NEA Project - Boids Simulation

*By Dmytrenko Vadym*

*BEDFORD SCHOOL*

Contents

[Analysis 4](#_Toc161128927)

[Introduction 4](#_Toc161128928)

[Background 4](#_Toc161128929)

[User Identification 6](#_Toc161128930)

[Existing Systems 9](#_Toc161128931)

[First Existing System 9](#_Toc161128932)

[Second Existing System 11](#_Toc161128933)

[Interview with User 14](#_Toc161128934)

[Solution Requirements 17](#_Toc161128935)

[Problem Statements 21](#_Toc161128936)

[Modelling 21](#_Toc161128937)

[Prototype 22](#_Toc161128938)

[Design 23](#_Toc161128939)

[High-Level Overview 23](#_Toc161128940)

[Page Navigation 23](#_Toc161128941)

[Hierarchy Chart 24](#_Toc161128942)

[System Flowcharts 26](#_Toc161128943)

[User Interface 29](#_Toc161128944)

[31](#_Toc161128945)

[Internal Data Storage 31](#_Toc161128946)

[Class Diagram 32](#_Toc161128947)

[32](#_Toc161128948)

[Flocking Algorithm 32](#_Toc161128949)

[Cohesion Rule: 33](#_Toc161128950)

[Separation Rule 33](#_Toc161128951)

[Alignment Rule 34](#_Toc161128952)

[Additional Algorithms 34](#_Toc161128953)

[Escaping Predators 34](#_Toc161128954)

[Obstacle Avoidance 35](#_Toc161128955)

[Technical Solution 36](#_Toc161128956)

[Introductory Techniques List 36](#_Toc161128957)

[Code 38](#_Toc161128958)

[Start/Main Screen 38](#_Toc161128959)

[Tutorial Screen 39](#_Toc161128960)

[Simulation Screen 40](#_Toc161128961)

[Testing 65](#_Toc161128962)

[Evaluation 66](#_Toc161128963)

[Self-Evaluation 66](#_Toc161128964)

[End Client Feedback Evaluation 66](#_Toc161128965)

[Possible Improvements 67](#_Toc161128966)

# Analysis

## Introduction

A lot of students who are doing biology at the university will eventually come across a topic that involves the behaviour and movement of the animals. Some students may find this topic hard and boring. I will be creating a simulation of animals that are moving in a group to make students’ education more entertaining and efficient.

## Background

Most of the animals move in a group like fish or sheep and some prefer to stay alone like bears or sharks. Animal behaviour and movement of a single creature is very complicated which is very hard to program, however it is possible to make a simplified simulation of the animals that are in a group. Such a program will show the collective behaviour observed in nature, where individuals in a group move in a coordinated way without a central leader. Simulation will show in 2D how animals’ group themselves together and how they respond together to other factors that will be included: predators and obstacles. This will help ethologist (a scientist who observes animals in their natural habitats to understand their behaviours, interactions, and survival strategies) and students to understand how the behaviour of simple-minded animals like birds will work when they are in the group or so-called flock. Such behaviour is called flocking behaviour and can be seen in a lot of animals that prefer staying in groups like sheep, fish, birds, zebras etc.

This program will be made using flocking algorithm **(requirement 4.a, 4.b & 4.c)**, which is algorithm that groups animals together, forming a flock. Flocking algorithm is the simplest way of achieving complex flocking behaviour in a program and flocking behaviour is where animals move around in formation. The reason for forming flocks in nature is that it makes them safer. It’s difficult for a predator to distinguish one animal from the rest.

Originally, the word flocking came from the behaviour when birds are foraging or in a flight.

With these three simple rules, the flock moves in an extremely realistic way, creating complex motion and interaction that would be extremely hard to create otherwise.

Animals are going to be referred as “boid” which corresponds to a shortened version of “bird-oid object”, which refers to a bird-like object. Most commonly, flocking behaviour can be seen in birds, that’s why the word “boid” is named after them. Such simulation is called Boid’s Simulation which essentially means that this simulation will be showing how birds group themselves and move around.

**Flocking Algorithm is made out of:**

1. **Separation (showed in Picture 1) – which is where each boid (animal) is avoiding crowding neighbours (short range repulsion), creating distance between themselves. Private radius for each boid can be applied so if someone enters it, that boid must leave.**

[A picture containing circle, diagram, clock, line

Description automatically generated](https://en.wikipedia.org/api/rest_v1/page/mobile-html/File:Rule_separation.gif)

Picture 1: Showing Separation Rule applied for the green boid (triangle).

1. **Alignment (showed in Picture 2) – which where each boid has tendency to follow in same direction (steering towards average heading of neighbours) within a threshold distance radius for a green boid (triangle). Each vector of a boid is calculated and summed. Average vector is deduced where all other boids steer towards.**

[A picture containing circle, origami, design

Description automatically generated](https://en.wikipedia.org/api/rest_v1/page/mobile-html/File:Rule_alignment.gif)

Picture 2: Showing Alignment Rule applied for the green boid (triangle).

1. **Cohesion (showed in Picture 3) – which where each boid has tendency to stick together within a threshold distance radius. Centre of mass is calculated, and green boid is attracted toward it.**

[A picture containing circle, diagram, line, origami

Description automatically generated](https://en.wikipedia.org/api/rest_v1/page/mobile-html/File:Rule_cohesion.gif)

Picture 3: Showing Cohesion Rule applied to a green boid(triangle).

Here is an example of birds being in a flock (Picture 4). Important notice: the simulation won’t be producing triangle shape like structure of foraging birds in a picture, the movement of the boids will be more complex, but predictable as it will be following those rules that are mentioned above.

A group of birds flying in the sky

Description automatically generated with medium confidence

Picture 4: Birds flocking together in the sky.

## User Identification

There are a lot of people that will benefit from my program, for example:

1. Engineers in robotics where simulation can be used to program swarms of robots who exhibit coordinated behaviours, allowing for efficient exploration of unknown environments, search and rescue operations, or collaborative tasks. For swarms of drones as well. Flocking algorithm can be employed to control swarms of drones. This can be useful in applications such as surveillance, agriculture, and disaster response, where multiple drones need to coordinate their movements to do specific tasks.
2. Students or scientists that study ecology, where boid simulation is applied in ecological studies to model the collective behaviour of animals. This can help researchers understand how animals move and interact in groups, leading insights into migration patterns, foraging behaviours, and predator-prey relationships.
3. For teachers and professors in education and training. Boid simulation can be incorporated into educational tools to teach concepts related to collective behaviour, artificial intelligence, and complex systems. They provide an engaging way for students to understand emergent behaviours in dynamic systems.

From developing such a program, a lot of fields would benefit from it, but I specifically want to focus on area 2 and 3, as the program would involve predators and obstacles which are not needed for programming drones or robots. I would like to concentrate on that specific area as a student myself, it is related to me and a chance to explore different fields of science like biology. So, my users would be university biology students and professors that would be using this program to make their learning and teaching more innovative. They will be using this simulation in a research field, for investigating how animals would behave using flocking algorithm. For different animals that are in the groups, will have different parameters for each flocking rule (separation, alignment and cohesion ), so the researchers would be changing those parameters in order to make their study more non-bias. It will be hard investigating how for example sheep flock themselves together, but by using flocking simulation (Boids Simulation), it would be more efficient for the scientist.

Proof of students studying animal behaviour and movements (Showed in Picture 5 and 6)

A screenshot of a computer

Description automatically generated

Picture 5: Animal Behaviour BSc degree, module details taken from Exeter University ([link](https://www.exeter.ac.uk/study/undergraduate/courses/ecology/animal/#course-content)).

A screenshot of a computer

Description automatically generated

Picture 6: Animal Behaviour BSc degree, another module degree taken from Exeter University ( [link](https://www.exeter.ac.uk/study/undergraduate/courses/ecology/animal/#course-content) ).

## Existing Systems

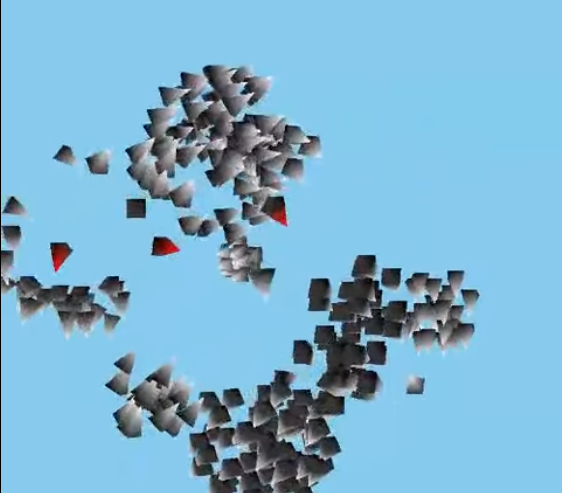
### First Existing System

The first simulation ever to exist to demonstrate flocking behaviour was developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds and is called Boids Simulation. Boid simulation is an example of emergent behaviour; that is, the complexity of Boids arises from the interaction of individual agents (boids or triangles). As mentioned before this program uses same rules **(requirement 4.a, 4.b & 4.c)**:

1. Separation: steer to avoid crowding flock mates.
2. Alignment: steer towards the average heading of local flock mates.
3. Cohesion: steer to move towards the average position (centre of mass) of local flock mates.

Later flocking simulation advanced the model by incorporating the effects of fear and adding the change of leader rule, making model more realistic.

Picture 7 shows screenshot of the Boids Simulation that was developed by him. Simulation is 3 dimensional and involves 3 predators that are in red, and other normal (friendly) boids that are coloured in grey.



Picture 7: Screenshot of Boids Simulation developed by Craig Reynolds

Such existing system has clearly shown how boids move together and how they escape from predator.

Advantages of such system:

1. 3-Dimensional GUI, which makes the program more convincing, but a bit confusing as it doesn’t clearly represent depth of each boid.
2. Has predators, makes program more realistic **(requirement 3.e**.**x & 3.e.xi)**
3. Has realistic group behaviour. Boids simulations effectively emulate natural flocking behaviours, displaying cohesive group movement, collision avoidance, and alignment among entities.
4. Has simple rules. The Boids algorithm relies on straightforward principles of separation, alignment, and cohesion **(requirement 4.a, 4.b & 4.c)**. Its simplicity facilitates easy implementation and comprehension, even for those without extensive expertise in graphics or simulation.
5. Has good scalability. Boids simulations scale efficiently to handle a large number of entities, making them suitable for simulating diverse scenarios like crowds, schools of fish, or swarms of insects. The algorithm's efficiency supports the simulation of intricate collective behaviours.
6. Has good adaptability. Boids simulations can be tailored to represent various entities and behaviours. Whether mimicking birds, fish, or human crowds, the fundamental principles can be adjusted to fit different contexts.
7. The obvious one, it is real-time applicable. Boids simulations are commonly employed in real-time applications, such as video games or interactive simulations. The algorithm's efficiency allows for dynamic and responsive simulations in these applications.

