

Ecosystem in Aquatic Environment Simulator

Mr. Elie Khoury

Mr. Wissam Salameh

B.S in Computer Science

June 2009

Abstract:

This senior simulates the behavior of three independent creatures in a sea environment. It helps to observe the theory of the survival of the fittest between these creatures taking into consideration the characteristics of each one.

It can be used in fields of environmental sciences to give an idea about how certain specific specie' evolution can help it to adapt in its environment and survive extinction.

In coming versions it will had a library containing all species of aquatic environment, as can simulates environments with places to hide, salty water, powerful current...

In its introductory version it's the basis of a great idea helping every company or organization taking care of aquatic environment to simulate the life of certain species they choose to help.

We implement it using C# programming language and the 3.5 .Net framework.

In the aid of polymorphism, inheritance and object oriented programming and a part of artificial intelligence we succeeded to finish this basic version of this software.

Beyond that this work introduces the Swarm Intelligence through the Fish Schooling Algorithm.

We include ZedGraph control to generate a report showing the final situation of the tested species in a graphical user interface.

Table of Figures

Figure 1 Separation: steer to avoid crowding local flockmates	9
Figure 2 Alignment: steer towards the average heading of local flockmates	9
Figure 3 Cohesion: steer to move toward the average position of local flockmates	10
Figure 4 Neighborhood	10
Figure 5 Class Diagram	13
Figure 6 Sea environment form	26
Figure 7 Options Form	27
Figure 8 Sharks	28
Figure 9 Fish	29
Figure 10 Plants	29
Figure 11 Fish schooling	30
Figure 12 Fish eating	31
Figure 13 Sharks hunting, and fish in evading mode	32
Figure 14 Different generations	33
Figure 15 Report	34

Table of Contents

Contents

Abstract:	2
Acknowledgements:	5
Chapter I	6
Introduction and Problem Definition	6
<u>1.1 Introduction to the general problem:</u>	<u>6</u>
<u>1.2 Problem Definition:</u>	<u>6</u>
<u>1.3 Senior Project Objectives:</u>	<u>6</u>
<u>1.4 Approach and main Results:</u>	<u>7</u>
<u>1.5 Report Organization:</u>	<u>7</u>
Chapter II	8
Background	8
<u>2.1 Definitions of the Basic Concepts:</u>	<u>8</u>
<u>2.2 Descriptions of the Used Algorithms:</u>	<u>8</u>
<u>2.3 Previous Work in the Field:</u>	<u>11</u>
<u>2.4 Project Motivation:</u>	<u>11</u>
Chapter III	12
Original Work	12
<u>3.1 Challenges:</u>	<u>12</u>
<u>3.2 Difficulties:</u>	<u>12</u>
<u>3.3: Skills and Techniques:</u>	<u>14</u>
Chapter 4	35
Conclusions	35
<u>4.1 Summary of The main result:</u>	<u>35</u>
<u>4.2 Main contribution of this project:</u>	<u>35</u>
<u>4.3 Possible extensions and future work:</u>	<u>36</u>

Acknowledgements:

First of all we thank God for giving us our great mother the nature, to take a little part of it as our work.

We would like to express our gratitude to all those who gave us the possibility to complete this senior project.

A great thanks to our beloved parents who gave us all the moral support needed to stay enthusiast and ready to face all the problems that faces us.

Our sincere recognition to the Faculty of Natural and Applied Sciences, especially the Department of Computer Science in allowing us the chance to show and work on our idea.

To our teachers, who gave us all their perseverance to show us the right road while giving us all novelty in computer science domain.

Special thanks to our great advisor Mr. Victor Sawma, who was the lighting candle in our walk toward success.

And finally we would like to thank all our colleagues that share us our difficult moments and helped us get through them.

Hoping this work will be faithful to every person who has contributed for its realization, we wish it will be valued to its just worth.

Chapter I

Introduction and Problem Definition

1.1 Introduction to the general problem:

Aquatic Biodiversity Crisis is an emerging problem threatening different species living in aquatic environments. Some types are vulnerable to be extinct due to the superiority of their predator which surpasses the line of balance.

The problem is the lack of software that human can use to help certain species fighting to survive in balance with other species.

1.2 Problem Definition:

Aquatic species are in need of human being intervention to minimize the risk of their misbalance. Rare are software that simulates and analyze the behavior and evolution of aquatic species.

1.3 Senior Project Objectives:

Visual simulation is one of the best ways for humans to interact with certain types of problems. Objectives of this work is to simulate the behavior of three aquatic species (Sharks, fish, aquatic plant), to track the theory of the survival of the fittest.

One of the most important objectives is to be the first step toward more advanced software that will simulates and analyze aquatic species in many conditions and with a wide variety of libraries, specifications and characteristics.

1.4 Approach and main Results:

To begin the first step, we found that it's better to choose species that we can work on theoretically first and have the major characteristics of aquatic life.

We choose three generic aquatic species:

- Sharks as predators for fish
- Fish as predators for plant and prey for sharks
- Plants as prey for fish

Like this we reflect the hierarchy existing in real aquatic life.

Although we worked very hard on this introductory version of this software, we were able to harvest a fully object oriented software using polymorphism, inheritance and part of artificial intelligence "Swarm Intelligence, fish schooling".

Moreover, we got a first version of a simulator that later on will be more advanced to contain all known aquatic species with their characteristics and behaviors.

1.5 Report Organization:

In the following chapters we will introduce the concepts of our work as the method we used to achieve it, as we will show a part of the problems we face and the hard stuff that was await.

In Chapter II we will define the basic concepts as we will describe the fish schooling algorithm we used. We will also show how genes pass from generation to other. Also we will briefly state works done in this field and the motivations that encourage us to accomplish this work.

In Chapter III we will focus on the work done by showing parts of code that was hard to implement and other to manipulate. We will also include screenshots for the software at runtime.

In Chapter IV we will conclude by stating summary of the main results and contributions of the project, and end it with possible future extensions.

Chapter II

Background

2.1 Definitions of the Basic Concepts:

A simulator simulates an environment for the purpose of training or research.

The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system. It is the imitation of some real thing, state of affairs, or process. It aims to create an environment that is as close as possible to reality.

In this project, sharks, fish and plants will be simulated in their minimal behaviors and characteristics. We'll show predators and preys, how a prey can fight to survive "fish can evade", and how it will genetically evolve from generation to generation to fight extinction.

2.2 Descriptions of the Used Algorithms:

Swarm intelligence (SI) is an artificial intelligence technique based around the study of collective behavior in decentralized, self-organized systems. SI systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. We used fish schooling algorithm in our work to let fish school together when met.

This algorithm uses mathematical formulas and functions to operate.

The distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This formula is used to get the distance between two points. We use it in our work to compare it with the speed of the creature because speed here is pixels far from current location and to check the sight of the creature.

