain, which results to be an artificial neural network. Inputs from their surrounding are being collected and evaluated by several neurons, which produce and output that is transmitted to the motion system. The outcome depends on the situation of other preys, predators, hunger and more.

The rules for their behavior are explained in the next point and the deep detail will be shown later in the code.

The programming language that has been used is JavaScript, in conjunction with Git, to be able to allow errors in the developing line and not having to worry about the code messing up. Summing up, it makes the modeling process easier and better for working in group.

## 2. RULES

Xizoids have eyes to look for the prey, which provide an input for the artificial neural network depending on the Ypsiloid position. The desired output is to chase them until the prey is being eaten and gives energy to its hunter. Each one of the Xizoids will be able to see eight cells in front of him and think consequently.

In this picture we can observe that the prey is on the predator's visual range, as those are the eight cells in the wolf's vision when he is moving upwards.



Furthermore, the rules for motion in the environment were given and thought in a realistic way. The basic behavior is captured in this table.

Input	Output	Result
Sees prey on top right corner or front right	Move left leg	Moves front right
Sees prey on top left corner or front left	Move right leg	Moves front left
Sees prey on top 1 or 2 cells	Move both legs	Moves forward
Sees prey on the back	Move both legs backwards	Moves backwards
Sees prey on laterals or other direction	Move right leg laterally	Rotate 90°
Not worth the movement	Do not move	Does not move

As the energy level goes down by half a unit each movement the Xizoid makes or when and Xizoid reproduces, they might run out of it and die, being removed from the game. The ones with worse genetics will probably be the ones dying first.

Some other considerations are that the predators are hermaphrodites and the breeds will spawn in a random free, in order to simplify the code and modeling.

There is a way of evaluating the Xizoid's performance, which is measured by his age. The older, the better it is, as he has survived more time. We can visualize the older Xizoid in the world since he is represented with a golden background.



### 3. ENVIRONMENT DESIGN

---

First of all, when designing the world where the creatures are going to live, we need to define all the possible directions that an element on the board can go to.

```
//Enumeration containing all the possible directions that an element on the board can go to
const directionsEnum = {
  possibleDir:[
    UP = 'up',
    DOWN = 'down',
    LEFT = 'left',
    RIGHT = 'right',
    STAY = 'stay',
    EAT = 'eat',
    REPRODUCE = 'reproduce'
  ]
}
```

Right after that, we should model two functions that can be called afterwards to create the grid or physical environment, with the preferred number of rows and columns.

```
function getRows(numRows){
  for(var i = 0; i < numRows; i++){
    let row = document.createElement("div");
    board.appendChild(row).className = "gridRows";
    $(row).attr('id', 'r' + (i+1));
  }
}

function makeColumns(cellNum) {
  for (i = 0; i < rows.length; i++) {
    for (j = 0; j < cellNum; j++) {
      let newCell = document.createElement("div");
      rows[j].appendChild(newCell).className = "cell";
    }
  }
}

//create a board with different number of rows and columns
function makeBoard(num, num1){
  getRows(num);
  makeColumns(num1);
}
var numColumns = 16;
var numRows = 16;
makeBoard(numRows, numColumns);
```

We decided to include an interchangeable background color to each cell if you click on it, to be able to follow and register changes, certain positions and mainly to trace predator's paths.

```
function getRandomColor() {
  var letters = '0123456789ABCDEF';
  var color = '#';
  for (var i = 0; i < 6; i++) {
    color += letters[Math.floor(Math.random() * 16)];
  }
  return color;
}

Array.from(document.getElementsByClassName("cell")).forEach(function(element) {
  element.addEventListener('click', function () {
    $(element).css('background', getRandomColor());
  });
});
```

In order to establish some order and reference, a unique id is given to each cell in the board.

```
//give an unique id to every cell in the board
function setId(doc, prefix){
  var idCount = 1;
  $(doc).each(function() {
    $(this).attr('id', prefix + idCount);
    idCount++;
  });
}
```