Disadvantages of such systems include:

1. It’s simplicity. While the simplicity of the Boids algorithm is an asset, it may also be a constraint. The model may not capture all the intricacies of complex flocking behaviours observed in real-world situations.
2. Lack of individuality. Boids treat entities as uniform agents following identical rules. This may not accurately mirror the diversity and distinct behaviours present in nature, where members of a flock may exhibit individual characteristics.
3. Limited Environmental Awareness. Boids typically focus on interactions among nearby entities and may lack advanced awareness of their environment. This can result in less realistic responses when entities need to react to external stimuli or obstacles.
4. Parameter sensitivity. Boids simulations frequently require parameter fine-tuning to achieve desired outcomes. Achieving the right balance between separation, alignment, and cohesion can be an iterative process. The behaviour of the simulation may be sensitive to changes in these parameters.
5. Limitations of Boids simulations. Although Boids simulations are effective in scenarios such as flocking, they may not be the best option for simulating other complex behaviours, such as predator-prey interactions or intricate social structures. So if I want to add predator-prey simulation, other algorithm would be recommended to be used.
6. Lack of settings. As seen in this program, you can’t change the strength of each rule, for example making each boid more separate by increasing separate rule. So adding extension to the program so that user can change settings itself would be useful.
7. Lack of description. Third party user may have no idea what’s the program about. Adding description of how it works I believe will be essential, but the decision will be made after having an interview with my user.
8. Same environment. This program doesn’t allow to change the background of it, making it more monotone.

To summarize, Boids simulation provides a straightforward and adaptable method for modelling flocking behaviour. However, they have limitations regarding realism, individuality, environmental awareness, parameter tuning, and suitability for different scenarios.

### Second Existing System

Second existing system was found in this website: [https://boids.cubedhuang.com](https://boids.cubedhuang.com/) which was made by Daniel Huang (*Picture 8, 9 & 10*). It is more modern compared to the previous one.

A screenshot of a computer screen

Description automatically generated

*Picture 8:* Screenshot of second existing system.

A screenshot of a computer screen

Description automatically generated

Picture 9: Second Screenshot of second existing system.

A map of different colored triangles

Description automatically generated

**Picture 10: Third Screenshot of second existing system.**

Advantages:

1. Nice User Interface.
2. Has settings, user can change system if he wants. Each setting has appropriate name, user understands what happens if he changes each section.
3. Has a lot of different diverse settings. On top of having basic rule settings ( alignment force, cohesion force and separation force ) options like max / min speed, movement randomness, drag, steering force, alignment bias, movement accuracy and boid vision are added, which makes program very realistic, especially drag which simulates air/water resistance.
4. Has option to reset settings back to normal, which is very useful in my opinion **(requirement 3.e.vi)** as it will be hard for user to remember everything he/she changed throughout the usage of program. Of course, program can be closed to achieve that, but it will require more time.
5. Has option to hide menu for the user to focus on the simulation itself.
6. Very detailed through colorization of each boid, which represents the speed at which it travels making it livelier. Red represents boid being stationary, and then the lighter the colour ( red -> orange -> yellow -> green -> cyan ) the faster it moves.
7. Having an ability to change the number of boids.
8. Having an ability to import and export settings of the simulation, thus other people can experience same settings as the other one, if he sends those settings to him.
9. An ability to freeze the simulation which I again find very useful **(requirement 3.d.ii & 3.d.iii)**

Disadvantages:

1. Having same monotone background, no ability to change it.
2. No ability to add obstacles, predators.
3. No description of what this simulation represents.
4. Too many settings can confuse third-party user.
5. 2-Dimensial, makes it unrealistic.

Overall, this system is much more developed compared to the system that was developed by Craig Reynolds. After doing such research, it helped me to realize how the program should be made and what bits can I add or remove.

After observing 2 existing systems, it helped me to realise how I want my system to be structured. In addition it helped me to understand what requirements for my system I want to add.

## Interview with User

I have decided to take an interview with my identified user, so that I could make requirements for my project, and understand what exactly are they looking for. Before having an interview, I showed them 2 of the Existing Systems that I researched earlier ( *First Existing System & Second Existing System* ).

*Question 1:*

* **Do you know what Boids Simulation or Flocking Algorithm is? If no, would you like to have a description of what it is, or do you think it will be useful to have one?**

A: In my opinion it is very useful for students, as some may not know what Boids simulation or flocking algorithm is **(requirement 2.c, 2.d, 2.e, 2f)**. For people who are not into programming, it might be difficult to understand, so it will be good to add a small quick tutorial or explanation so that if people don’t know stuff about that, have the ability to learn it through the use of this program.

*Question 2:*

* **What specific goals or outcomes you are hoping to achieve by using the Boid simulation?**

A: I think it will be very useful to module animal behaviour, as students we can’t observe everything in real world physically as it requires time and money, we can’t see how fish react to a shark in real world. Such simulation will help us learn and visualize the group behaviour in such simplistic way, help us understand the implications of group behaviour.

*Question 3:*

* **Are there any particular aspects of the simulation that are crucial for your intended use?**

A: To be able to adjust the number of predators **(requirement 3.d.viii)**, boids itself **(requirement 3.d.vii** and add obstacles in the way of the flock like a wall **(requirement 3.e.iv)** that will block the flock, and to see how the flock will react when they face them, how will they deal with them? Rules like alignment, cohesion and separation as different animals behave differently. Adding a restart simulation button **(requirement 3.d.vi)** so that it will reset everything back to normal – will be quite useful as after playing with it for a while it can get too confusing.

*Question 4:*

* **How comfortable are you with the current user interface and controls of the Boids simulation?**

A: The user interface is minimalistic and straightforward, and I am quite satisfied with it. Additional different background that imitates environment is not needed as we are students, we just want to see the behaviour of the animals, making background colourful in my opinion is a bit childish, I am more for functionality.

*Question 5:*

* **Are there any specific features or controls you would like to see added or modified to enhance your interaction with the simulation?**

A: Adding the ability to see when normal Boids interact with predators by perhaps changing its colour? It would make the observation easier **(requirement 3.e.iii)**.

*Question 6:*

* **How important is the ability to customize parameters within the Boids simulation to meet your requirements?**

A: In my opinion it’s a key element of the simulation as it allows me, a student to change the group behaviour according to my needs and according to the animal I am observing **(requirement 3.d.ix, 3.d.x, 3.d.xi & 3.d.xii)**, allows me to add obstacles and customize the environment **(requirement 3.d.xiii)** at which my simulation is moving in.

*Question 7:*

* **Are there specific parameters or settings that you would like to be able to adjust to better suit your needs?**

A: Not really, I would like to adjust everything as it will make program more concise.

Specifically, the 3 main rules: cohesion, alignment and separation. Also as stated before, number of boids and predators **(requirement 3.d.vii & 3.d.viii**).

*Question 8:*

* **How helpful do you find the current visualization features in understanding the behaviour of the boids?**

A: I find it very helpful but in my opinion there are too many different colours and I don’t know the reason for that, makes it to complicated ( my eyes hurt ) just make them all single colour or if making with different colour add explanation for it so that other third-party user understands that. Of course, if making same colour, then for normal boids one colour for predators another **(requirement 3.e.i)**.

*Question 9:*

* **How responsive do you expect the simulation to be, especially when dealing with a large number of Boids?**

A: I want to have adjustable number, I want to focus on one group of animals, between 20 to 200 number of boids, too little will make it biased and too much will make it too packed. For predators I think between 0 to 5 will be good **(requirement 3.d.vii & 3.d.viii)**.

After doing interview, it helped me realize on what I should focus my program on and what requirements I should make.

## Solution Requirements

1. When program is loaded, a Main Screen will appear, where there:
   1. Main Screen is loaded in the centre of the screen.
   2. Have Main Screen that opens when program is started.
   3. Have an ability from Main Screen to open Tutorial Screen trough a “Tutorial” button click.
   4. Have an ability from Main Screen to open Simulation Screen through a “Play” button click.
   5. Have an ability from Main Screen to close/shut down the program through an “Exit: button click.
2. For Tutorial Screen, that will open if “Tutorial” button is clicked through the main screen:
   1. When the Tutorial Screen is opened, it has to open in the same location as the Main Screen was closed.
   2. Main Screen has to be closed – no overlapping of 2 forms.
   3. Have a description of what is Boids Simulation.
   4. Have a description of rule “Separation”.
   5. Have a description of rule “Cohesion”.
   6. Have a description of rule “Alignment”.
   7. Have a button that will close Tutorial Screen and open Main Screen.
   8. When “Exit” button is clicked, Main Screen form must be opened in the exact same location as Tutorial Screen was closed.
   9. Have a description of the “Vision”.
   10. Have description of “Wrapping” and “Not Wrapping”.
   11. Have a description of what happens when you interact with the system.
   12. There shouldn’t be overlapping of 2 forms, when “Exit” button is pressed in the Tutorial Screen – Tutorial Screen has to disappear and at the same time Main Screen has to appear.
3. When “Play” Button is pressed from the Main Screen, opens the Simulation Screen where the Boids Simulation will run.
   1. When the Simulation Screen is opened, it has to open in the same location as the Main Screen was closed.
   2. Main Screen has to be closed – no overlapping of 2 forms.
   3. Splitting Simulation Screen into 2 sections: Simulation section where the simulation will run and the Settings section where a user will be able to

adjust parameters.

* 1. On Setting Section, be able to change parameters:
     1. Have an ability to exit Simulation Screen by pressing “Exit” button that will direct to the Main Screen.
        1. When “Exit” button is clicked, Main Screen form has to be opened in the exact same location as Simulation Screen was closed.
        2. There shouldn’t be overlapping of 2 forms, when “Exit” button is pressed in the Simulation Screen – Simulation Screen has to disappear and at the same time Main Screen has to appear.
     2. Have an ability to start simulation through a click of “Start” button.
        1. If Start button is pressed, change its text to Stop.
     3. Have an ability to stop simulation through a click of “Stop” button.
        1. If Stop button is pressed, change its text to start.
     4. Have an ability to make boids go through the one side of the screen and make them appear on the opposite side of the screen by default in real-time while program is running. Such button will be called “Wrapping”.
        1. If button is clicked, changes its text to “Not Wrapping”.
     5. When button “Wrapping” is clicked and its text is changed to “Not Wrapping”, it makes boids bounce back of the edges of the Simulation Section in real-time while program is running.
        1. If button is clicked, changes its text to “Wrapping”.
     6. Have an ability to restart simulation, which makes both boids and predators respawn. All the rules are go back to normal: number of boids, number of predators, Cohesion strength, Alignment strength, Separation strength, Boids and Predators Vision and Obstacle Size in real-time while program is running.
     7. Have an ability to adjust the number of boids (between 20-200) and when the Simulation runs start at a 100 as a initial value in real-time while program is running.
     8. Have an ability to adjust the number of predators (between 0-5) and when the Simulation runs start at a 5 as a initial value in real-time while program is running.
     9. Have an ability to adjust the strength of Cohesion rule in real-time while program is running.
        1. When Cohesion rule is adjusted, number should correspond to the trackbar value.
     10. Have an ability to adjust the strength of Alignment rule in real-time while program is running.
         1. When Alignment rule is adjusted, number should correspond to the trackbar value.
     11. Have an ability to adjust the strength of Separation rule in real-time while program is running.
         1. When Separation rule is adjusted, number should correspond to the trackbar value.
     12. Have an ability to adjust the vision of the Boids and Predators – how far will both of them see. Predators must see 2x more in radius than normal boids.
         1. When Vision rule is adjusted, number should correspond to the trackbar value.
     13. Have an ability to adjust Obstacles Size (from 10-500). Initial size is 50.
  2. On Simulation Section:
     1. Represent normal Boid and predator with 2 different colours. Normal boid = blue & Predator = Red.
     2. Represent Boids and predators as some sort of geometrical shape.
     3. When Boids interact with predators, represent them with different colour (yellow) which will be represented as “panic mode”.
     4. Add the ability to add obstacles by mouse clicking inside Simulation Section.
     5. Add the ability to remove the obstacles by mouse clicking on the obstacle.
     6. When boids are loaded in the beginning of the simulation, make them that that they don’t intersect between themselves, adds realism to the simulation.
     7. Apply Flocking Rules for every Boid.
     8. Apply the Predator rule for every Predator.
     9. Apply the avoiding behaviour of obstacles for both Predators and Boids.
     10. Has boids (100 when loaded).
     11. Has predators (5 when loaded).
     12. Have an ability to add a tail/past projectory to the Boid and Predator which will appear at previous movements of the Boids and Predator – as it will make the observation of the current and the past direction easier in my opinion.
     13. Have an ability to add radius of vision for Boid and Predators, so that it would be easier to observe the radius of action – when normal boids start to run away from predators.
     14. Represent Obstacles as a hexagon shape.

