University of Sheffield

Modelling the Interaction of a Murmuration of Starlings with a Predator

Thomas Boyd

Supervisor: Paul Watton

Module code: COM3610

This report is submitted in partial fulfilment of the requirement for the degree of Software Engineering by Thomas Luke Boyd

May 16, 2021

Declaration

Name: **Thomas Boyd**

Date: **May 16, 2021**

Abstract

Starling murmurations are a natural phenomenon which create interesting and complex shapes and patterns in the sky. When a murmuration interacts with a predator such as the peregrine falcon, it creates more complex shapes as the murmurations appears as if it's one organism escaping the predator. This project aims to capture these shapes and patterns in a visually pleasing 3D model with a murmuration and predator.

The work has achieved an accurate replication of the interactions between a murmuration and predator, along with many realistic features which bring the simulation to life such as a landscape and sky as well as accurate models and animations. To increase the size of the murmuration, performance of the simulation has been optimised with use of a uniform grid structure and multi-threading.

COVID-19 Impact Statement

The lockdown imposed because of COVID-19 caused additional challenges for the completion of this project. In the year of the project, the university used an online delivery of all teaching, and university buildings were closed. All project meetings were shifted to email correspondence and video meetings.

Acknowledgements

I would like to thank my supervisor, Dr Paul Watton for reaching out to me with this interesting and exciting project, helping me throughout the project with useful advice and helping me question all the aspects of designing and implementing this project. I would also like to thank the department of computer science who hosted a dissertation video competition where I came 2nd place which gave me confidence in this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The phenomenon of starling murmurations is an example of collective animal behaviour, like fish shoals and herds of migrating blue wildebeest (Connochaetes taurinus) [9]. More specifically, a flock of starling, also called a starling murmuration (called because of the sound produced by multiple wingbeats), it is a collection which can involve thousands of individual birds forming a coherent three-dimensional murmuration “cloud” within which the movement of each individual bird is highly cohesive and synchronised [3]. The synchronised movement of the starling murmuration means they can form different shapes including spheres, planes and waves whilst remaining static with respect to a focal point on the ground [3]. When the starling murmuration is introduced to a predator such as the falcon, due to the starling’s large structure, consequently the falcon will struggle to single out a starling, as a response the falcon will wait for an opening and then dive towards the murmuration in trying to capture a single starling or separate them to single one out for another dive. As a result of the predator’s attack, the murmuration will form patterns of evasive manoeuvres to avoid the predator, these patterns are an interesting natural phenomenon which this project aims to capture.

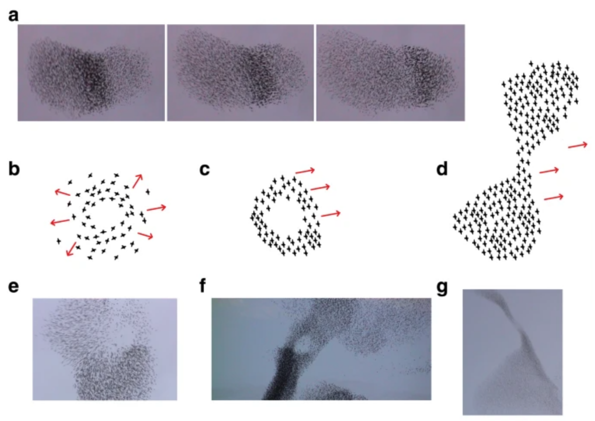


Figure 1.1: Images showing different starling murmuration patterns caused by a predator [27]

1.1 Aims and Objectives

This work aims to study the murmurations interaction with a predator and capture the patterns which arise from the interactions by developing a 3D simulation with accurate 3D representations of the starlings and falcon to improve visualisation on previous models.

To achieve this aim, the objectives which will need to complete are as follows:

- Create an accurate 3D model of a starling and peregrine falcon
- Design a set of rules of starling murmuration behaviour by critiquing previous models
- Implement the rules to a flock of starlings in Unity
- Study the peregrine falcon and its behaviour and interaction with murmurations to develop a set of rules
- Introduce the peregrine falcon to Unity and create small scenarios to test its rules and behaviour
- Improve the visualisation of the model e.g. sky, objects such as trees and animations
- Implement a way of evaluating the simulation with the use of metrics

The work aims to be completed by 12th May 2021, within this time there may arise some constraints which may affect the way the project is carried out. Some constraints which the work may encounter is time, the quality of the project can be improved if there were more time as more realistic features could be applied into the simulation such as calculating starlings banking angles and more advanced physics. Performance is also a constraint, as simulating a murmuration with its efficiency at $O(n^2)$ with 3D models will hinder the size of the murmuration as the processing power will be a factor.