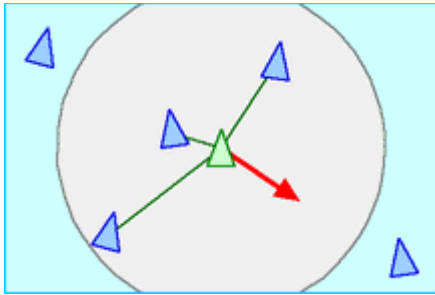


Figure 1 Separation: steer to avoid crowding local flockmates

In Figure 1 the fish try to make some distance between itself and the one in its space, because of this algorithm when fish school the ones on sides have space each from other while the ones inside can't have it because they are surrounded . The function which does this job is:

```

if (distance < space)
{
    // Create seperation.
    uX += Location.X - creature.Location.X;
    uY += Location.Y - creature.Location.Y;
}

```

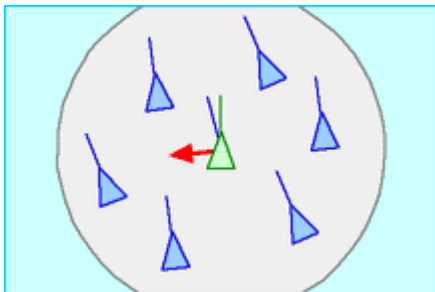


Figure 2 Alignment: steer towards the average heading of local flockmates

In Figure 2 the fish try to align itself with the surrounding fish, because of this algorithm all of them stay in harmony without chaos. The function which does this job is:

```

else if (distance < sight)
{
    // Align PerformBehaviorment.
    uX += creature.uX * 0.5f;
    uY += creature.uY * 0.5f; }

```

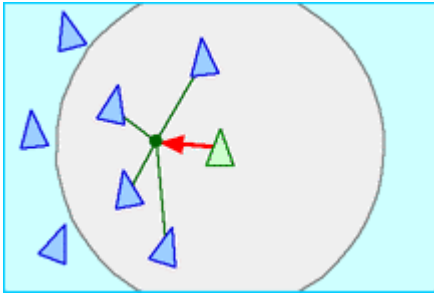


Figure 3 Cohesion: steer to move toward the average position of local flockmates

In Picture 3 the fish is moving a little bit at a time toward each neighbor in order to accomplish the schooling. The function which does this job is:

```

else if (distance < sight)
{
    // School together.
    uX += (creature.Location.X - Location.X) * 0.05f;
    uY += (creature.Location.Y - Location.Y) * 0.05f;
}

```

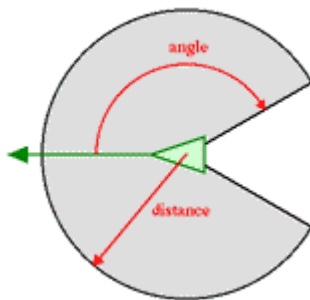


Figure 4 Neighborhood

Each fish has direct access to the whole scene's geometric description, but schooling requires that it reacts only to schoolmates within a certain small neighborhood around itself. The neighborhood is characterized by a distance (measured from the center of the fish) and an angle, measured from the fish' direction of swim. Schoolmates outside this local neighborhood are ignored. We obtain the angle by using the arctangent trigonometric formula. The function which does this job is:

```

else angle = (float) (Math.Atan(creature.uY / creature.uX) * 57.3);

```

2.3 Previous Work in the Field:

Software simulators have been done since time, but not with enough numbers to cover all fields and sections that can be simulated, especially simulators concerning life in aquatic environments. We will state three different simulators that we found while doing our research.

Examples of known simulators:

- ASCEND: It's a free, open source, mathematical modeling system developed at Carnegie Mellon University since the late 1980s. Its main uses have been in the field of chemical process modeling although its capabilities are general.
- Simulator: is a genus of small freshwater snails or limpets, aquatic gastropod mollusks in the family Planorbidae.
- Simulator consetti is a species of small freshwater snail or limpet, an aquatic gastropod mollusk in the family Planorbidae, the ram's horn snails and their allies.

2.4 Project Motivation:

Still in research phase about this work where we found that very rare are the software that do the job done and intended to be done by this project, this is the first of misses done by this field.

We chose this project to participate in protecting the sea life, and we found it as opportunity to begin developing an advanced system that will find place in many environmental science companies and organizations working in this field.

Our great motivation was that our senior project will be the first version of advanced software that we'll continue to develop.

Chapter III

Original Work

3.1 Challenges:

While searching for the right subject to base our senior project on, we stumbled upon an idea related to sea environment and the dangers that are facing the sea and its species. One of these dangers is the biodiversity crisis, so we decided to adopt this problem by creating a simulator of an aquatic ecosystem which will simulate the theory of the survival of the fittest.

When we reached the technical part of our research, we found that advanced skills and techniques must be used to achieve our purpose. This was a big challenge for us, but wasn't enough to breakdown our morals, on the contrary it was a push to overcome these difficulties.

3.2 Difficulties:

The technical section included polymorphism, object oriented programming, inheritance and artificial intelligence.

Throughout our simulator we implemented objects like the "Creature" class, "Shark" class, and many others.

Polymorphism and inheritance were clear in our code where the parent class named "Creature" was inherited by three children "Shark", "Fish" and "Plant" each one with overridden functions and other private data members and functions. Visually speaking you will see one creature in many shapes through polymorphism.

Also we used the Fish Schooling algorithm which is a part of Swarm Intelligence to let fish school together when met. And to take evolution into consideration, we implemented genetic inheritance algorithm.

A class diagram before demonstrating skills and techniques.