1. Flocking Rules:
   1. Implement Alignment rule for each boid. Part of flocking algorithm so must be used to represent realistic animal group behaviour.
   2. Implement Cohesion rule for each boid. Part of flocking algorithm so must be used to represent realistic animal group behaviour.
   3. Implement Separation rule for each boid. Part of flocking algorithm so must be used to represent realistic animal group behaviour.
   4. Implement Predators rule, so that Predator goes after the normal boids when spotted.

## Problem Statements

After doing research and analysis as well as interview with my user, I realized some of the limitations.

Firstly, the system I am making will be 2D, not 3D, so it won’t be as realistic as other programs that are more complicated but on the other hand, it’s simplicity will help to understand the flocking algorithm.

Secondly, the graphical aspect of the boids or predators, they will be presented as simple geometrical shapes with different colours, some may would like to have a cool 2d design for boids and for predators. But it could be debated that it is not essential part of the program, and the main purpose of the program is to represent simulation’s function, but adding specialised designs for boids will be time consuming and unnecessary, as users of my program will be educated adults.

Thirdly, even though my program will be realistic and representative, not including aspects like wind ( air resistance ) or water resistance ( drag ) may adjust some behaviours of the animals. Not having them may make seem my simulation as biased.

Fourthly, not including some more natural complicated rules like part time leader of the flock, which basically means that leader ( main boid ) will be found based on some calculations for some period of time and his movements will be observed by the rest of the flock. This rule could add some realism.

## Modelling

The flowchart below *( Picture 11 )* shows the specific problems and requirements of the system that I am making.

A diagram of a diagram

Description automatically generated

*Picture 11: Modelling of the system*

## Prototype

Below is a prototype of a Simulation Screen *( Picture 12 )*. When program is loaded, it solves requirement **3.e.vi** , thus when program is loaded, boids are not overlapping. Has an ability to start/stop simulation **(requirement 3.d.ii & 3.d.iii)**, but when simulation is running, boids are just moving downwards. Now I have to add more settings and add the mathematical rules to achieve mathematical behaviour as well as adding obstacles and predators. Furthermore, Exit button has been added, which when pressed, sends the user to the Main Screen **(requirement 3.d.i).** Wrap Around button doesn’t work yet.

A screenshot of a computer

Description automatically generated

*Picture 12: Initial Prototypes*

# Design

## High-Level Overview

Before starting to create the simulation itself, I would need to plan out the overview of the system and split a large problem into smaller chunks, in order to ease the process of programming and to have a clear vision of what the system is going to look like.

### Page Navigation

As discussed earlier, my program will have 3 screens: First screen, the one that shows up is Main Screen. After that, through controls, user have a decision to either open Tutorial Screen – where a description of what program does will show up or to open Simulation Screen – where the Boids Simulation will run. User will be able to navigate easily between those 3 screens and if the user decides to exit application, the only way will be through the “Exit” button at the Main Screen *( shown in picture 13 ).*

A pink sign with black text

Description automatically generated

*Picture 13: Page Navigation.*

### Hierarchy Chart

To demonstrate how the system works together, I have decided to split a large problem into numerous subroutines which are linked together, creating a hierarchy chart. It will help me visualise the abstract concept of the simulation and will be my reference point during the programming stage.

Firstly, this Hierarchy chart *( Picture 14 )* shows how my Simulation Screen work. It will be split in 2 sections: Simulation itself and Settings Side **( requirement 3.c** **)**.

A diagram of a computer program

Description automatically generated

*Picture 14: Simplified Main Hierarchy Chart.*

For Settings Section *( Picture 15 )*, I broke down all the controls of the Simulation Screen that the user has requested ( **requirement 3.d ).**

A diagram of a system

Description automatically generated

*Picture 15: Hierarchy Chart Settings Section.*

For the other half of the screen where the simulation itself happens, I designed another hierarchy chart specifically when the program is loaded *( Picture 16 )* **( requirement 3.e.vi )***.*

A white rectangular sign with black text

Description automatically generated

*Picture 16: Loading of the Simulation Screen where the Boids Simulation happens.*

If the user presses “Start” on the Settings Side of the Simulation Screen – simulation starts. Hierarchy Chart for when that happens *( Picture 17 ).* **Requirements 3.e.iii , 4, 3.d.iv , 3.d.v , 3.e.xii and 3.e.xiii.**

A diagram of a process

Description automatically generated with medium confidence

*Picture 17: Hierarchy Chart when simulation Is running.*

### System Flowcharts

I want to demonstrate specific subroutines and major stages of the game in more detail through the use of system flowcharts. As I demonstrated already a simple flowchart of my system in the Modelling Section from Analysis. I cannot specify anymore from Tutorial Screen and the Main Screen as their structure is pretty simple. *Picture 18* shows more detailed flowchart.

A diagram of a simulation

Description automatically generated

*Picture 18: More complex flowchart diagram of Simulation.*

For Predator and Boid, they both will have different vision (**requirement 3.d.xii**). Flowchart diagram below will ( *Picture 19 )* will show the steps of meeting this requirement.

A diagram of a process flow

Description automatically generated

*Picture 19: Flowchart diagram of giving different vision parameters for different objects.*

For adding obstacles on the screen, user has to press with his mouse on the Simulation Screen where the Boids Simulation is happening. *Picture 20* shows this process.

A diagram of a computer program

Description automatically generated

*Picture 20: Flowchart diagram of adding and removing an obstacle.*

## User Interface

My program heavily relies on GUI as it requires real time animation of the Boids flocking and Predators going after them.

When the program gets loaded. It starts at the Main Screen *Picture 21.*

*A diagram of a software development

Description automatically generated with medium confidencePicture 21: Main Screen*

Tutorial Screen *( Picture 22 )*

*A close-up of a diagram

Description automatically generatedPicture 22: Tutorial Screen*

Simulation Screen *( Picture 23 )*:

## Internal Data Storage

Users input – changing parameters of the simulation will be stored in a variable. Each trackbar will have their own variable.

In order for my program to remember the coordinates of the Boids and Predators I have decided to use a Queue as it perfectly fits my needs. I need this abstract data structure as it is First In First Out.

Example of queue operation (number in square brackets represents position index inside the queue):

A black and white logo

Description automatically generatedA picture containing text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| [0] {1078, 679} | [1] {1075, 678} | [2] {1071, 676} | [3] {1068, 674} |

This way, new coordinates are going to be enqueued to the back of the queue, and the oldest one’s in the start of the queue.

Let’s say we have to add new coordinates to the queue:

A picture containing text

Description automatically generatedA black and white logo

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
|  | [1] {1075, 678} | [2] {1071, 676} | [3] {1068, 674} |

If the Queue size is bigger than our variable we dequeue and then we add new element. Process repeats.

|  |
| --- |
| Queue<Point> HistoryOfPositions = new Queue<Point>(); // Initialises new Queue  HistoryOfPositions.Enqueue(new Point((int)XCentrePosition, (int)YCentrePosition));  while (HistoryOfPositions.Count > NumberOfHistoryPositions) // Deletes if exceeds the variable  { HistoryOfPositions.Dequeue(); } |

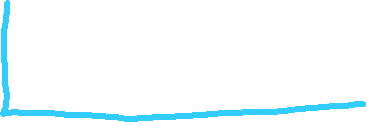
## Class Diagram

Star sign next to a variable/method name means it is protected.

Lock sign next to a variable/method name means it is private.

This picture below is a class diagram but without fields of the MainSimulationFormA screenshot of a computer

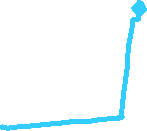
Description automatically generated



This picture below is a class diagram but without methods for MainSimulationForm

A screenshot of a computer

Description automatically generated



## Flocking Algorithm

To create a realistic flocking/grouping behaviour of the animals. I will be using Flocking Algorithm which is made from 3 rules.

### Cohesion Rule:

Cohesion Rule is based on calculating the centre of mass for the group. The aim of the algorithm is to bring Boids together by finding the centre of mass.

*Pseudocode for Cohesion Rule:*

|  |
| --- |
| CohesionBoids = FindBoidsWithinVision(BoidList, Vision) // Find boids within vision range to apply cohesion  SumX = 0  FOREACH boid in CohesionBoids // Calculate the mean X position of the boids  SumX = SumX + boid.XCentrePosition  MeanX = SumX / Count of CohesionBoids  SumY = 0  FOREACH boid in CohesionBoids // Calculate the mean Y position of the boids  SumY = SumY + boid.YCentrePosition  MeanY = SumY / Count of CohesionBoids  // Adjust the current boid's velocity towards the mean position of the boids  this.XVelocity = this.XVelocity + (MeanX - this.XCentrePosition) \* 0.0005 \* CohesionStrength  this.YVelocity = this.YVelocity + (MeanY - this.YCentrePosition) \* 0.0005 \* CohesionStrength |

### Separation Rule

Separation Rule is based on separating each Boid between themselves as well as predators.

*Pseudocode for Separation Rule:*

|  |
| --- |
| FoundSeparations = FindBoidsWithinSeparationDistance(BoidList, VisionRepel) // Identify boids within a separation distance  // Calculate cumulative separation force on the X axis  SumSeparationX = 0  FOREACH predator in FoundSeparations  Distance = GetDistance to predator  SeparationForce = (XCentrePosition of this boid - XCentrePosition of predator) \* (VisionRepel - Distance)  SumSeparationX = SumSeparationX + SeparationForce  // Calculate cumulative separation force on the Y axis  SumSeparationY = 0  FOREACH predator in FoundSeparations  Distance = GetDistance to predator  SeparationForce = (YCentrePosition of this boid - YCentrePosition of predator) \* (VisionRepel - Distance)  SumSeparationY = SumSeparationY + SeparationForce  // Adjust the current boid's velocity based on the separation forces  this.XVelocity = this.XVelocity + SumSeparationX \* 0.005 \* SeparationStrength  this.YVelocity = this.YVelocity + SumSeparationY \* 0.005 \* SeparationStrength |

### Alignment Rule

Alignment Rule is based on finding mean vector of the group and sticking to it.