To place elements randomly on the board and setting up the starting point of the game, the code asks for the quantity and symbol of the element, defined on the initial configuration menu.

```
//places the element on the board by asking the user the quantity and the symbol for the elemnt
function placeElement(times, symbol, elementClass, html){
  var placements = [];
  var count =0;
  var predatorPlacements = [];
  var preyPlacements = [];

  for(var i = 0; i <document.getElementsByClassName("Xpsilon").length; i++){
    predatorPlacements.push(parseInt(document.getElementsByClassName("Xpsilon")[i].getAttribute("id")));
  }
  for(var i = 0; i <document.getElementsByClassName("Ypsilon").length; i++){
    preyPlacements.push(parseInt(document.getElementsByClassName("Ypsilon")[i].getAttribute("id")));
  }

  while(count < times){

    var randomPosition = Math.floor((Math.random() *numColumns*numRows) +1);

    if(!placements.includes(randomPosition) && !predatorPlacements.includes(randomPosition) && !preyPlacements.includes(randomPosition)){
      var Y = html;
      $("#q" +randomPosition).append(Y);
      $("#q" +randomPosition).children('img').attr('id', randomPosition);
      placements.push(randomPosition);
      count++;
      console.log(placements);
      console.log(count-1);
    }

  }
  return placements;
}
```

## 4. PREY MODELING

At the beginning of the code, we defined the prey symbol, type and graphic.

```
const Ypsilon = {
  symbol: 'Y',
  type: 'prey',
  html: ''
};
```

Afterwards, we modeled the possible movements of the Ypsiloids and we realized that was also feasible to use this function to code the motion for the rest of the elements, not just the preys, by refactorization.

```
//function allows the Ypsilons to move to one of the 4 directions randomly using the Enumeration
//TODO: refactor so it can work for any element not just Y's
function moveY({
  Array.from(document.getElementsByClassName('Ypsilon')).forEach(function(element){

    var direction = directionsEnum.possibleDir[Math.floor(Math.random() * 4)];

    var move = movements[direction];

    if(move){
      move(element);
    }
    else{
      throw new Error('movement not allowed');
    }
  })
}
```

```
function moveX({
  xList.forEach(function(element){
    //connect the elements and the visual components

    pair = element.makeDecision();
    direction = directionsEnum.possibleDir[pair[1]];
    console.log(direction);
    var move = movements[direction];

    if(move){
      console.log("it moved");
      move(element);
    }
  })
}
```

Then we designed the logic behind the movements. There are four functions and each one of them represent one of the possible movements (up, down, left and right). It takes into account the circularity of the board. If an element is on one of the edges, it will be possible for it to move into that direction. The function first checks if this is the case and if not, it follows the normal situation scenario.

```
up(element){
  element.id = element.id || element.position;
  // case in which the element is in the top row and wants to move up

  //check if the element is in the first row
  if(parseInt(element.id)<=1+(numColumns-1) && parseInt(element.id)>=1){

    //move the element to the
    $("#q" + (parseInt(element.id) + ((numRows-1)*numColumns))).append(element);
    $("#q" + element.id).remove(element);
    console.log(+element.id + " was " + directionsEnum.possibleDir[0] + " and moved to the position " + (parseInt(element.id) + ((numRows-1)*numColumns)));
    $(element).attr('id', (parseInt(element.id) + ((numRows-1)*numColumns)));

  }
  //normal case
  else{
    $("#q" + (parseInt(element.id) - numColumns)).append(element);
    $("#q" + element.id).remove(element);
    console.log(element.id + " was " + directionsEnum.possibleDir[0] + " and moved to the position " + (parseInt(element.id) - numColumns));
    $(element).attr('id', (parseInt(element.id) - numColumns));
    console.log(element.id);
  }
}
```