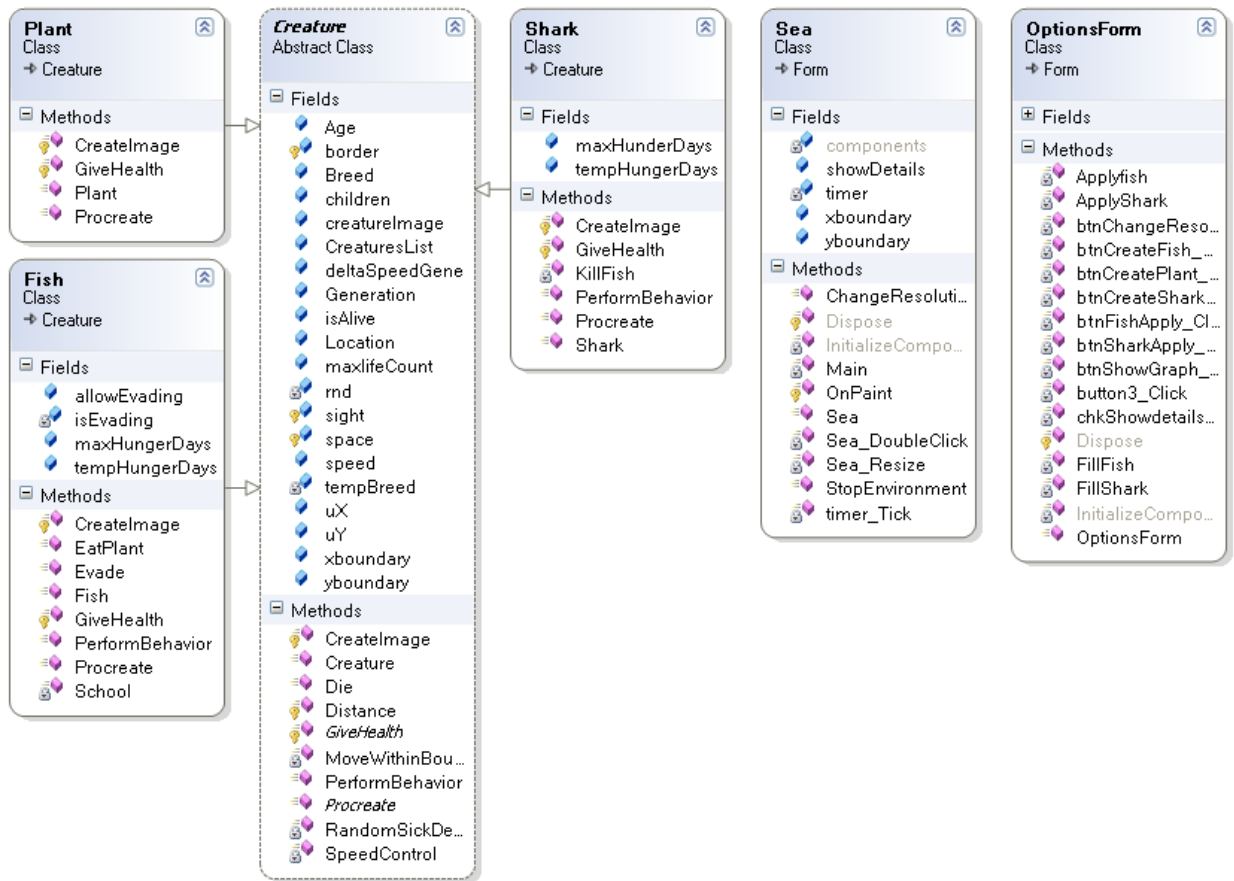


Figure 5 Class Diagram

3.3: Skills and Techniques:

Parent Class “Creature” implemented code:

```
#region Using

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

#endregion

namespace SeaEnvironment
{
    public abstract class Creature
    {
        #region Data Members For Class Creature

        public bool isAlive = true;
        public int maxlifeCount = 200;
        private static Random rnd = new Random();
        public static List<Creature> CreaturesList = new List<Creature>();
        protected static float border = 100f;
        protected static float sight = 75f;
        protected static float space = 30f;
        public float speed = 5f;
        public float xboundary;
        public float yboundary;
        public int Age = 400;
        public int Breed = 200;
        public int children = 4;
        private int tempBreed = 0;
        public float uX; //unit for x-axis
        public float uY; //unit for y-axis
        public int Generation = 1;
        public float deltaSpeedGene = 0;
```

```

public Image creatureImage;
public PointF Location;

#endregion

#region Constructor For Class Creature

public Creature(int xboundary, int yboundary, Creature ancestor)
{
    Location = new PointF(rnd.Next(xboundary), rnd.Next(yboundary));
    this.xboundary = xboundary;
    this.yboundary = yboundary;

    if (ancestor != null)
    {
        this.Generation = ancestor.Generation + 1;
        Random rnd2 = new Random();
        this.speed = this.speed + ancestor.deltaSpeedGene;
    }
    this.CreateImage();
    CreaturesList.Add(this);
}

#endregion

#region Functions For Class Creature

#region Virtual Function Die

public virtual void Die()
{
    isAlive = false;
}

#endregion

#region Virtual Function PerformBehavior

public virtual void PerformBehavior()
{
    RandomSickDeath();
}

```

```
    if (this.isAlive)
    {
        Age--;
        tempBreed++;
        if (tempBreed == Breed)
        {
            int randomChildren = rnd.Next(0, children);
            for (int i = 0; i < randomChildren; i++)
            {
                this.Procreate();
            }
            tempBreed = 0;
        }
        if (Age == 0)
            this.Die();
        MoveWithinBounds();
    }
}
```

#endregion

#region Virtual Function CreateImage

```
protected virtual void CreateImage()
{
    this.creatureImage = null;
}
```

#endregion

#region Abstract Function Procreate

```
public abstract void Procreate();
```

#endregion

#region Abstract Function GiveHealth

```
protected abstract void GiveHealth();
```

#endregion

```

#region Protected Function Distance

protected static float Distance(PointF p1, PointF p2)
{
double val = Math.Pow(p1.X - p2.X, 2) + Math.Pow(p1.Y - p2.Y, 2);
return (float)Math.Sqrt(val);
}

#endregion

#region Private Function RandomSickDeath

private void RandomSickDeath()
{

    if (this.isAlive)
    {
        int optim = rnd.Next(0, 1000);
        if (optim == 500)
            this.Die();
    }
}

#endregion

#region Private Function MoveWithinBounds

private void MoveWithinBounds()
{

    float xval = xboundary - border;
    float yval = yboundary - border;

    if (Location.X < border) uX += border - Location.X;
    if (Location.Y < border) uY += border - Location.Y;
    if (Location.X > xval) uX += xval - Location.X;
    if (Location.Y > yval) uY += yval - Location.Y;
    if (uX == 0 && uY == 0)
    {
        Random rn3 = new Random();
        uX = rn3.Next(100);
    }
}
}

```

```

        uY = rn3.Next(100);
    }
    SpeedControl();
    Location.X += uX;
    Location.Y += uY;
}

#endregion

#region Private Function SpeedControl

private void SpeedControl()
{
    float s;
    s = speed;

    float val = Distance(new PointF(0f, 0f), new PointF(uX,uY));
    //unit vector
    if (val > s)
    {
        uX = uX * s / val;
        uY = uY * s / val;
    }
}

#endregion

#endregion

}
}

```

Child Class “Shark” implemented code:

```
#region Using

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

#endregion

namespace SeaEnvironment
{
    public class Shark:Creature
    {
        #region Data Members For Class Shark

        public int maxHunderDays = 150;
        public int tempHungerDays;

        #endregion

        #region Constructor For Class Shark

        public Shark(int xboundary, int yboundary, Creature crt)
            : base(xboundary, yboundary, crt)
        {
            this.Breed = 150;
            this.children = 4;
            tempHungerDays = maxHunderDays;
            this.maxlifeCount = 50;
        }

        #endregion

        #region Functions For Class Shark

        #region Overrided Function CreateImage

        protected override void CreateImage()
        {
            Bitmap icon = new Bitmap(25, 10);
            Graphics g = Graphics.FromImage(icon);
            Point p1 = new Point(0, 0);
            Point p2 = new Point(0, 10);
            Point p3 = new Point(3, 5);
            Point p4 = new Point(17, 10);
            Point p5 = new Point(25, 5);
            Point p6 = new Point(17, 0);
            Point p7 = new Point(3, 5);
            Point[] points = { p1, p2, p3, p4, p5, p6, p7 };

            g.FillPolygon(Brushes.Red, points);
            this.creatureImage = icon;
        }

        #endregion
    }
}
```

```

#region Overriden Function Procreate

public override void Procreate()
{
    int sharkCount = 0;

    foreach (Creature crt in CreaturesList)
    {
        if (crt.isAlive && crt is Shark)
            sharkCount++;

    }

    if (sharkCount <= maxlifeCount)
    {
        Shark sc = new Shark(Sea.yboundary,Sea.yboundary,this);
    }
}

#endregion

#region Overriden Function GiveHealth

protected override void GiveHealth()
{
    this.Age = Age +1;
    tempHungerDays = maxHunderDays+10;
    this.deltaSpeedGene = deltaSpeedGene + 0.15f;
}

#endregion

#region Overriden Function PerformBehavior

public override void PerformBehavior()
{
    this.KillFish();
    base.PerformBehavior();
}

#endregion

#region Private Function KillFish

private void KillFish()
{
    if (this.isAlive)
    {
        tempHungerDays--;

        if (tempHungerDays <= 0)
            this.Die();

        float range = float.MaxValue;
        Fish prey = null;

        foreach (Creature fishy in CreaturesList)
        {
            if (fishy is Fish && fishy.isAlive)
            {