1.2 Overview of the Report

The remaining chapters of the report cover a literature survey, requirements and analysis, a design section, implementation and testing, results, and the conclusion. The main part of this report is the literature survey where literature will be reviewed and analysed relating to the aims and objectives, in this it will reveal answers to questions, issues that that may be encountered and there will be critiquing previous models of starling murmurations with a predator. This is followed by the requirements and analysis; this will be going into further detail on the aims and objectives and establishing a method of evaluating the work allowing methods to consider the work successful. The design section puts together ideas for creating different aspects of the simulation, where these are then implemented and tested in the following section. The results section will discuss findings and evaluate the how successful the work, discuss how the project is moving forward in the future and finally a conclusion which summarises the work in this project.

Chapter 2

Literature Survey

In this literature survey, the latest literature and research will be reviewed on the behaviour of starling murmurations, a peregrine falcon, how the behaviour of a starling murmuration can change to form complex shapes within the proximity with a peregrine falcon as well as critique and insight into previous models.

2.1 Starlings

The literature survey will begin with an introduction to starlings. Starlings are medium-sized passerines and they are a highly social family of birds where they are generally found in flocks of birds varying in sizes throughout the year [37]. In North America in 1970, the number of starlings was considered to be 200 million, however this figure is declining by an average of 1% per year and the figure stands at approximately 140 million this year [14].

Figure 2.1: A common starlings [37]

2.1.1 Murmurations

A starling murmuration is a collection of starlings in a group forming a range of different shapes including spheres, planes and waves while remaining more-or-less static over a focal point on the ground such as a breeding ground [3]. The size of a starling murmuration can range from 1,000 to over 100,000 of starlings which averages 30,000 per flock and can last up to an hour. There are few explanations to the reason starlings form a murmuration, one explanation being the ”warming together” hypothesis; murmurations have been seen to occur before roosting and during the late Autumn and Winter months suggesting they create the murmuration to attract other starlings and as more starlings gather, the roost becomes warmer [18]. A more logical explanation is the murmuration is an anti-predator strategy, as the starlings are prey to the peregrine falcon the murmuration reduces the chance of the peregrine falcon capturing a starling in several ways, for example the probability of a starling being targeted by themselves compared with being in a murmuration deceases, as with the size of the flock becoming more massive, the opportunity for success for survival increases [15]. The constant movement and maneuvers of the murmuration can confuse the predator, rendering them unable to lock onto a single bird [25], this is known as target degeneracy, with the flock size increasing, it will significantly impact the success of attacks from the predator as there is unclear vision of tracking a target [4]. Additionally, the increase in the size of murmuration correlates with the detection of a predator, therefore earlier evasive maneuvers occur decreasing the success rate of the predator. A study on understanding why birds flock revealed that in social animals (starlings), affiliative contact is reinforced because it reduced a negative affective state caused by social exclusion or isolation [17], therefore when the starlings create a murmuration, there is positive reinforcement from this and when are they are outside of a murmuration there is a negative reinforcement effect. Murmurations create waves which are used to send information across such as to signal the location of food sources or a safe roosting site. More importantly, the information is used to defend attacks from a predator [16] which allows coordinated movement of the murmuration to respond effectively to a predator which creates patterns which will be explored later in this literature review.