*Pseudocode for Alignment Rule:*

|  |
| --- |
| // Identify boids within the vision distance for alignment  FoundBoids = FindAllBoidsWithinVisionDistance(BoidList, Vision)  // Calculate the mean (average) velocity of these boids along the X-axis  SumXVelocity = 0  FOREACH in FoundBoids  SumXVelocity = SumXVelocity + boid.XVelocity  MeanXVelocity = SumXVelocity / NumberOf FoundBoids  // Calculate the mean (average) velocity of these boids along the Y-axis  SumYVelocity = 0  FOREACH in FoundBoids  SumYVelocity = SumYVelocity + boid.YVelocity  MeanYVelocity = SumYVelocity / NumberOf FoundBoids  // Adjust the current boid's velocity towards the mean velocity of the found boids  this.XVelocity = this.XVelocity + (MeanXVelocity - this.XVelocity) \* 0.03 \* AlignmentStrength  this.YVelocity = this.YVelocity + (MeanYVelocity - this.YVelocity) \* 0.03 \* AlignmentStrength |

## Additional Algorithms

### Escaping Predators

Escaping Predators algorithm aim is to allow boids to steer away from the predators when they get too close to them.

*Pseudocode for Escaping the Predators:*

|  |
| --- |
| FoundPredators = FindAllPredatorsWithinVision(BoidList, Vision)  // Calculate cumulative displacement from all predators along X and Y axes  SumPredatorX = 0  SumPredatorY = 0  FOREACH predator in FoundPredators  SumPredatorX = SumPredatorX + (XCentrePosition of this boid - XCentrePosition of predator)  SumPredatorY = SumPredatorY + (YCentrePosition of this boid - YCentrePosition of predator)  // Adjust this boid's velocity to move away from predators  this.XVelocity = this.XVelocity + SumPredatorX \* 0.002  this.YVelocity = this.YVelocity + SumPredatorY \* 0.002 |

### Obstacle Avoidance

Obstacle Avoidance is dealing in a section which is aimed to make Boids and Predators steer away from the Obstacle if they get too close.

*Pseudocode for Obstacle Avoidance:*

|  |
| --- |
| // Define the current position of this boid  PositionThis = CreatePoint with coordinates (XCentrePosition, YCentrePosition)  // Find all obstacles within a modified vision range  FoundObstacle = FindObstaclesWithinRange(ObstacleList, VisionObstacleMod)  // Calculate cumulative force to apply based on obstacles' positions and sizes  SumObstacleX = 0  SumObstacleY = 0  FOREACH obstacle in FoundObstacle  DistanceToObstacle = GetDistance from PositionThis to obstacle.Location  ObstacleInfluence = (obstacle.Size + VisionObstacleMod) - DistanceToObstacle  SumObstacleX = SumObstacleX + (XCentrePosition - obstacle.Location.X) \* ObstacleInfluence  SumObstacleY = SumObstacleY + (YCentrePosition - obstacle.Location.Y) \* ObstacleInfluence  // Adjust this boid's velocity based on calculated forces from obstacles  this.XVelocity = this.XVelocity + SumObstacleX \* 0.001  this.YVelocity = this.YVelocity + SumObstacleY \* 0.001 |

# Technical Solution

## Introductory Techniques List

|  |  |  |  |
| --- | --- | --- | --- |
| Technical Skill | Requirement contributed towards | Purpose | Methods used in |
| Object Orientated Programming, Inheritance, Polymorphism and Encapsulation | 3.e | A way of organising code. Makes it more compact as well as makes it easier to understand what is where. In addition, my program is more suitable to that style of coding. | Inheritance: Predator inherits from Boid.  Polymorphism: Overriding the FoundBoid and ColourUpdate methods in the Predator class to provide specialized behaviour.  Encapsulation: Using access modifiers (like public, protected, private) to control access to class members. |
| Usage of a Queue and queues operations as well as lists | 3.e | Queue: queue was needed in order to use it’s speciality of First In First Out data structure for memorizing Boids and Predators movement in order to add a tail for them.  Lists: one of the lists were used to keep the objects of both predators and boids and second list was used to keep the elements of obstacles | MoveLimit method |
| Exception handling |  | In Boid class, due to a lot of mathematics there was a possibility of X or Y velocity to become an infinite decimal. Exception handling has been added so that the program won’t crash. | MoveBoid method |
| Complex GUI | 1,2,3 | To create a functional, practical and user friendly device |  |
| Use of advanced mathematics |  | Using Pythagoras for calculating the distance between 2 objects, using Pythagoras for calculating speed of the object through the use of X and Y velocity and using sin and cos in order to draw hexagon as an obstacle | GetDistance, GetSpeed and CalculateSize |
| Use of C# Random Class |  | Used to generate random numbers in the specified range for generating X and Y positions for Boids and Predators when they were being loaded. | MainSImulationForm\_Load method |
| Usage of global variables, constants and local variables |  | Global variables: so that they can be accessed throughout the whole system.  Constant: for keeping immutable information.  Local variables: for holding data that is only needed inside the method | They are placed throughout the system |

## Code

### Start/Main Screen

|  |
| --- |
| **Requirement: 1** |
| Purpose: To provide a suitable Start/Main Screen for the user |
| Class: StartScreen |
| Techniques used:   * Event handling * OOP |
| public partial class StartScreen : Form  {  public StartScreen()  {  InitializeComponent();  StartPosition = FormStartPosition.CenterScreen; // makes it centrual in the beginning  }  private void ExitButton\_Click(object sender, EventArgs e)  {  this.Close(); // Closes form  Application.Exit(); // Terminates the execution of the program/shuts down the program  }  private void PlayButton\_Click(object sender, EventArgs e)  {    MainSimulationForm simulationScreen = new MainSimulationForm(); // Creates new instance of the Tutorial Screen form  simulationScreen.Show();// Shows the startScreen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  simulationScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  simulationScreen.Location = this.Location; // Remembers it  }  private void TutorialButton\_Click(object sender, EventArgs e)  {    TutorialScreen tutorialScreen = new TutorialScreen(); // Mreates new instance of the Tutorial Screen form  tutorialScreen.Show(); // Shows the startScreen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  tutorialScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  tutorialScreen.Location = this.Location; // Remmebers it    }  private void StartScreen\_Load(object sender, EventArgs e)  {  StartScreen startscrene = new StartScreen(); // Sets up a Start Screen  startscrene.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  }  } |

### Tutorial Screen

|  |
| --- |
| **Requirement: 2.h and 2.i** |
| Purpose: To provide a suitable Tutorial Screen for the user, to give explanation of what program does |
| Class: Tutorial Screen |
| Techniques used:   * Event handling * OOP |
| public partial class TutorialScreen : Form  {  public TutorialScreen()  {  InitializeComponent();  }  private void TutorialScreen\_Load(object sender, EventArgs e)  {  }  private void TutorialExitButton\_Click(object sender, EventArgs e)  {    StartScreen startScreen = new StartScreen(); // Creates new instance of the Tutorial Screen form  startScreen.Show(); // Shows the startScreen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  startScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  startScreen.Location = this.Location; // Remmebers it  }  } |

### Simulation Screen

|  |
| --- |
| **Requirement: 3.d** |
| Purpose: Setting up variables and constants |
| Class: MainSimulationForm |
| Techniques used:   * Using Random Class * Using constants |
| List<Boid> BoidList = new List<Boid>(); // Creates a list of Boids objects called BoidList  Random random = new Random();  List<Obstacle> ObstacleList = new List<Obstacle>(); // Creates a list of objects called ObstacleList  bool Wrapping = false; // Setting up variables  bool Tail = false;  bool ShowRadius = false;    public const int BOID\_RADIUS = 10;  const int BOID\_BORDER\_RADIUS = 20;  const int MINIMUM\_BOIDS\_AMOUNT = 20;  const int MAXIMUM\_BOIDS\_AMOUNT = 200;  const int BOIDS\_COUNT = 100;    const int PREDATORS\_RADIUS = 20;  const int MINIMUM\_PREDATORS\_AMOUNT = 0;  const int MAXIMUM\_PREDATORS\_AMOUNT = 5;  const int PREDATORS\_COUNT = 5;  const int COHESION\_VALUE = 1;  const int COHESION\_MIMNIMUM = 1;  const int COHESION\_MAXIMUM = 40;  const int ALIGNMENT\_VALUE = 1;  const int ALIGNMENT\_MIMNIMUM = 1;  const int ALIGNMENT\_MAXIMUM = 40;  const int SEPARATION\_VALUE = 1;  const int SEPARATION\_MIMNIMUM = 1;  const int SEPARATION\_MAXIMUM = 40;  const int VISION\_VALUE = 100;  const int VISION\_MIMNIMUM = 1;  const int VISION\_MAXIMUM = 200;  const int OBSTACLE\_SIZE\_VALUE = 50;  const int OBSTACLE\_SIZE\_MINIMUM = 10;  const int OBSTACLE\_SIZE\_MAXIMUM = 500;  public MainSimulationForm()  {  InitializeComponent();  NumberOfBoidstrackbar.Value = BOIDS\_COUNT; // Making trackbar value equal to the variable  NumberOfBoidstrackbar.Minimum = MINIMUM\_BOIDS\_AMOUNT; // Making trackbar minimum value equal to the variable  NumberOfBoidstrackbar.Maximum = MAXIMUM\_BOIDS\_AMOUNT; // Making trackbar maximum value equal to the variable  NumberOfBoidlabel.Text = BOIDS\_COUNT.ToString(); // Making labels text equal to the variable  NumberOfPredatorstrackbar.Value = PREDATORS\_COUNT; // Making trackbar value equal to the variable  NumberOfPredatorstrackbar.Minimum = MINIMUM\_PREDATORS\_AMOUNT; // Making trackbar minimum value equal to the variable  NumberOfPredatorstrackbar.Maximum = MAXIMUM\_PREDATORS\_AMOUNT; // Making trackbar maximum value equal to the variable  NumberOfPredatorslabel.Text = PREDATORS\_COUNT.ToString(); // Making labels text equal to the variable  CohesionStrengthtrackBar.Value = COHESION\_VALUE; // Making trackbar value equal to the variable  CohesionStrengthtrackBar.Minimum = COHESION\_MIMNIMUM; // Making trackbar minimum value equal to the variable  CohesionStrengthtrackBar.Maximum = COHESION\_MAXIMUM; // Making trackbar maximum value equal to the variable  CohesionStrengthnumberlabel.Text = CohesionStrengthtrackBar.Value.ToString(); // Making labels text equal to the variable  AlignmentStrengthtrackBar.Value = ALIGNMENT\_VALUE; // Making trackbar value equal to the variable  AlignmentStrengthtrackBar.Minimum = ALIGNMENT\_MIMNIMUM; // Making trackbar minimum value equal to the variable  AlignmentStrengthtrackBar.Maximum = ALIGNMENT\_MAXIMUM; // Making trackbar maximum value equal to the variable  AlignmentStrengthNumberlabel.Text = AlignmentStrengthtrackBar.Value.ToString(); // Making labels text equal to the variable  SeparationStrengthtrackBar.Value = SEPARATION\_VALUE; // Making trackbar value equal to the variable  SeparationStrengthtrackBar.Minimum = SEPARATION\_MIMNIMUM; // Making trackbar minimum value equal to the variable  SeparationStrengthtrackBar.Maximum = SEPARATION\_MAXIMUM; // Making trackbar maximum value equal to the variable  SeparationStrengthNumberlabel.Text = SeparationStrengthtrackBar.Value.ToString(); // Making labels text equal to the variable  VisiontrackBar.Value = VISION\_VALUE; // Making trackbar value equal to the variable  VisiontrackBar.Minimum = VISION\_MIMNIMUM; // Making trackbar minimum value equal to the variable  VisiontrackBar.Maximum = VISION\_MAXIMUM; // Making trackbar maximum value equal to the variable  VisionNumberlabel.Text = VisiontrackBar.Value.ToString(); // Making labels text equal to the variable  ObstacleSizetrackBar.Value = OBSTACLE\_SIZE\_VALUE; // Making trackbar value equal to the variable  ObstacleSizetrackBar.Minimum = OBSTACLE\_SIZE\_MINIMUM; // Making trackbar minimum value equal to the variable  ObstacleSizetrackBar.Maximum = OBSTACLE\_SIZE\_MAXIMUM; // Making trackbar maximum value equal to the variable  ObstacleSizelabel.Text = ObstacleSizetrackBar.Value.ToString(); // Making labels text equal to the variable  } |