```

```

        float distance = Distance(Location, fishy.Location);

        if (distance < sight && distance < range)
        {
            range = distance;
            prey = fishy as Fish;
        }
    }

    if (prey != null)
    {
        uX += prey.Location.X - Location.X;
        uY += prey.Location.Y - Location.Y;
        if ((uX <= 60 && uX >= -60) && (uY <= 60 && uY >= -60))
        {
            prey.Die();
            this.GiveHealth();
        }
    }
}

#endregion

#endregion

}
}

```

Child Class “Fish” implemented code:

```
#region Using

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Windows.Forms;

#endregion

namespace SeaEnvironment
{
    public class Fish:Creature
    {
        #region Data Members For Class Fish

        public bool allowEvading = true;
        bool isEvading = false;
        public int maxHungerDays = 100;
        public int tempHungerDays;

        #endregion

        #region Constructor For Class Fish

        public Fish(int xboundary,int yboundary,Creature ancestor)
            : base(xboundary,yboundary,ancestor)
        {
            tempHungerDays = maxHungerDays;
            this.maxlifeCount = 100;
        }

        #endregion

        #region Functions For Class Fish

        #region Overriden Function Create Image

        protected override void CreateImage ()
        {
            Bitmap icon = new Bitmap(18, 8);
            Graphics g = Graphics.FromImage(icon);
            Point p1 = new Point(0, 0);
            Point p2 = new Point(0, 8);
            Point p3 = new Point(3, 4);
            Point p4 = new Point(11, 8);
            Point p5 = new Point(18, 4);
            Point p6 = new Point(11, 0);
            Point p7 = new Point(3, 4);
            Point[] points = { p1, p2, p3, p4, p5, p6, p7 };

            g.FillPolygon(Brushes.Blue, points);
            this.creatureImage = icon;
        }
    }
}
```

```

#endregion

#region Overrided Function GiveHealth

protected override void GiveHealth()
{
    this.Age = Age + Age * 5 / 100;
    tempHungerDays = maxHungerDays;
}

#endregion

#region Overrided Function Procreate

public override void Procreate()
{
    int fishCount = 0;

    foreach (Creature crt in CreaturesList)
    {
        if (crt.isAlive && crt is Fish)
            fishCount++;
    }
    if (fishCount <= maxlifeCount)
    {
        Fish fsc = new Fish(Sea.xboundary, Sea.yboundary, this);
    }
}

#endregion

#region Overrided Function PerformBehavior

public override void PerformBehavior()
{
    this.Evade();
    if (!this.isEvading)
    {
        School();
        Random rdd = new Random();
        int rnd = rdd.Next(0, 4);
        if (rnd == 2)
            this.EatPlant();
    }
    base.PerformBehavior();
}

#endregion

#region Public Function Evade

public void Evade()
{
    if (allowEvading)
    {
        foreach (Creature cr in CreaturesList)
        {
            if (cr is Shark && cr.isAlive)
            {
                float distance = Distance(Location, cr.Location);

```

```

        if (distance < sight)
        {
            // Avoid sharks.
            isEvading = true;
            this.deltaSpeedGene = deltaSpeedGene + 0.02f;
            uX += Location.X - cr.Location.X;
            uY += Location.Y - cr.Location.Y;
        }
    }
}

#endregion

#region Public Function EatPlant

public void EatPlant()
{
    if (this.isAlive)
    {
        tempHungerDays--;
        if (tempHungerDays <= 0)
            this.Die();

        float range = float.MaxValue;
        Plant prey = null;

        foreach (Creature planty in CreaturesList)
        {
            if (planty is Plant && planty.isAlive)
            {
                float distance = Distance(Location, planty.Location);

                if (distance < sight && distance < range)
                {
                    range = distance;
                    prey = planty as Plant;
                }
            }

            if (prey != null)
            {
                uX += prey.Location.X - Location.X;
                uY += prey.Location.Y - Location.Y;
                if ((uX <= 60 && uX >= -60) && (uY <= 60 && uY >= -
60))
                {
                    prey.Die();
                    this.GiveHealth();
                }
            }
        }
    }
}

#endregion

#region Private Function School

```

```

private void School()
{
    foreach (Creature creature in CreaturesList)
    {
        if (creature.isAlive && creature is Fish)
        {
            float distance = Distance(Location, creature.Location);

            if (creature != this)
            {
                if (distance < space)
                {
                    // Create seperation.
                    uX += Location.X - creature.Location.X;
                    uY += Location.Y - creature.Location.Y;
                }

                else if (distance < sight)
                {
                    // Align PerformBehaviorment.
                    uX += creature.uX * 0.5f;
                    uY += creature.uY * 0.5f;
                    // School together.
                    uX += (creature.Location.X - Location.X) * 0.05f;
                    uY += (creature.Location.Y - Location.Y) * 0.05f;
                }
            }
        }
    }
}

#endregion

#endregion
}
}

```

The above code show how the parent class “Creature” is shown in three different child; “Shark”, “Fish” and “Plant”. Like this the technique of polymorphism and inheritance was used. And we can notice the private school function in the “Fish” class which is a part of the Swarm Intelligence AI.

Below we will present screenshots taken from our simulator taking into consideration all cases and explained.

The sea environment form where species live to be simulated.