Same reasoning for the other movements, but changing the restriction.

```
down(element){
  element.id = element.id || element.position;
  // case in which the element is in the bottom row and wants to move down
  if(parseInt(element.id)<=numRows*numColumns && parseInt(element.id)>=(numRows-1)*numColumns + 1 ){

left(element){
  element.id = element.id || element.position;
  //case in which the element is in the left most column and wants to move to the left
  if((parseInt(element.id) + (numColumns-1)) % numColumns == 0){

right(element){
  element.id = element.id || element.position;
  //case in which the element is in the right most column and wants to move to the right
  if(parseInt(element.id) % numColumns == 0){
```



## 5. PREDATOR MODELING

First of all, we set up the function for creating the Xizoids, which contains its generation and energy level, followed by giving them their characteristics. Those are unique to each of the predators, as every parameter is generated randomly. It also defines its symbol, class and graphic. After this, they are located in the board using the position function.

```
function createX(energyLevel, generation){
  this.generation = generation
  this.energyLevel = energyLevel;
  this.energy = {
    weight(){
      return 2;
    },
    connection:[
      left = Math.floor(Math.random()*10) +1,
      right = Math.floor(Math.random()*10) +1,
      up = Math.floor(Math.random()*10) +1,
      down = Math.floor(Math.random()*10)+1,
      stay = Math.floor(Math.random()*10)+1,
      eat = Math.floor(Math.random()*10)+1,
      reproduce = Math.floor(Math.random()*10)+1
    ]
  }

  this.symbol = 'X';
  this.class = "Xpsilon";
  this.html = '';
  this.fieldOfView = [];
  this.nonVisualInputs = [];
  this.sexDrive = {
    weight(){
      return 2;
    },
    connection:[
      left = Math.floor(Math.random()*10) +1,
      right = Math.floor(Math.random()*10) +1,
      up = Math.floor(Math.random()*10) +1,
      down = Math.floor(Math.random()*10)+1,
      stay = Math.floor(Math.random()*10)+1,
      eat = Math.floor(Math.random()*10)+1,
      reproduce = Math.floor(Math.random()*10)+1
    ]
  }
}
```

Now it is time for setting up the weights, which will let the artificial neural network to take understandable decisions and run properly. The main idea is to choose the maximum length path, this is, the highest number, which it is suppose to be the best option.

```
//TODO: SET THE WEIGHTS
this.setWeights = function(weights){
  this.weights = weights;
}
this.getPosition = function(){
  return this.position;
}
this.makeDecision = function(){
  possibleDecision = Array.from(Array(7), () => 0);
  var dec =0;
  for(var i = 0; i < this.network.length; i++){
    dec =0;
    var weight = this.network[i].weight();

    for(var j = 0; j < this.network[i].connection.length; j++){
      dec = parseInt(weight) * parseInt(this.network[i].connection[j]);

      possibleDecision[j] += dec;
    }
  }
  console.log(possibleDecision.indexOf(Math.max.apply(null, possibleDecision)));
  return[directionEnum.possibleDir[possibleDecision.indexOf(Math.max.apply(null, possibleDecision))],possibleDecision.indexOf(Math.max.apply(null, possibleDecisi ...
```

Lastly, it comes the largest part of the code, which contains the set up for the **field of vision** or FOV. It is divided on 11 parts, each one them referring to a different case scenario for the predator. The FOV has to change depending on the Xizoid's position, due to the sphere-like world. In the edge cases where the predator is located on the limit of the environment, in order to keep vision of the other side of the board, various models have to be developed.

There are 10 limit cases, when a new field of vision set up has to be done. The last model corresponds to the normal case where the Xizoid is surrounded by at least two rows and two columns on his FOV. The 10 limit cases are when the predator is positioned on the world's:

1. Upper right corner
2. Upper left corner
3. Down left corner
4. Down right corner
5. First row
6. Last row
7. Rightmost column
8. Second rightmost column excluding its first and last cell
9. Second rightmost column first cell
10. Second rightmost column last cell

Afterwards, the field of vision analysis is structured in three different levels, called radius on the code. The first level corresponds to the two closest cells, the second scans the first row above, and the third one observes the last row that the Xizoid has in reach.