|  |
| --- |
| **Requirement: 3.e.vii** |
| Purpose: To draw obstacles and Boids and update their movements by implementing flocking behaviour |
| Class: MainSimulationForm |
| Techniques used:   * OOP * Iterations * Graphical Rendering |
| private void SimulationBoxPaint(object sender, PaintEventArgs e) // Box painting  {  foreach (Boid Eachboid in BoidList) // Walks trough each element of the Boid list  {  Eachboid.DrawBoid(e.Graphics, Tail, VisiontrackBar.Value, ShowRadius); // Draws boid and predator  }  foreach (Obstacle obstacle in ObstacleList) // Walks through each element of the Obstacle list  {  obstacle.Draw(e.Graphics); // Draws Obstacle  }  }  private void SimulationTimer\_Tick(object sender, EventArgs e)  {  foreach (Boid EachBoid in BoidList) // Goes through Boid list  {  EachBoid.MoveBoid(BoidList, CohesionStrengthtrackBar.Value,AlignmentStrengthtrackBar.Value,  SeparationStrengthtrackBar.Value, VisiontrackBar.Value, ObstacleList); // MoveBoid method applied here  }    foreach (Boid EachBoid in BoidList) // Goes through Boid list  {  EachBoid.MoveLimit(SimulationScreenPicBox.Width, SimulationScreenPicBox.Height, Wrapping, ObstacleList); // MoveLimit method appleid here  }  SimulationScreenPicBox.Invalidate(); // Redraws when possible  } |

|  |
| --- |
| **Requirement: 3.e.vi** |
| Purpose: To load Boids and Predators without intersecting, also the ability to start and stop the simulation |
| Class: MainSimulationForm |
| Techniques used:   * OOP * User Interface Interaction * Maths and Physics Logic Implementation |
| private void MainSimulationForm\_Load(object sender, EventArgs e)  {  int Index = 0; // Sets up a variable that helps us to keep a number of objects in the BoidList  int PredatorAmount = NumberOfPredatorstrackbar.Value; // Sets up a variable that will euqal to the trackbar value  int BoidsAmount = NumberOfBoidstrackbar.Value; // Sets up a variable that will euqal to the trackbar value  while (Index < PredatorAmount) // Creates predators  {  AddPredators(Index);  Index++;  }    while (true) // Iterating until boids = boidsamount  {  bool IsColliding = false;  Boid NewBoid = newBoid(Index);  Index++;  if (BoidList.Count == 0)  {  BoidList.Add(NewBoid); // Adds first boid  if (BoidList.Count >= BoidsAmount) { break; } // breaks through while loop if BoidList = BoidsAmount  continue;  }  for (int i = 0; i < BoidList.Count; i++) // loops through until not collided, only works here  {    if (BoidList.Count < 2)  {  double XDifference = BoidList[0].XCentrePosition - NewBoid.XCentrePosition; // Finds the X coordinate difference  double YDifference = BoidList[0].YCentrePosition - NewBoid.YCentrePosition; // Finds the Y coordinate difference  double BoidsDistance = Math.Sqrt(Math.Pow(XDifference, 2) + Math.Pow(YDifference, 2)); // Finds the distance between to objects  if (BOID\_BORDER\_RADIUS < BoidsDistance) // Checks for intersection  {  BoidList.Add(NewBoid); // If not intersecting, add Boid  }  // if (BoidList.Count >= BoidsAmount) { break; }  }    else if (!IsColliding)  {  for (int j = 0; j < i; j++) // Second for loop, so that eeach object is compared with the rest of the list  {  double XDifference = BoidList[j].XCentrePosition - NewBoid.XCentrePosition; // Finds the X coordinate difference  double YDifference = BoidList[j].YCentrePosition - NewBoid.YCentrePosition; // Finds the Y coordinate difference  double BoidsDistance = Math.Sqrt(Math.Pow(XDifference, 2) + Math.Pow(YDifference, 2)); // Finds the distance between to objects  if (BOID\_BORDER\_RADIUS > BoidsDistance && !IsColliding) // If they don't intersect  {  IsColliding = true;  break; // Exit the second (j) loop  }  if (BoidList.Count >= BoidsAmount) { break; } // If the number of objects in the list equal or more the BoidsAmount, break out of the (j) for loop  }  }  }  if (!IsColliding) { BoidList.Add(NewBoid); }  if (BoidList.Count >= BoidsAmount) { break; } // Breaks through the while loop if the number of Boids corresponds to BoidList  }  }  private void StartStopSimulationButton\_Click(object sender, EventArgs e) // Start and stop button  {  if (SimulationTimer.Enabled == true) // Is timer is on  {  SimulationTimer.Enabled = false;  StartStopSimulationButton.Text = "Start"; // Changes it's text to "Start"  }  else // If timer is off  {  SimulationTimer.Enabled = true;  StartStopSimulationButton.Text = "Stop"; // Changes it's text to "Stop"  }  } |
| Completeness:  Produces circles without intersection. |

# Testing

# Evaluation

In general, the project was a success I m opinion as it met almost all the requirements and objectives that were set before writing the actual code. Analysing and Designing the problem before solving me helped me a lot with understanding how to structure my code and made it much easier for me as I had a plan of what to do with clear vison and diagrams. The hardest part of the project is getting used to graphics as it was my first experience with programming a 2-dimensional simulation with graphical simulation that looks like a game with movement. It also took some time for me to understand the mathematics behind the rules for Flocking Algorithm and how to use them to showcase a realistic animal behaviour. In addition, in took some time to understand and to make a convenient user interface with adding all the parameters (settings) so that user can adjust each of the Flocking Rules strength. However, excluding those difficulties, the production overall went smoothly. Furthermore, I would like to assess how successfully each of the requirements was met.

I have asked my friend who was a user of this system to give his honest feedback after using simulation for a while and after reminding him what the requirements were. The feedback is stored in this video.

Text in *italics* – represents independent feedback analysis, as well as how the project could be further developed or improved.

Test in normal – represents user honest opinion.

**Requirements 1.a - 1.e** *requirements were met successfully. Main Screen was loaded in the centre of the screen and it is opened when the program starts running. All 3 buttons work and direct you to different pages.*

**Requirements 2.a - 2.l** *were met successfully, user really liked that I have included A Tutorial Screen with explanation of how system works and their rules.* So new users that are not familiar with the Boids Simulation could spend some time understanding it. If they don’t understand what is a “Alignment Rule”, “Cohesion Rule” and “Separation Rule”, after reading description they will understand what it means to be changing those parameters in the Simulation Screen. However it was packed with information, which makes it a bit overwhelming. Adding short and interactive description is advised next time. *Possibly adding a feature on the Simulation Screen Setting Section that whenever you point the cursor of your mouse on the parameter that is going to be changed for some period of time and explanation of this parameter could be helpful as the new users won’t be able to remember everything from the beginning and travelling between screens is time consuming.*

**Requirement 3.a - 3.c** *were met successfully, Simulation Screen was opened in the same location as the Main Screen was closed, there were no overlapping of 2 forms and the Simulation Screen was split into 2 sections, the Settings Section and the Section where the simulation is going to run. In the future, to make simulation more practical I would add the ability to show/close the Settings Section so that the user can maximise the benefits of using whole window so that there would be more space for boids to move and helps to focus on the simulation itself rather than settings.*

**Requirement 3.d.i – 3.d.xii** were met successfully, he implemented quite a few things that I have suggested like changing the number of boids, changing the Flocking Rule strength. You can change the number of Predators and all the rules : Cohesion, Alignment, Separation, Vision and Obstacle size that were described in the tutorial. Obstacles could be added if wanted, also what I wanted. By adding obstacles, it created mountain like terrain, which was making simulation more natural. Restart button was quite useful. *All the buttons and the parameters were working, and the text of the buttons was changing. The way Boids and Predators avoided obstacles was too unnatural. If I will be doing same project again, I will be using different maths equations for it.*

**Requirement 3.e.i - 3.e.xiv** were met successfully, simulation was realistic and having an ability to see the radius of vision or tail is quite fascinating. However, there are some stuff on what simulation is missing out. The use of colour could have been improved for example background, to represent some sort of environment – jungle, mountains, ocean etc. In addition, have an ability to modify the shape of the obstacles to improve robustness and create more complex environments. Adding safe areas on the screen where Boids are safe from Predators. But these are all suggestions for improvements. *Program was fine until you started playing with parameters. Program just stopped being a simulation and you couldn’t call it being realistic. Cohesion was the biggest problem. Cohesion strength was too great for boids to escape and in the end, Boids and Predators were flocking together. Also, separation rule seemed unrealistic in real life simulation as there aren’t rapid movements away like that in real nature. Using different coefficient or algorithms could be more realistic. Possibly it would have been better if I created one base class and after made Boid and Predator inherit from it. It would make my system much neater and there would not be any problems with Predators being in a group together with Boids when Cohesion Strength was increased. As Cohesion rule and Predator rule work in a similar way. It reaches a point where Cohesion Strength is much greater than the force of the Boids running away so in the end, they all flock together: Boids and Predators. Also in my program, both Boids and Predators can see in 360 degrees which is also not realistic. Next time I should probably use sector of around 220 degrees so that Predators could come from behind and attack them by killing them perhaps, not just going after them but hunting – and if Predator and Boid collide, Boid could just disappear which would illustrate its death.*

**Requirement 4.a – 4.d** *were met successfully. Flocking algorithm was applied to each Boid and the Predator was after them which created a realistic behaviour across the simulation.*

A lot of time complexity was spent on method DrawBoid, which made program run slower when the number of boids increased. Nex time I would be using a different graphics rendering application like OpenGl instead of a PictureBox for drawing Boids and Predators, it would be better to use OpenGl instead which would reduce Space and Time complexity greatly which will make my program able to survive big quantities for Boids and Predators.