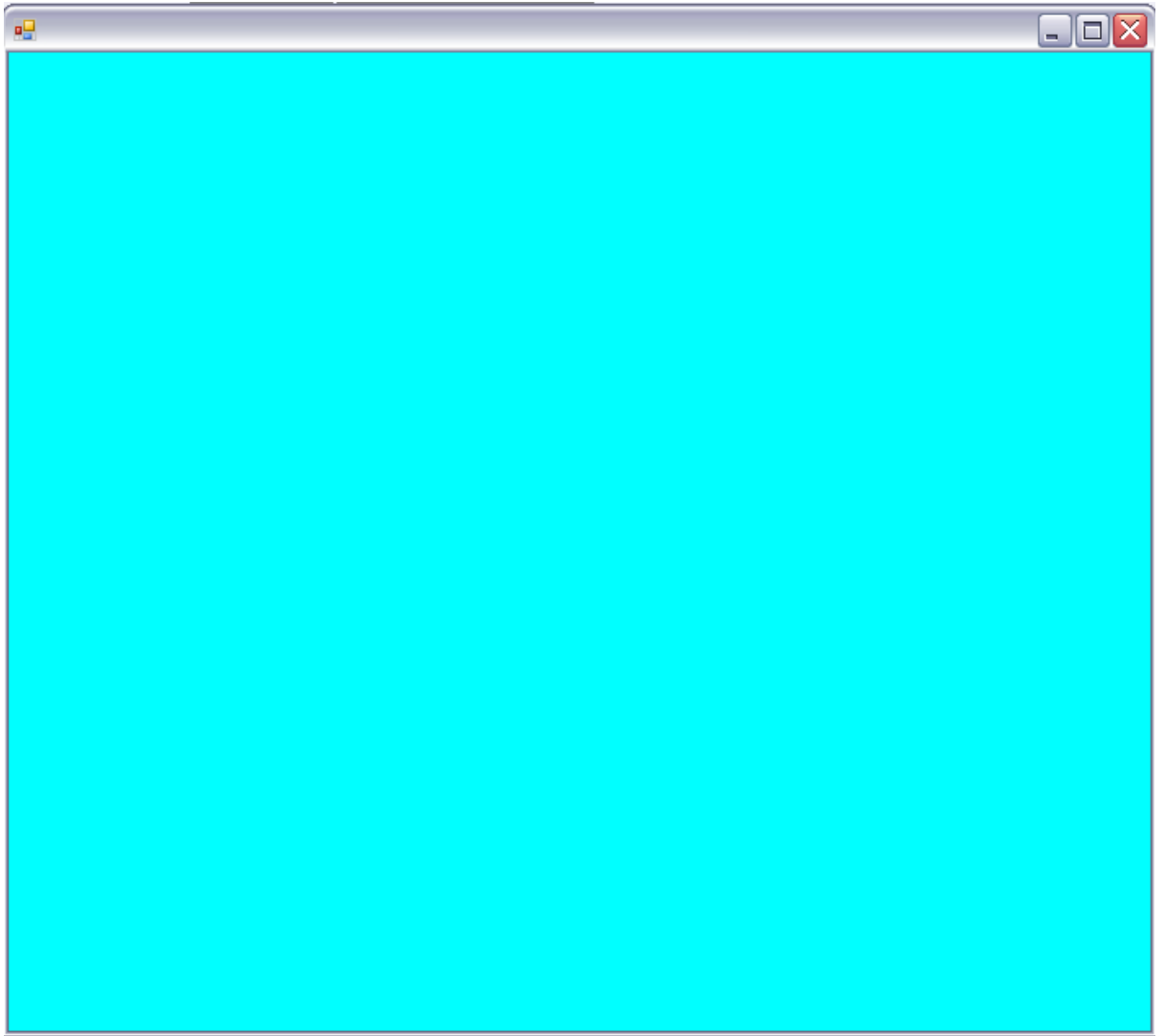


Figure 6 Sea environment form

The options form where the user can input the speed, age, breeding period...

The screenshot shows a dialog box titled "OptionsForm" with a close button (X) in the top right corner. The dialog is organized into three main sections: "Fish", "Sharks", and "Plants".

- Fish Section:** Contains four spinners for "Speed" (value 0), "Age" (value 0), "Breeding period" (value 0), and "Breed children" (value 0). There are two buttons: "Create Fish" and "Apply". A checkbox labeled "AllowEvasion" is present and is currently unchecked.
- Sharks Section:** Contains five spinners for "Speed" (value 0), "Age" (value 0), "Breeding period" (value 0), "Breed children" (value 0), and "Max Hunger Hours" (value 0). There are two buttons: "Create Shark" and "Apply".
- Plants Section:** Contains three spinners for "Age" (value 0), "Breeding period" (value 0), and "Breed children" (value 0). There are two buttons: "Create Plant" and "Apply".

At the bottom of the dialog, there are two text boxes showing "800x600" and "1024x768", a "Change Resolution" button, a checkbox labeled "Show Details" (unchecked), and a "View Report" button.

Figure 7 Options Form

Sharks which inherit creature. Printed, will show us especially when compared to followed pictures how the parent class “Creature” can be seen with different faces.

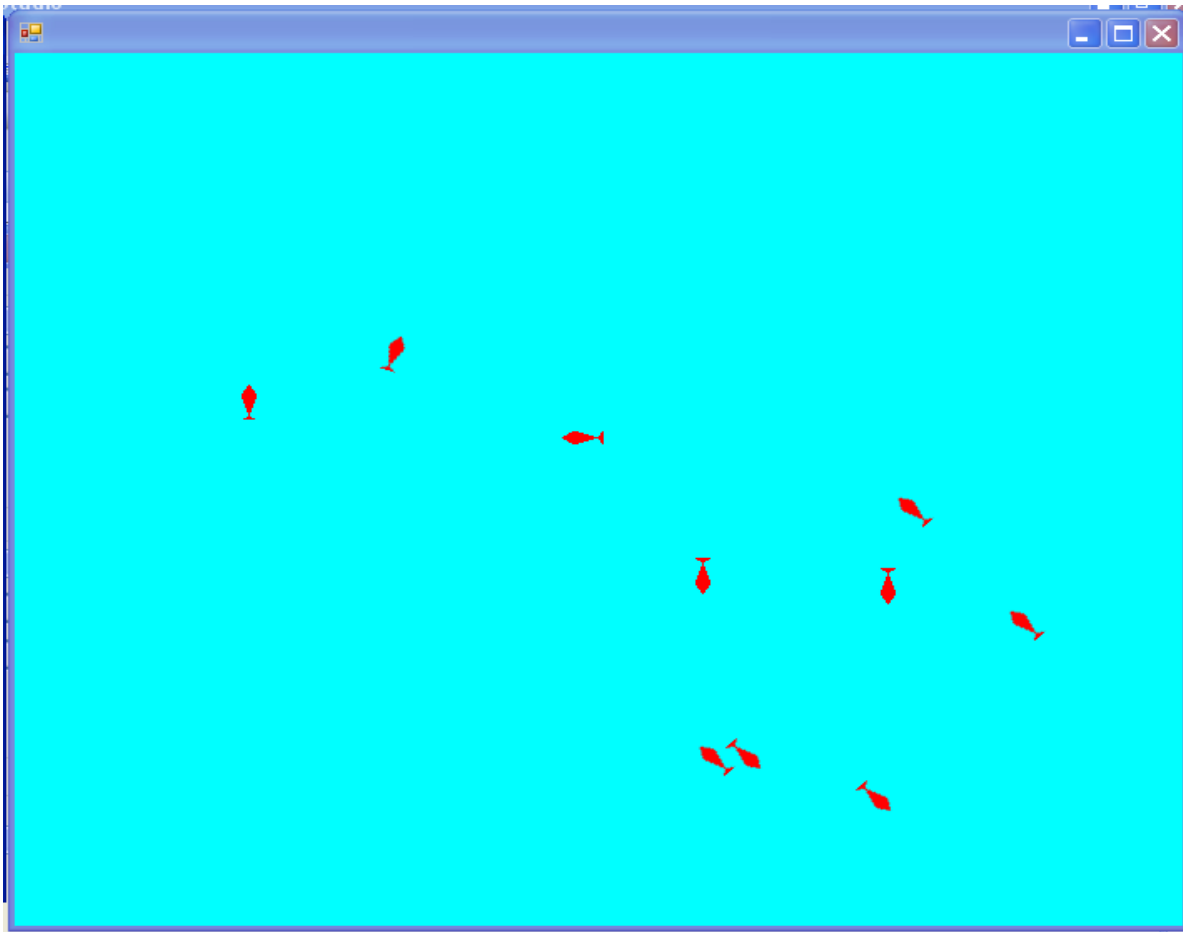


Figure 8 Sharks

Fish are the middle key in this system, below a pic for them.