The predator recognizes if in those cells there is an Ypsiloid, in which the code returns 3, another Xizoid (returns 5) or nothing (returns 1). Here we can see how the preference system works, assigning the highest number if there a possible prey to be eaten, and if not, maybe there is a Xizoid who can reproduce with him. This information is used as inputs for the weights in the artificial neural network. Finally, the brain decides which direction to take and moves the legs based on the rules described, always having survival in mind, but depending on its own characteristics.

Here it is how the normal case works:

```
//*****normal case*****
else{

    //check to see if the field of view is set
    if(!this.fieldOfView.length){
        this.fieldOfView = [];
    }
    //first radius of the FOV. The two closest cells
    this.fieldOfView.push(
        xCharacteristics = {
            position : this.getPosition()-numColumns,

            weight(){
                FOVContent = $('#q' +this.position).children().attr("alt");
                if(FOVContent == 'Y'){

                    return 1;
                }
                else if(FOVContent == 'X'){
                    return 0;
                }
                else{
                    return -1;
                }
            },
            connection:[
                left = Math.floor(Math.random()*10) +1,
                right =Math.floor(Math.random()*10) +1,
                up = Math.floor(Math.random()*10) +1,
                down = Math.floor(Math.random()*10)+1,
                stay = Math.floor(Math.random()*10)+1,
                eat = Math.floor(Math.random()*10)+1,
                reproduce = Math.floor(Math.random()*10)+1
            ]
        }
    );

    this.fieldOfView.push(xCharacteristics = {
        position : this.getPosition() + numColumns,

        weight(){
            FOVContent = $('#q' +this.position).children().attr("alt");
            if(FOVContent == 'Y'){

                return 1;
            }
            else if(FOVContent == 'X'){
                return 0;
            }
            else{
                return -1;
            }
        },
        connection:[
            left = Math.floor(Math.random()*10) +1,
            right =Math.floor(Math.random()*10) +1,
            up = Math.floor(Math.random()*10) +1,
            down = Math.floor(Math.random()*10)+1,
            stay = Math.floor(Math.random()*10)+1,
            eat = Math.floor(Math.random()*10)+1,
            reproduce = Math.floor(Math.random()*10)+1
        ]
    });
}
```

Then it is the turn for the second radius or level, finishing with the third one. Both of them follow the same structure that the first level has, calling the same functions but adapting the situation to the cells.

```
//second radius of the FOV
this.fieldOfView.push(
{
    position: this.getPosition() - (numColumns - 1),

//third radius of the FOV
this.fieldOfView.push(
{
    position: this.getPosition() -(numColumns - 2),
```