In the end the possibilities of improvement are immeasurable, and I am grateful for taking this project as it has improved my programming skills greatly. It helped me to be more aware of what’s happening where and If I had to do this project all over again. I would use everything I learned from it and try to make it even better.

# Appendix

## Obstacle Class

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Net;  using System.Text;  using System.Threading.Tasks;  namespace BOIDS\_SIMULATION\_NEA\_2024  {  public class Obstacle  {  private PointF location;  private int size;  public PointF Location { get { return location; } set { location = value; CalculateSize(); } } // Setting up location  public int Size { get { return size; } set { size = value; CalculateSize(); } } // Setting up size  public Point[] points = new Point[6];  public Obstacle(Point location, int size) // Constructor  {  Location = location;  Size = size;  CalculateSize();  }  public void Draw(Graphics graphics)  {  SolidBrush solidbrush = new SolidBrush(Color.DarkGray); // Sets up brush with the BoidsColour  graphics.FillPolygon(solidbrush, points); // Fill the area inside the hexagon  }  private void CalculateSize()  {  double Angle = Math.PI / 3.0;  for (int i = 0; i < 6; i++)  {  points[i] = new Point  (  (int)(Location.X + Size \* Math.Cos(i \* Angle)), // Generates edges of the hexagon  (int)(Location.Y + Size \* Math.Sin(i \* Angle)) // Generates edges  );  }  }  }  } |

## Predator Class

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace BOIDS\_SIMULATION\_NEA\_2024  {  public class Predator : Boid  {  public Predator(double xcentreposition, double ycentreposition, double xvelocity, double yvelocity, double radiusofboid, int id)  : base(xcentreposition, ycentreposition, xvelocity, yvelocity, radiusofboid, id) // Predator inheriting from Boids  {  BoidsColour = Color.Red; // Seeting up colour for predators  }  public override IEnumerable<Boid> FoundBoid(List<Boid> BoidList, double Vision)  {  return BoidList.Where(other => (this.GetDistance(other.GetLocation()) < Vision \* 2)); // Predator Vision is 2x better  }  public override void ColourUpdate(bool Stage) // must stay clear as it stops Predator in becoming yellow  {  }  }  } |

## Boid Class

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Runtime.InteropServices;  using System.Security.Cryptography;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using static System.Windows.Forms.VisualStyles.VisualStyleElement.Tab;  namespace BOIDS\_SIMULATION\_NEA\_2024  {  public class Boid  {  public double XCentrePosition;  public double YCentrePosition;  public double XVelocity;  public double YVelocity;  protected double RadiusOfBoid = 10;  protected Color BoidsColour;  public int ID;  private readonly int NumberOfHistoryPositions = 20; // Creates a memory through constant  Queue<Point> HistoryOfPositions = new Queue<Point>(); // Creats Queue  public Boid(double xcentreposition, double ycentreposition, double xvelocity, double yvelocity, double radiusofboid, int id) // Sets up constructor  {  XCentrePosition = xcentreposition;  YCentrePosition = ycentreposition;  XVelocity = xvelocity;  YVelocity = yvelocity;  RadiusOfBoid = radiusofboid;  BoidsColour = Color.Blue;  ID = id;  }  public virtual IEnumerable<Boid> FoundBoid(List<Boid> BoidList, double Vision)  {  return BoidList.Where(other => (this.GetDistance(other.GetLocation()) < Vision)); // Checks for objects  }  public void MoveBoid(List<Boid> BoidList,  double CohesionStrength = 1,// Sets up the trackbar values to 1  double AlignmentStrength = 1,  double SeparationStrength = 1,  double Vision = 100,  List<Obstacle> ObstacleList = null)  {  double VisionRepel = 20; // The distance between each boid when they flock  double VisionObstacleMod = 10; // The distance between the (Boids and Predators) and Objects, how closely they come before noticing it    // Cohesion  var Cohesion = FoundBoid(BoidList, Vision);  double MeanX = Cohesion.Sum(other => other.XCentrePosition) / Cohesion.Count(); // Finds the X centre of the group  double MeanY = Cohesion.Sum(other => other.YCentrePosition) / Cohesion.Count(); // Finds the Y centre of the group  this.XVelocity += (MeanX - this.XCentrePosition) \* 0.0005 \* CohesionStrength; // Applying the cohesion rule, updates X Velocity  this.YVelocity += (MeanY - this.YCentrePosition) \* 0.0005 \* CohesionStrength; // Applying the cohesion rule, updates X Velocity  // Separation  var FoundSeparations = BoidList.Where(other => (this.GetDistance(other.GetLocation()) < VisionRepel)); // if anyone comes closer that 20 = change your movement, you are too close  double SumSeparationX = FoundSeparations.Sum(predator => (this.XCentrePosition - predator.XCentrePosition) \* (VisionRepel - this.GetDistance(predator.GetLocation()))); // Calculate the sum of separation force that needs to be applied to the current boid to move it away from its neighbors found in FoundSeparations for X coordinates.  double SumSeparationY = FoundSeparations.Sum(predator => (this.YCentrePosition - predator.YCentrePosition) \* (VisionRepel - this.GetDistance(predator.GetLocation()))); // Calculate the sum of separation force that needs to be applied to the current boid to move it away from its neighbors found in FoundSeparations for Y coordinates.  this.XVelocity += SumSeparationX \* 0.005 \* SeparationStrength; // Applying the separation rule, updates X Velocity  this.YVelocity += SumSeparationY \* 0.005 \* SeparationStrength; // Applying the separation rule, updates Y Velocity  // Alignment  var FoundBoids = BoidList.Where(other => (this.GetDistance(other.GetLocation()) < Vision)); // Checks for objects near Vision radius  double MeanXVelocity = FoundBoids.Sum(other => other.XVelocity) / FoundBoids.Count(); // Finds average X Velocity  double MeanYVelocity = FoundBoids.Sum(other => other.YVelocity) / FoundBoids.Count(); // Finds average Y Velocity  this.XVelocity += (MeanXVelocity - this.XVelocity) \* 0.03 \* AlignmentStrength; // Applying the alignment rule, updates X Velocity  this.YVelocity += (MeanYVelocity - this.YVelocity) \* 0.03 \* AlignmentStrength; // Applying the alignment rule, updates Y Velocity  // Predator  var FoundPredators = BoidList.Where(other => (other is Predator && this.GetDistance(other.GetLocation()) < Vision)); // Find the predators in the radius: Vision  double SumPredatorX = FoundPredators.Sum(predator => (this.XCentrePosition - predator.XCentrePosition)); // Determines the general X direction in which the boid should move to escape the predators  double SumPredatorY = FoundPredators.Sum(predator => (this.YCentrePosition - predator.YCentrePosition)); // Determines the general Y direction in which the boid should move to escape the predators.  this.XVelocity += SumPredatorX \* 0.002; // Adjust the boid's X Velocity in response to the presence of predators  this.YVelocity += SumPredatorY \* 0.002; // Adjust the boid's Y Velocity in response to the presence of predators  // Obstacle  Point positionThis = new Point((int)this.XCentrePosition, (int)this.YCentrePosition); // we look at the hexagon as a circle as it is very close  var FoundObstacle = ObstacleList.Where(obstacle => (this.GetDistance(obstacle.Location) < (obstacle.Size + VisionObstacleMod))); // Checks for all obstacles that is close for specified Boid  double SumObstacleX = FoundObstacle.Sum(obstacle => (this.XCentrePosition - obstacle.Location.X) \* ((obstacle.Size + VisionObstacleMod) - this.GetDistance(obstacle.Location))); // X coordinate that change angle to not interfere the obstacle  double SumObstacleY = FoundObstacle.Sum(obstacle => (this.YCentrePosition - obstacle.Location.Y) \* ((obstacle.Size + VisionObstacleMod) - this.GetDistance(obstacle.Location))); // Y coordinate that change angle to not interfere the obstacle  this.XVelocity += SumObstacleX \* 0.001; // Applys the rule to changes Boids/Predatos X Velocity  this.YVelocity += SumObstacleY \* 0.001; // Applys the rule to changes Boids/Predatos Y Velocity    ColourUpdate(FoundPredators.Count() > 0); // If there are predators in the radius : Vision change normal boids colour to Yellow.  }  public virtual void ColourUpdate(bool Stage)  {  BoidsColour = Color.Blue; // Sets Boids colour to blue  if (Stage) { BoidsColour = Color.Yellow; } // Chanegs Boids colour to yellowe if they spot predator  }  public void MoveLimit(int PicBoxWidth, int PicBoxHeight, bool Wrapping = false, List<Obstacle> ObstacleList = null)  {  HistoryOfPositions.Enqueue(new Point((int)XCentrePosition, (int)YCentrePosition)); // Adds X and Y points to the queue  while (HistoryOfPositions.Count > NumberOfHistoryPositions) { HistoryOfPositions.Dequeue(); } // Removes the first points of the queue, so that the tail updates  XCentrePosition += XVelocity; // Adds X Velocity  YCentrePosition += YVelocity; // Adds Y Velocity  var Speed = GetSpeed();  if (Speed > 6) // Slows down  {  XVelocity = (XVelocity / Speed) \* 2;  YVelocity = (YVelocity / Speed) \* 2;  }  else if (Speed < 0.8) // Accelerates  {  XVelocity = (XVelocity / Speed) \* 0.8;  YVelocity = (YVelocity / Speed) \* 0.8;  }  if (double.IsNaN(XVelocity)) // Exception handling, if X Velocity gets too small, make it 0.  XVelocity = 0;    if (double.IsNaN(YVelocity)) // Exception handling, if Y Velocity gets too small, make it 0.  YVelocity = 0;  /// Code that makes circle bounce of the edges of the screen by changing its velocity  if (Wrapping)  {  if (XCentrePosition < 20) // If X position of the object is below 20 pixels  {  XVelocity += 0.5; // Change its X Velocity  }  if (XCentrePosition > PicBoxWidth - 20) // If X position of the object is above (Picture box width - 20 pixels)  {  XVelocity -= 0.5; // Change its X Velocity  }  if (YCentrePosition < 20) // If Y position of the object is below 20 pixels  {  YVelocity += 0.5; // Change its Y Velocity  }  if (YCentrePosition > PicBoxHeight - 20) // If Y position of the object is above (Picture box height - 20 pixels)  {  YVelocity -= 0.5; // Change its Y Velocity  }  }  else  {  /// Code that makes circles/boids Wrap Around  if (XCentrePosition < 0) // If X position of the object is below 0 pixels  {  XCentrePosition = PicBoxWidth; // Changes it's X Position to Picture box width  }  if (XCentrePosition > PicBoxWidth) // If X position of the object is above Picture box width  {  XCentrePosition = 0; // Changes it's X Position to 0 pixels  }  if (YCentrePosition < 0) // If Y position of the object is below 0 pixels  {  YCentrePosition = PicBoxHeight; // Changes it's Y Position to Picture box height  }  if (YCentrePosition > PicBoxHeight) // If Y position of the object is above Picture box height  {  YCentrePosition = 0; // Changes it's Y Position to 0 pixels  }  }  }  public void DrawBoid(Graphics g, bool tail, int Vision, bool ShowRadius = true)  {  Pen pen = new Pen(BoidsColour); // Sets up pen with the BoidsColour  SolidBrush solidbrush = new SolidBrush(BoidsColour); // Sets up brush with the BoidsColour  g.DrawEllipse(pen, (int)XCentrePosition - (int)RadiusOfBoid / 2, (int)YCentrePosition - (int)RadiusOfBoid / 2, (int)RadiusOfBoid, (int)RadiusOfBoid);// Draws Elipse with these variables  g.FillEllipse(solidbrush, (int)XCentrePosition - (int)RadiusOfBoid / 2, (int)YCentrePosition - (int)RadiusOfBoid / 2, (int)RadiusOfBoid, (int)RadiusOfBoid); // Fills the Elipse with these variables  if (ShowRadius) // If true  {  if (this is Predator) { Vision \*= 2; } // If predator, it's vision is twice bigger    g.DrawEllipse(pen, (int)(XCentrePosition - Vision), (int)(YCentrePosition - Vision), Vision \* 2, Vision \* 2); // Shows field of vision fore predator and boid  g.DrawEllipse(pen, (int)XCentrePosition - 20 / 2, (int)YCentrePosition - 20 / 2, 20, 20); // Shows boundaries of each boid  }  if (tail) // If true  {  Point point = new Point((int)XCentrePosition, (int)YCentrePosition); // Creates new points  int i = 0;  foreach (var item in HistoryOfPositions) // Foreach loop to go through each point in the Queue History Of Positions  {  if (GetDistance(item) < 100) // Compares distances  {  pen.Width = (int)((i \* RadiusOfBoid) / NumberOfHistoryPositions); // Draws a triangular shape trail  g.DrawLine(pen, point, item); // Draws line between "point" and "item"  point = item;  }  i++;  }  }  }  public double GetDistance(PointF Other) // Checking distance  {  double dx = Other.X - XCentrePosition; // Pythagoras, same thing  double dy = Other.Y - YCentrePosition;  return Math.Sqrt(dx \* dx + dy \* dy); // Returns distance  }  public double GetSpeed() // speed  {  return Math.Sqrt(XVelocity \* XVelocity + YVelocity \* YVelocity); // Pythagoras to find speed  }  public PointF GetLocation()  {  return new PointF((float)this.XCentrePosition, (float)this.YCentrePosition);// Returns X and Y coordinates with float conversion  }  }  } |