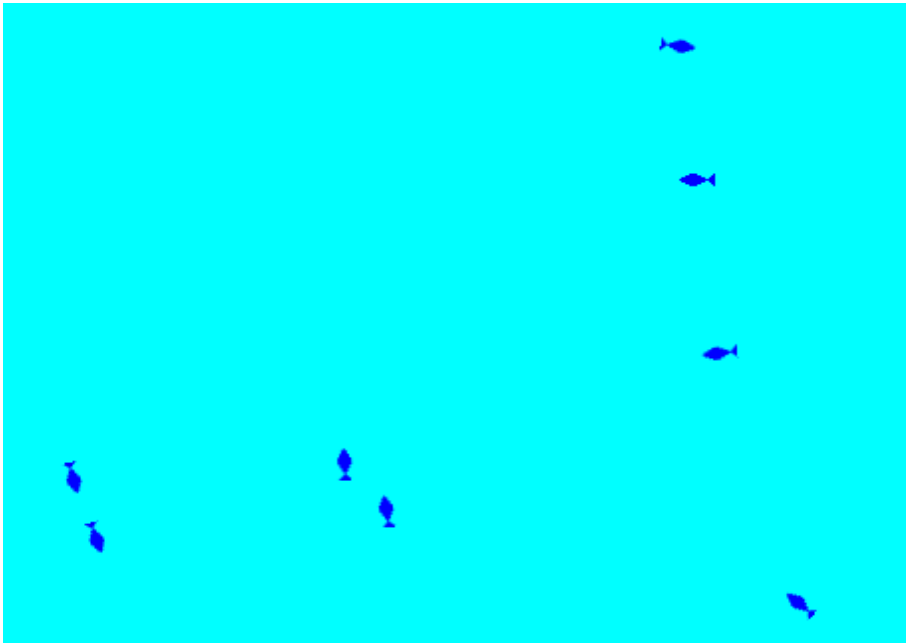


Figure 9 Fish

Plants are prey for fish to survive, play a role in this simulator.

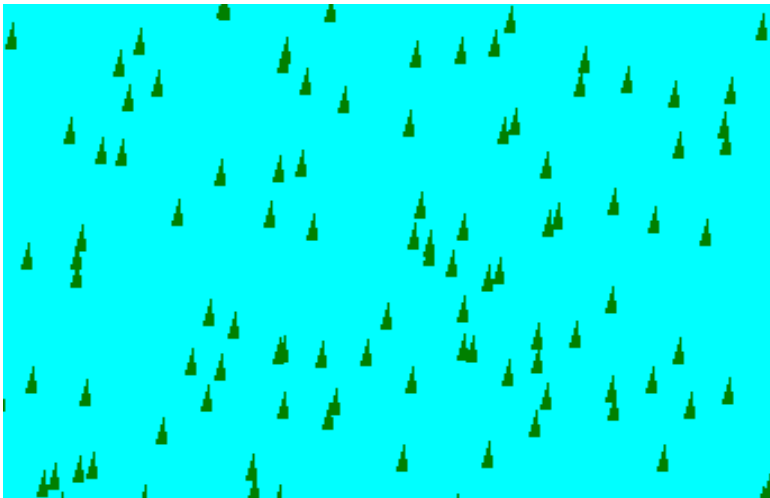


Figure 10 Plants

Schooling is a main part of this project, below visually is how it's viewed.

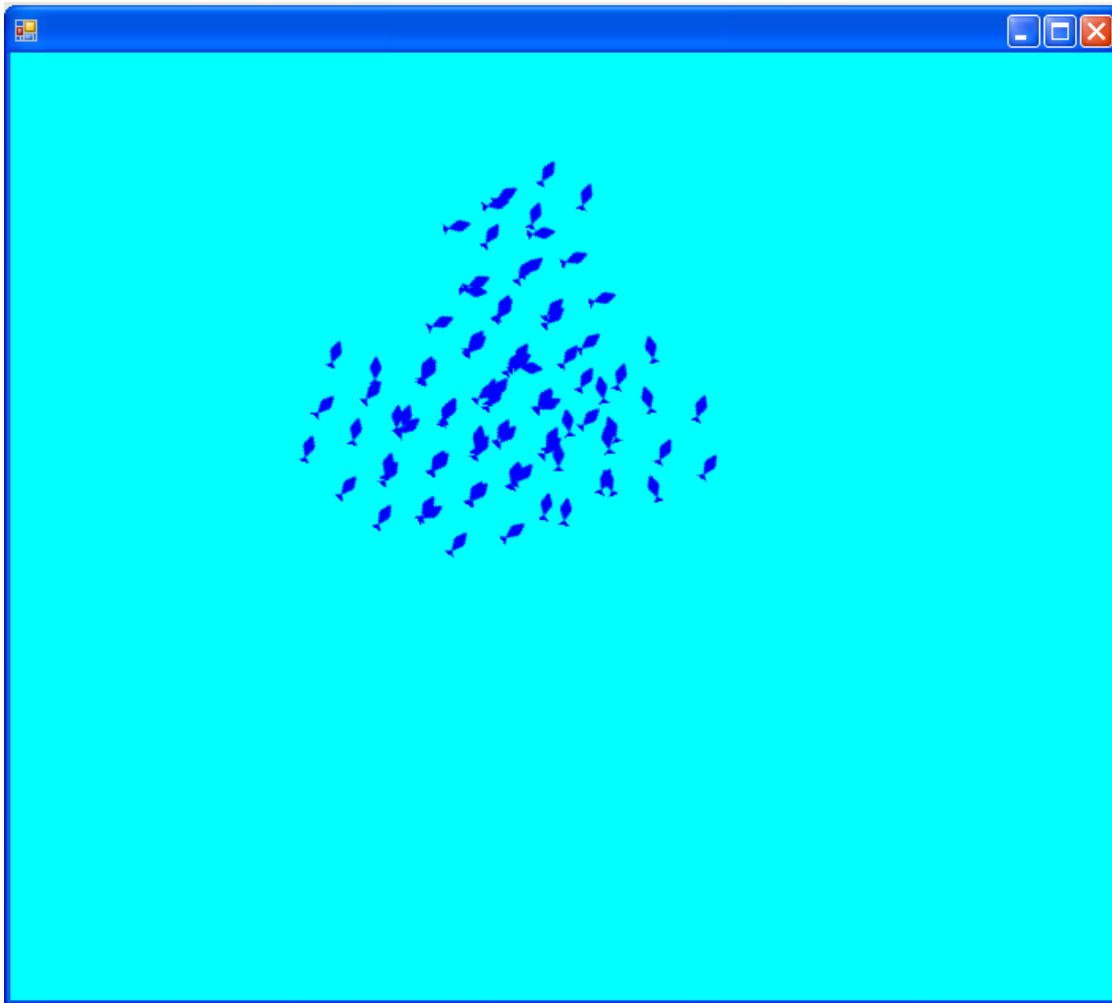


Figure 11 Fish schooling

Now fish eating.