As an example, here it is how one of the limit cases work, in particular the upper right corner situation. The rest of them adapt the code to fit the environment's field of vision needs as a spherical world.

```
//*****upper right corner*****
if(this.position == numColumns){

    //check to see if the field of view is set
    if(!this.fieldOfView.length){
        this.fieldOfView = [];
    }
    //first radius of the FOV. The two closest cells
    this.fieldOfView.push(
        xCharacteristics = {
            position : this.getPosition() + ((numRows * numColumns) - numColumns),

            weight(){
                FOVContent = $('#q' +this.position).children().attr("alt");
                if(FOVContent == 'Y'){

                    return 1;
                }
                else if(FOVContent == 'X'){
                    return 0;
                }
                else{
                    return -1;
                }
            },
            connection:[
                left = Math.floor(Math.random()*10) +1,
                right =Math.floor(Math.random()*10) +1,
                up = Math.floor(Math.random()*10) +1,
                down = Math.floor(Math.random()*10)+1,
                stay = Math.floor(Math.random()*10)+1,
                eat = Math.floor(Math.random()*10)+1,
                reproduce = Math.floor(Math.random()*10)+1
            ]
        }
    );

    this.fieldOfView.push(xCharacteristics = {
        position : this.getPosition() + numColumns,

        weight(){
            FOVContent = $('#q' +this.position).children().attr("alt");
            if(FOVContent == 'Y'){

                return 1;
            }
            else if(FOVContent == 'X'){
                return 0;
            }
            else{
                return -1;
            }
        }
    });
```

```

//second radius of the FOV
this.fieldOfView.push(
{
    position: this.getPosition() + ((numRows * numColumns) - (numColumns * 2) + 1),

    weight(){
        FOVContent = $('#q' +this.position).children().attr("alt");
        if(FOVContent == 'V'){

            return 1;
        }
        else if(FOVContent == 'X'){
            return 0;
        }
        else{
            return -1;
        }
    },
    connection:[
        left = Math.floor(Math.random()*10) +1,
        right =Math.floor(Math.random()*10) +1,
        up = Math.floor(Math.random()*10) +1,
        down = Math.floor(Math.random()*10)+1,
        stay = Math.floor(Math.random()*10)+1,
        eat = Math.floor(Math.random()*10)+1,
        reproduce = Math.floor(Math.random()*10)+1
    ]
});

```

Here it continues as the first level does and ends with the third one, same reasoning as the normal case scenario.

```

//third radius of the FOV
this.fieldOfView.push(
{
    position: this.getPosition() + ((numRows * numColumns) - (numColumns * 2) + 2),
    ...

```

Finally, we have to design the function that allows new breeds to be born and placed, who are a genetical combination of its parents.

```

reproduce(element) {
  loopstop = false;
  element.id = element.position;
  var position;
  var partner;
  if(xList.length >= optionsArray[7].value){
    console.log("Cannot reproduce anymore, the population is too big");
  }else{
    //in the field of view try to find the position a new mating partner
    element.fieldOfView.forEach(function(cell){
      console.log("the cell: ");
      console.log(cell.position(element.position));
      if(loopstop){return;}

      if(cell.weight(cell.position(element.position)) == 5){
        position = cell.position(element.position);
        console.log("the position is " +position);

        //do not try to find another partner if partner is found
        loopstop = true;
      }
    });

    //after finding the position select the correct partner from the xList
    xList.filter(function(wolf){
      if(wolf.position == position){
        partner = wolf;
        console.log("the partner is" +partner);
      }
    });

    if(partner == undefined){
      console.log("There is no one to mate");
    }else{
      //drain the energy of the predators by the amount set by the user
      console.log("mating partner found");
      element.energyLevel = element.energyLevel - optionsArray[6].value;
      partner.energyLevel = partner.energyLevel - optionsArray[6].value;

      //try to reproduce using the likelihood of reproduction given by the user
      if(Math.floor(Math.random()*100) < optionsArray[2].value ){
        var child = element.crossover(partner);
        child.mutate(optionsArray[1].value);
        xList.push(child);
      }
    }
  }
}

```

The team added a final functionality to the game, with which the user is able to see the predator's position, generation and energy left by moving the mouse over his virtual representation, so it is possible to compare and rank them. Although not being completely compatible with the later generations of predators.

## 6. CONCLUSIONS

---

Modeling artificial life has been a great challenge. First, due to understanding how an artificial neural network works and also because it is not easy to translate those kind of systems into real code. It is really interesting the fact that we can reproduce and mimic a creature's brain using a computer, obviously with its restrictions and assumptions. For us this was our goal, and we are very satisfied with the journey and the results obtained.

We have created a simulation in where the user is able to choose the initial settings and observe the simulation of how a real environment would behave if there were hunters and hunted animals, at the most basic level, but including some surprising features, for instance, having the chance of watching new breeds being born.

This project has given us the opportunity to get closer to what the actual artificial intelligence is and now, this team can also imagine the AI boundaries which could be broken if the technology allows it, resulting in the most relevant developments that the human being will experience in the next years, at least that it is what we hope.