## MainSimulation Form (Simulation Screen)

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Configuration;  using System.Data;  using System.Drawing;  using System.Drawing.Drawing2D;  using System.Linq;  using System.Reflection;  using System.Reflection.Emit;  using System.Runtime.InteropServices;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  using System.Xml.Linq;  using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;  namespace BOIDS\_SIMULATION\_NEA\_2024  {  public partial class MainSimulationForm : Form //parameter ID - unique name of the boio, in order to find it later  {    List<Boid> BoidList = new List<Boid>(); // Creates a list of Boids objects called ListOfBoids  Random random = new Random();  List<Obstacle> ObstacleList = new List<Obstacle>(); // Creates a list of objects called ObjectList  bool Wrapping = false; // Setting up variables  bool Tail = false;  bool ShowRadius = false;    public const int BOID\_RADIUS = 10;  const int BOID\_BORDER\_RADIUS = 20;  const int MINIMUM\_BOIDS\_AMOUNT = 20;  const int MAXIMUM\_BOIDS\_AMOUNT = 200;  const int BOIDS\_COUNT = 100;    const int PREDATORS\_RADIUS = 20;  const int MINIMUM\_PREDATORS\_AMOUNT = 0;  const int MAXIMUM\_PREDATORS\_AMOUNT = 5;  const int PREDATORS\_COUNT = 5;  const int COHESION\_VALUE = 1;  const int COHESION\_MIMNIMUM = 1;  const int COHESION\_MAXIMUM = 40;  const int ALIGNMENT\_VALUE = 1;  const int ALIGNMENT\_MIMNIMUM = 1;  const int ALIGNMENT\_MAXIMUM = 40;  const int SEPARATION\_VALUE = 1;  const int SEPARATION\_MIMNIMUM = 1;  const int SEPARATION\_MAXIMUM = 40;  const int VISION\_VALUE = 100;  const int VISION\_MIMNIMUM = 1;  const int VISION\_MAXIMUM = 200;  const int OBSTACLE\_SIZE\_VALUE = 50;  const int OBSTACLE\_SIZE\_MINIMUM = 10;  const int OBSTACLE\_SIZE\_MAXIMUM = 500;  public MainSimulationForm()  {  InitializeComponent();  NumberOfBoidstrackbar.Value = BOIDS\_COUNT; // Making trackbar value equal to the variable  NumberOfBoidstrackbar.Minimum = MINIMUM\_BOIDS\_AMOUNT; // Making trackbar minimum value equal to the variable  NumberOfBoidstrackbar.Maximum = MAXIMUM\_BOIDS\_AMOUNT; // Making trackbar maximum value equal to the variable  NumberOfBoidlabel.Text = BOIDS\_COUNT.ToString(); // Making labels text equal to the variable  NumberOfPredatorstrackbar.Value = PREDATORS\_COUNT; // Making trackbar value equal to the variable  NumberOfPredatorstrackbar.Minimum = MINIMUM\_PREDATORS\_AMOUNT; // Making trackbar minimum value equal to the variable  NumberOfPredatorstrackbar.Maximum = MAXIMUM\_PREDATORS\_AMOUNT; // Making trackbar maximum value equal to the variable  NumberOfPredatorslabel.Text = PREDATORS\_COUNT.ToString(); // Making labels text equal to the variable  CohesionStrengthtrackBar.Value = COHESION\_VALUE; // Making trackbar value equal to the variable  CohesionStrengthtrackBar.Minimum = COHESION\_MIMNIMUM; // Making trackbar minimum value equal to the variable  CohesionStrengthtrackBar.Maximum = COHESION\_MAXIMUM; // Making trackbar maximum value equal to the variable  CohesionStrengthnumberlabel.Text = CohesionStrengthtrackBar.Value.ToString(); // Making labels text equal to the variable  AlignmentStrengthtrackBar.Value = ALIGNMENT\_VALUE; // Making trackbar value equal to the variable  AlignmentStrengthtrackBar.Minimum = ALIGNMENT\_MIMNIMUM; // Making trackbar minimum value equal to the variable  AlignmentStrengthtrackBar.Maximum = ALIGNMENT\_MAXIMUM; // Making trackbar maximum value equal to the variable  AlignmentStrengthNumberlabel.Text = AlignmentStrengthtrackBar.Value.ToString(); // Making labels text equal to the variable  SeparationStrengthtrackBar.Value = SEPARATION\_VALUE; // Making trackbar value equal to the variable  SeparationStrengthtrackBar.Minimum = SEPARATION\_MIMNIMUM; // Making trackbar minimum value equal to the variable  SeparationStrengthtrackBar.Maximum = SEPARATION\_MAXIMUM; // Making trackbar maximum value equal to the variable  SeparationStrengthNumberlabel.Text = SeparationStrengthtrackBar.Value.ToString(); // Making labels text equal to the variable  VisiontrackBar.Value = VISION\_VALUE; // Making trackbar value equal to the variable  VisiontrackBar.Minimum = VISION\_MIMNIMUM; // Making trackbar minimum value equal to the variable  VisiontrackBar.Maximum = VISION\_MAXIMUM; // Making trackbar maximum value equal to the variable  VisionNumberlabel.Text = VisiontrackBar.Value.ToString(); // Making labels text equal to the variable  ObstacleSizetrackBar.Value = OBSTACLE\_SIZE\_VALUE; // Making trackbar value equal to the variable  ObstacleSizetrackBar.Minimum = OBSTACLE\_SIZE\_MINIMUM; // Making trackbar minimum value equal to the variable  ObstacleSizetrackBar.Maximum = OBSTACLE\_SIZE\_MAXIMUM; // Making trackbar maximum value equal to the variable  ObstacleSizelabel.Text = ObstacleSizetrackBar.Value.ToString(); // Making labels text equal to the variable  }  private void SimulationBoxPaint(object sender, PaintEventArgs e) // Box painting  {  foreach (Boid Eachboid in BoidList) // Walks trough each element of the Boid list  {  Eachboid.DrawBoid(e.Graphics, Tail, VisiontrackBar.Value, ShowRadius); // Draws boid and predator  }  foreach (Obstacle obstacle in ObstacleList) // Walks through each element of the Obstacle list  {  obstacle.Draw(e.Graphics); // Draws Obstacle  }  }  private void MainSimulationExitButton\_Click(object sender, EventArgs e)  {  StartScreen startScreen = new StartScreen(); // Creates new instance of the Tutorial Screen form  startScreen.Show(); // Shows Start Screen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  startScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  startScreen.Location = this.Location; // Remmebers it  }  private void SimulationTimer\_Tick(object sender, EventArgs e)  {  foreach (Boid EachBoid in BoidList) // Goes through Boid list  {  EachBoid.MoveBoid(BoidList, CohesionStrengthtrackBar.Value,AlignmentStrengthtrackBar.Value,  SeparationStrengthtrackBar.Value, VisiontrackBar.Value, ObstacleList); // MoveBoid method applied here  }    foreach (Boid EachBoid in BoidList) // Goes through Boid list  {  EachBoid.MoveLimit(SimulationScreenPicBox.Width, SimulationScreenPicBox.Height, Wrapping, ObstacleList); // MoveLimit method appleid here  }  SimulationScreenPicBox.Invalidate(); // Redraws when possible  }  private void MainSimulationForm\_Load(object sender, EventArgs e)  {  int Index = 0; // Sets up a variable that helps us to keep a number of objects in the BoidList  int PredatorAmount = NumberOfPredatorstrackbar.Value; // Sets up a variable that will euqal to the trackbar value  int BoidsAmount = NumberOfBoidstrackbar.Value; // Sets up a variable that will euqal to the trackbar value  while (Index < PredatorAmount) // Creates predators  {  AddPredators(Index);  Index++;  }    while (true) // Iterating until boids = boidsamount  {  bool IsColliding = false;  Boid NewBoid = newBoid(Index);  Index++;  if (BoidList.Count == 0)  {  BoidList.Add(NewBoid); // Adds first boid  if (BoidList.Count >= BoidsAmount) { break; } // breaks through while loop if BoidList = BoidsAmount  continue;  }  for (int i = 0; i < BoidList.Count; i++) // loops through until not collided, only works here  {    if (BoidList.Count < 2)  {  double XDifference = BoidList[0].XCentrePosition - NewBoid.XCentrePosition; // Finds the X coordinate difference  double YDifference = BoidList[0].YCentrePosition - NewBoid.YCentrePosition; // Finds the Y coordinate difference  double BoidsDistance = Math.Sqrt(Math.Pow(XDifference, 2) + Math.Pow(YDifference, 2)); // Finds the distance between to objects  if (BOID\_BORDER\_RADIUS < BoidsDistance) // Checks for intersection  {  BoidList.Add(NewBoid); // If not intersecting, add Boid  }  // if (BoidList.Count >= BoidsAmount) { break; }  }    else if (!IsColliding)  {  for (int j = 0; j < i; j++) // Second for loop, so that eeach object is compared with the rest of the list  {  double XDifference = BoidList[j].XCentrePosition - NewBoid.XCentrePosition; // Finds the X coordinate difference  double YDifference = BoidList[j].YCentrePosition - NewBoid.YCentrePosition; // Finds the Y coordinate difference  double BoidsDistance = Math.Sqrt(Math.Pow(XDifference, 2) + Math.Pow(YDifference, 2)); // Finds the distance between to objects  if (BOID\_BORDER\_RADIUS > BoidsDistance && !IsColliding) // If they don't intersect  {  IsColliding = true;  break; // Exit the second (j) loop  }  if (BoidList.Count >= BoidsAmount) { break; } // If the number of objects in the list equal or more the BoidsAmount, break out of the (j) for loop  }  }  }  if (!IsColliding) { BoidList.Add(NewBoid); }  if (BoidList.Count >= BoidsAmount) { break; } // Breaks through the while loop if the number of Boids corresponds to BoidList  }  }  private void StartStopSimulationButton\_Click(object sender, EventArgs e) // Start and stop button  {  if (SimulationTimer.Enabled == true) // Is timer is on  {  SimulationTimer.Enabled = false;  StartStopSimulationButton.Text = "Start"; // Chaneges it's text to "Start"  }  else // If timer is off  {  SimulationTimer.Enabled = true;  StartStopSimulationButton.Text = "Stop"; // Changes it's text to "Stop"  }  }  private void NumberOfBoidstrackbar\_Scroll(object sender, EventArgs e)  {  