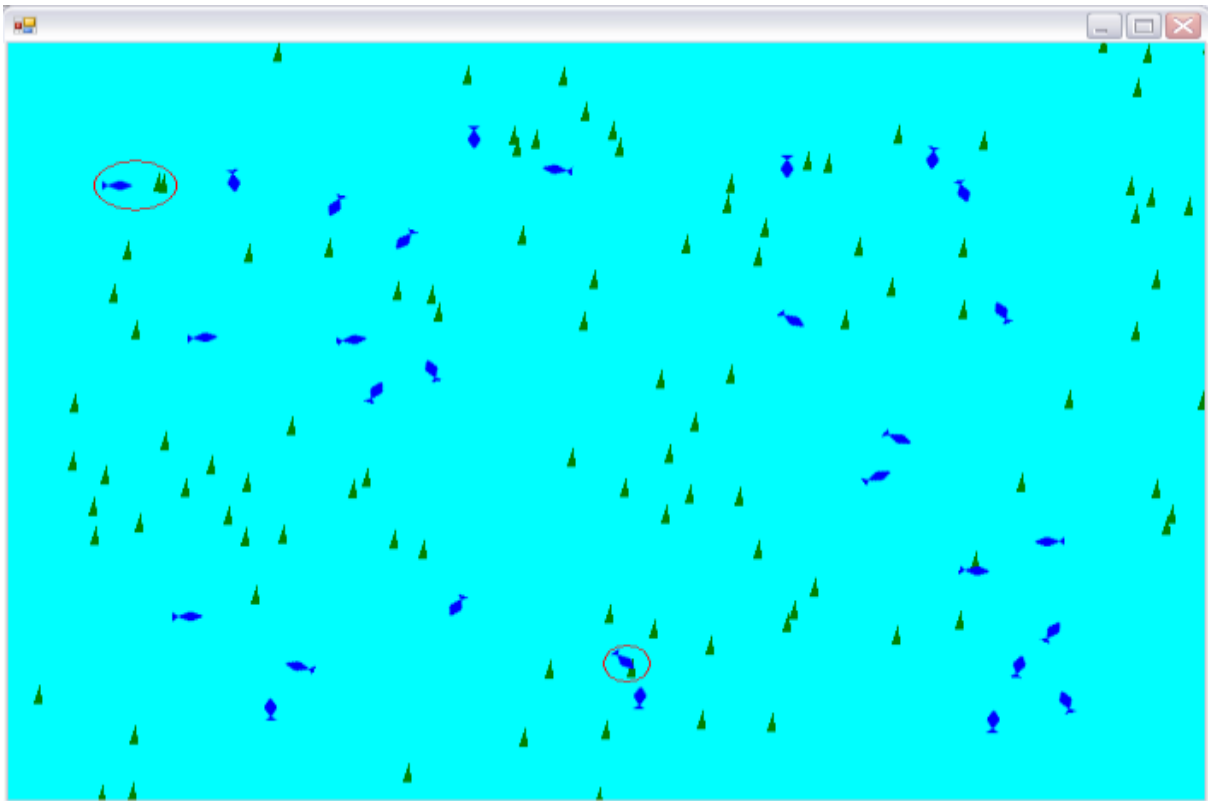


Figure 12 Fish eating

Sharks are predators for fish, below a screenshot showing them hunting.

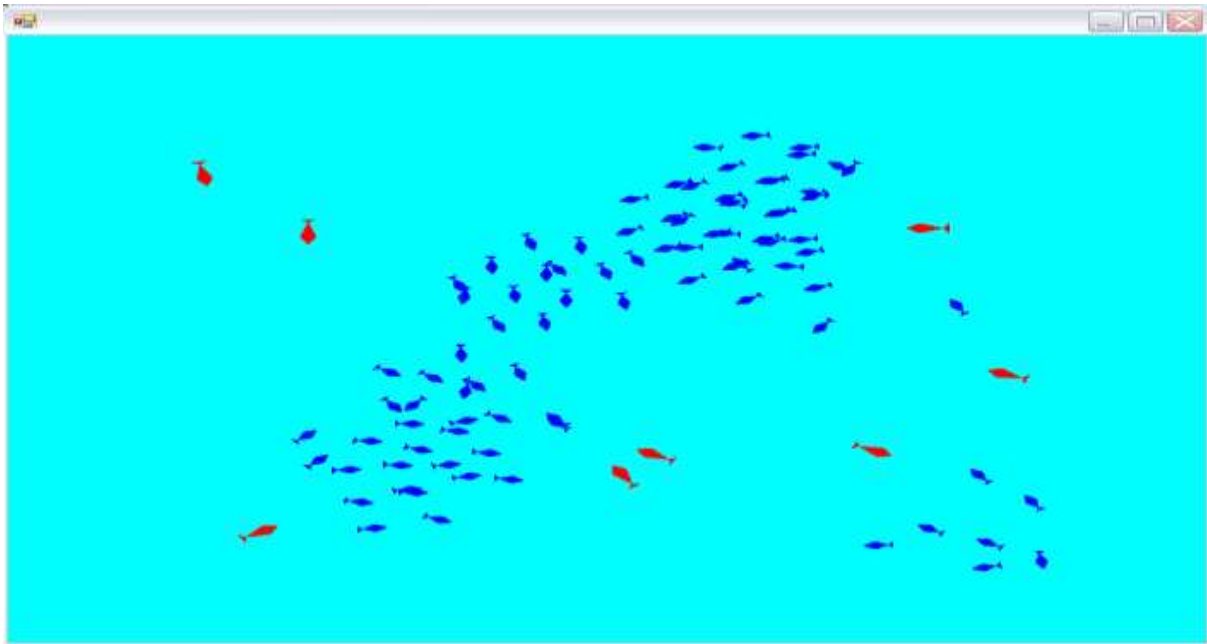


Figure 13 Sharks hunting, and fish in evading mode

Genetics play a main role too in this simulator, here's a screenshot of creatures with their distinct generations and speed.