NumberOfBoidlabel.Text = NumberOfBoidstrackbar.Value.ToString(); // Sets up the label's text to be equal to the trackbars value  int BoidsAmount = NumberOfBoidstrackbar.Value; // Sets up a variable that will be equal to the trackbar  int Index = 0;  for (int i = BoidList.Count() - 1; i >= 0; i--) // Starting from the top of list, beacusew we cant delete from begining as it decresses the count so in the end we miss some objects  {  if ((BoidList[i].GetType().Name == nameof(Boid))) // Search boid withoud predator  {  if (BoidsAmount > 0)  {  BoidsAmount--; // Decrease the number of the Boids  }  else  {  BoidList.RemoveAt(i); // Removes Boid from the Bodi List  }  continue;  }  if (BoidList[i].ID > Index) // Iterates through each element, if ID of the item is more than Index, we make it equal to it  {  Index = BoidList[i].ID; // Looking for maximum id of existing ones in this list  }  }  for (int i = 0; i < BoidsAmount; i++) // Loop to add Boids  {  Index++;  BoidList.Add(newBoid(Index)); // Adds Boid  }  }  private void WrapAroundbutton\_Click(object sender, EventArgs e) // Wrap Around button  {  if (Wrapping == true) // checks if its true  {  WrapAroundbutton.Text = "Wrapping"; // if yes then changes text in the the button  Wrapping = false;  }  else  {  WrapAroundbutton.Text = "Not Wrapping"; // if no changes text in the button  Wrapping = true;  }  }  private void NumberOfPredatorstrackbar\_Scroll(object sender, EventArgs e)  {  NumberOfPredatorslabel.Text = NumberOfPredatorstrackbar.Value.ToString(); // Sets up the label's text to be equal to the trackbars value  int PredatorAmount = NumberOfPredatorstrackbar.Value; // Sets up a variable that will be equal to the trackbar  int Index = 0;  for (int i = BoidList.Count()-1; i >= 0; i--) // Goes through a BoidList  {  if (BoidList[i].GetType().Name == nameof(Predator)) // If a member of the BoidList is from Predator class  {  if (PredatorAmount > 0)  {  PredatorAmount--; // Decrease the number of Predators  }  else  {  BoidList.RemoveAt(i); // Delets Predators from the BoidList  }  continue;  }  if (BoidList[i].ID > Index) // Makes sure we go through the list  {  Index = BoidList[i].ID; // Makes Index equal to the ID of the object from the list  }  }  for (int i = 0; i < PredatorAmount; i++)  {  Index++; // Increments variable  AddPredators(Index); // Adds Predator with an ID that is equal to index  }  }  private void AddPredators(int ID) // Creates new Predator Object  {  Predator NewPredator = new Predator(  random.Next(40, SimulationScreenPicBox.Width - 40),  random.Next(40, SimulationScreenPicBox.Height - 40),  random.Next(-30, 30) / 10,  random.Next(-30, 30) / 10, PREDATORS\_RADIUS,  ID);  BoidList.Add(NewPredator); // Adds this object to the BoidsList.  }  private Boid newBoid(int ID) // Creates new Boid Object  {  Boid NewBoid = new Boid(  random.Next(20, SimulationScreenPicBox.Width - 20),  random.Next(20, SimulationScreenPicBox.Height - 20),  random.Next(-30, 30) / 10,  random.Next(-30, 30) / 10, BOID\_RADIUS,  ID);  return NewBoid;  }  private void SimulationScreenPicBox\_Click(object sender, EventArgs e)  {  MouseEventArgs MouseLocation = e as MouseEventArgs; // Mouse click  for ( int i = ObstacleList.Count - 1; i >= 0; i--) // Walk through all the objects  {  double dx = MouseLocation.X - ObstacleList[i].Location.X; // Gets the X coordinates difference between the mouse click and object  double dy = MouseLocation.Y - ObstacleList[i].Location.Y; // Gets the Y coordinates difference between the mouse click and object  if ( Math.Sqrt(dx \* dx + dy \* dy) < ObstacleList[i].Size) // If radius of the object < radius , delets, size - my radius  {  ObstacleList.RemoveAt(i); // If inside the area, delets it  SimulationScreenPicBox.Invalidate(); // Redraws  return; // Leaves the method,  }  }    ObstacleList.Add(new Obstacle(MouseLocation.Location, ObstacleSizetrackBar.Value)); // Else adds an object  SimulationScreenPicBox.Invalidate(); // Redraws the picture box  }  private void CohesionStrengthtrackBar\_Scroll(object sender, EventArgs e)  {  CohesionStrengthnumberlabel.Text = CohesionStrengthtrackBar.Value.ToString(); // Sets up the label's text to be equal to the trackbars value  }  private void RestartButton\_Click(object sender, EventArgs e)  {  BoidList.Clear(); // Clears Boid list  ObstacleList.Clear(); // Clears Obstacle list  NumberOfBoidstrackbar.Value = BOIDS\_COUNT; // Sets all values below to it's standards  NumberOfBoidlabel.Text = BOIDS\_COUNT.ToString();  NumberOfPredatorstrackbar.Value = PREDATORS\_COUNT;  NumberOfPredatorslabel.Text = PREDATORS\_COUNT.ToString();    CohesionStrengthtrackBar.Value = COHESION\_VALUE;  CohesionStrengthnumberlabel.Text = CohesionStrengthtrackBar.Value.ToString();    AlignmentStrengthtrackBar.Value = ALIGNMENT\_VALUE;  AlignmentStrengthNumberlabel.Text = AlignmentStrengthtrackBar.Value.ToString();    SeparationStrengthtrackBar.Value = SEPARATION\_VALUE;  SeparationStrengthNumberlabel.Text = SeparationStrengthtrackBar.Value.ToString();    VisiontrackBar.Value = VISION\_VALUE;  VisionNumberlabel.Text = VisiontrackBar.Value.ToString();    ObstacleSizetrackBar.Value = OBSTACLE\_SIZE\_VALUE;  ObstacleSizelabel.Text = ObstacleSizetrackBar.Value.ToString();  MainSimulationForm\_Load(null, null); // Reloads the system  }  private void ObstacleSizetrackBar\_Scroll(object sender, EventArgs e)  {  ObstacleSizelabel.Text = ObstacleSizetrackBar.Value.ToString();// Sets up the label's text to be equal to the trackbars value  foreach (var item in ObstacleList)  {  item.Size = ObstacleSizetrackBar.Value; // Size of the object in the list is equal to the number in the trackbar  }  }  private void AlignmentStrengthtrackBar\_Scroll(object sender, EventArgs e)  {  AlignmentStrengthNumberlabel.Text = AlignmentStrengthtrackBar.Value.ToString();// Sets up the label's text to be equal to the trackbars value  }  private void SeparationStrengthtrackBar\_Scroll(object sender, EventArgs e)  {  SeparationStrengthNumberlabel.Text = SeparationStrengthtrackBar.Value.ToString();// Sets up the label's text to be equal to the trackbars value  }  private void PredatorsVisiontrackBar\_Scroll(object sender, EventArgs e)  {  VisionNumberlabel.Text = VisiontrackBar.Value.ToString(); // Sets up the label's text to be equal to the trackbars value  }  private void TailButton\_Click(object sender, EventArgs e)  {  if (Tail == true) // checks if its true  {  TailButton.Text = "No Tail"; // if yes then changes text in the button  Tail = false;  }  else  {  TailButton.Text = "Tail"; // if no changes text in the button  Tail = true;  }  }  private void ShowRadiusButton\_Click(object sender, EventArgs e)  {  if (ShowRadius == true) // checks if its true  {  ShowRadiusButton.Text = "Show Radius: no"; // if yes then changes text in the button  ShowRadius = false;  }  else  {  ShowRadiusButton.Text = "Show Radius: yes"; // if no changes text in the button  ShowRadius = true;  }  }  }  } |

## StartScreen Form

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace BOIDS\_SIMULATION\_NEA\_2024  {  public partial class StartScreen : Form  {  public StartScreen()  {  InitializeComponent();  StartPosition = FormStartPosition.CenterScreen; // makes it centrual in the beginning  }  private void ExitButton\_Click(object sender, EventArgs e)  {  this.Close(); // Closes form  Application.Exit(); // Terminates the execution of the program/shuts down the program  }  private void PlayButton\_Click(object sender, EventArgs e)  {    MainSimulationForm simulationScreen = new MainSimulationForm(); // Creates new instance of the Tutorial Screen form  simulationScreen.Show();// Shows the startScreen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  simulationScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  simulationScreen.Location = this.Location; // Remembers it  }  private void TutorialButton\_Click(object sender, EventArgs e)  {    TutorialScreen tutorialScreen = new TutorialScreen(); // Imitates new instance of the Tutorial Screen form  tutorialScreen.Show(); // Shows the startScreen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  tutorialScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  tutorialScreen.Location = this.Location; // Remmebers it    }  private void StartScreen\_Load(object sender, EventArgs e)  {  StartScreen startscrene = new StartScreen(); // Sets up a Start Screen  startscrene.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  }  }  } |

## TutorialScreen form

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace BOIDS\_SIMULATION\_NEA\_2024  {  public partial class TutorialScreen : Form  {  public TutorialScreen()  {  InitializeComponent();  }  private void TutorialScreen\_Load(object sender, EventArgs e)  {  }  private void TutorialExitButton\_Click(object sender, EventArgs e)  {    StartScreen startScreen = new StartScreen(); // Creates new instance of the Tutorial Screen form  startScreen.Show(); // Shows the startScreen  Visible = false; // Makes it invisible so that the form we open wont be on top the one we close  startScreen.StartPosition = FormStartPosition.Manual; // Sets up the location of the form we are currently in  startScreen.Location = this.Location; // Remmebers it  }  }  } |