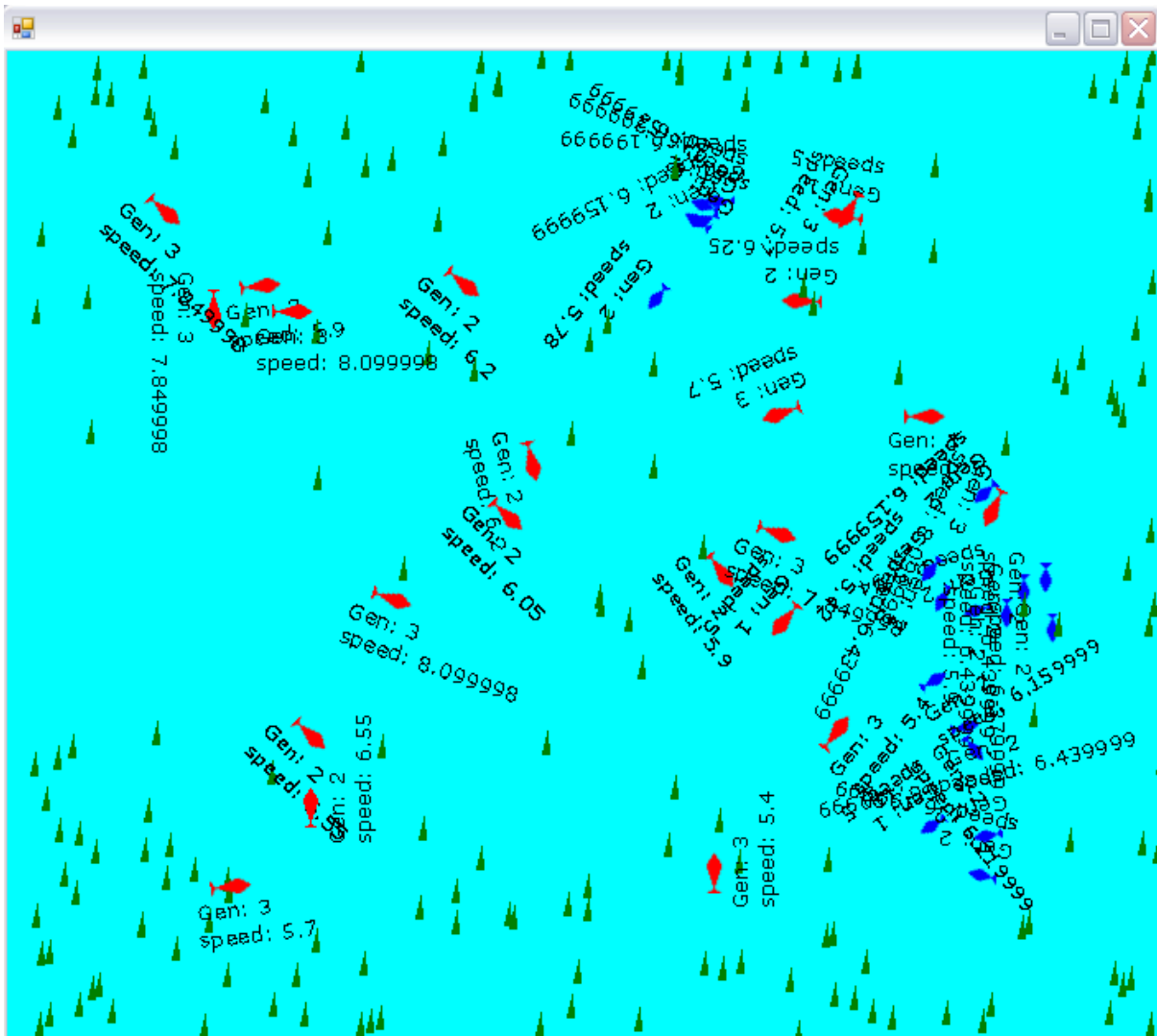


Figure 14 Different generations

A report tells us how many fish still alive, how many dead, and shows us a graph where we can track the evolution of generations by species.

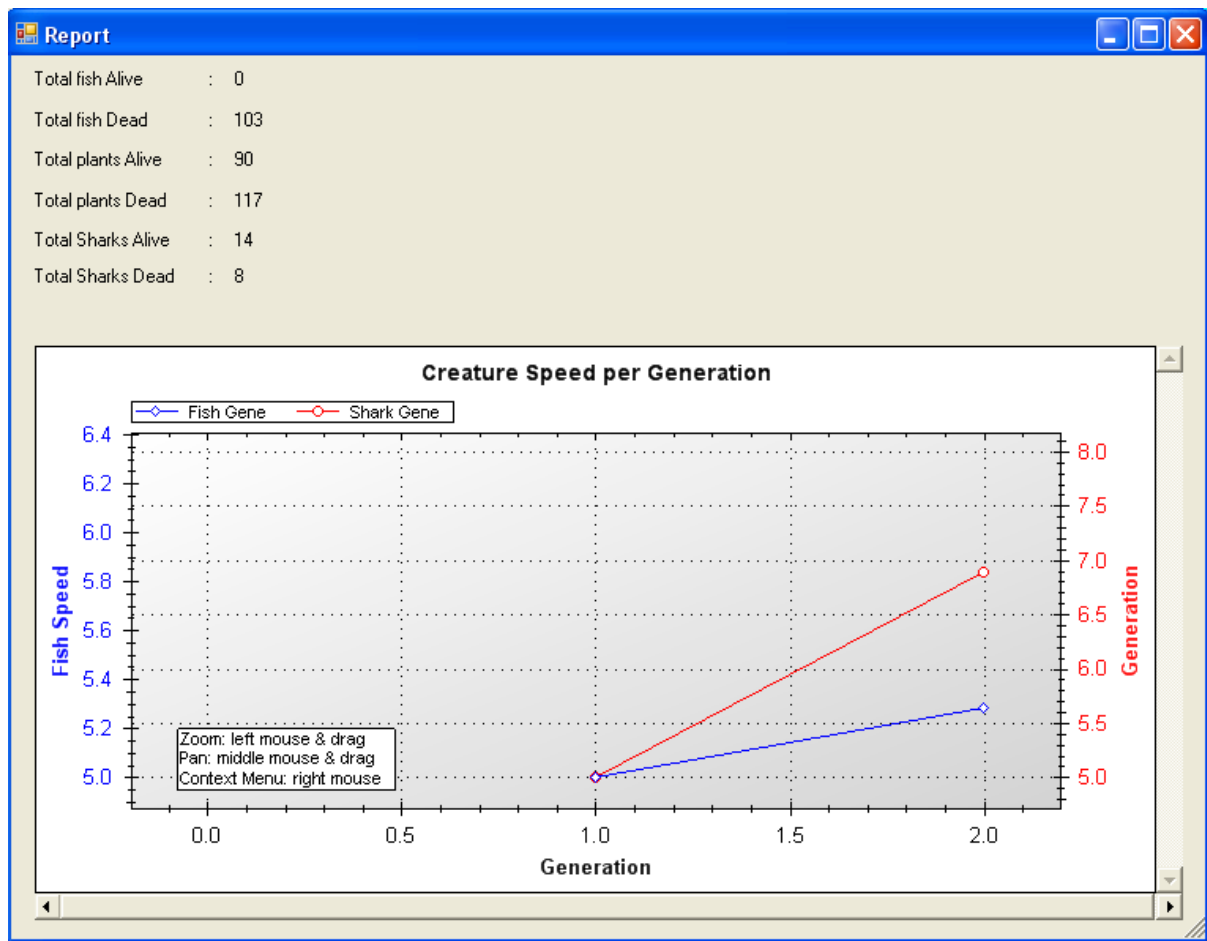


Figure 15 Report

Chapter 4

Conclusions

4.1 Summary of The main result:

The Main result of our senior project is a simulator of an aquatic ecosystem containing different species with different characteristics. It can be used by environmental scientist and aquatic research centers and the world blue peace organizations to simulate that aquatic ecosystem with a certain number of instances for each specie, in particular our project include shark, fish and plants, we can observe their behavior and understand how they should be balanced in a certain way to have a continuous environment without extinction of any one of them. This is done through trial and error from this part of the simulator and to help them comprehend the concept of the survival of the fittest.

4.2 Main contribution of this project:

To contribute in helping many countries preserve their aquatic ecosystems. And stop the extinction of various aquatic life forms, many of these life forms may be useful for human beings. And it can be a huge step in aiding certain concerned companies to try and find the correct balance of these species and balance is a key to helping this very rich resource to continue in a normal way.

4.3 Possible extensions and future work:

Our Project is an introductory version of simulator bases on the little information that we were able to acquire concerning aquatic species and their characteristics. Our big goal from this simulator is to be able to create libraries containing every specie, with every detail and characteristic to include them in our project to give a more specific solution to this crisis that the aquatic life is facing. And we want the simulator also to be able to give an analysis about the exact number per specie to coexist without anybody being able to extinct the other. The analysis should be based on different criteria's the environment living that they live in the depth of the sea, how much salty the water should be and the different situations of the sea.