2.1.2 Patterns

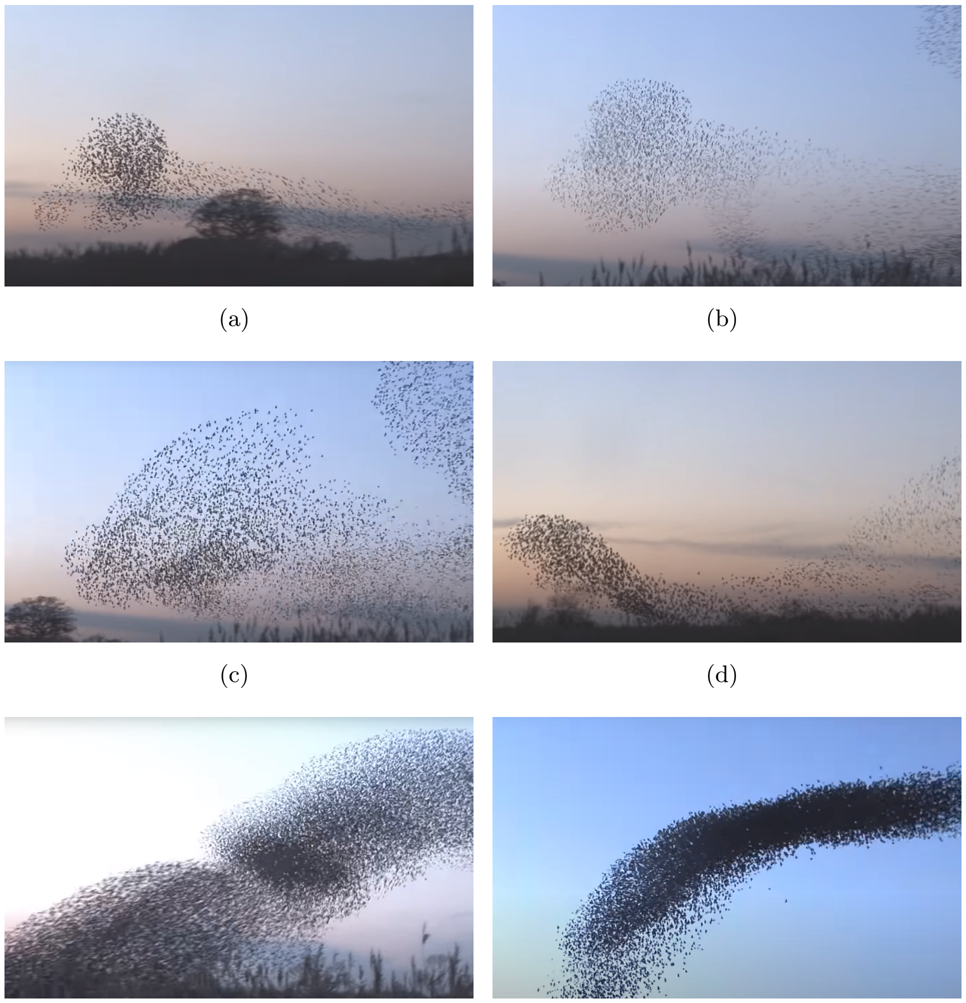


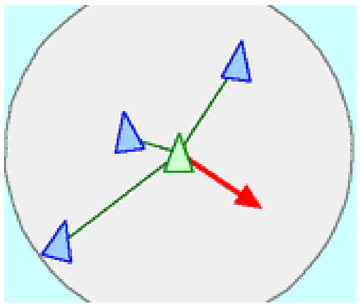
Figure 2.2: Images of a starling murmuration without a predator [38]

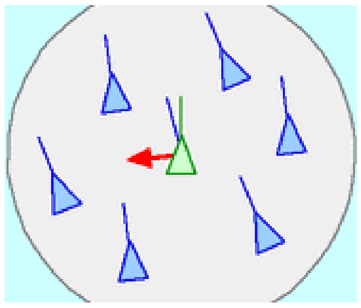
Starling murmurations create wonderful patterns as shown in figure 2.2, and this project aims to recreate the patterns shown in this murmuration before applying a predator. In creating an agent-based simulation of the murmuration, inspiration can be taken from models such as fish schools [13] which achieve a similar swarm like effect, in such a model, each individual is characterised by its position, scalar speed and orientation in space and its behaviour is based on the position and direction of its neighbours by use of cohesion, alignment and separation.

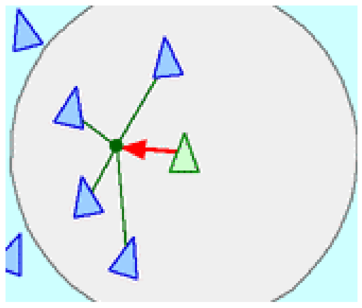
2.1.3 Starling Behaviour

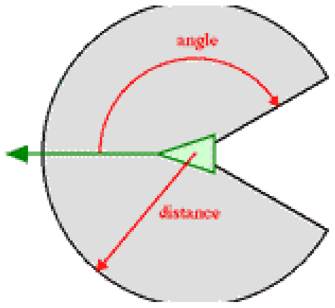
In 1986, a computer model [30] was created to model coordinated animal behaviour such as bird flocks and fish schools. The model is comprised of three steering behaviours, separation, alignment and cohesion to describe how an individual can manoeuvre based on the positions and velocities of its nearby flock mates. These steering behaviours are shown in figure 2.3. In the model, the nearby flock mates used to calculate each of the behaviours are found within a distance of the boid (generic simulated flocking creature) and an angle to avoid taking into consideration the other boids outside of its line of site. It was however found [20] [12] that instead of taking into account all the boids within a range of their position, that only the nearest 6-7 neighbours in a topological range should be considered.

Starlings have been shown to not always move in predictable patterns, instead their movement arises from the response to other birds and behave in response to attraction, alignment and avoidance [12], these rules can be generated using the boids computational model.


(a) Separation: steer to avoid crowding local flockmates


(b) Alignment: steer towards the average heading of local flockmates


(c) Cohesion: steer to move toward the average position of local flockmates



(d) A boid's neighbourhood

Figure 2.3: Demonstrates the steering behaviours of separation, alignment, and cohesion [30]

Reynolds model describes all the rules of avoiding collisions, matching velocities and centring the flock to create a natural murmuration. Reynold first proposed a paper at the 'SIGGRAPH 1987' computer graphics conference, which focused on how boids relate to computer graphics. The procedural animation was later used for flocks in films like "Batman Returns" as the colony of bats. The model runs at an efficiency of O(n^2), meaning as the size of the murmuration increases the computing power required will be greater, this will be particularly challenging in my model where I am to produce realistic graphics as well as having a natural size of a murmuration.

2.1.3.1 Separation

The separation steering behaviour is to move a boid away to avoid local crowding of its neighbours. In a model called StarDisplay, the separation is defined as the force to move in the opposite direction of the average direction of the locations of others in its neighbourhood, they have defined a small hard sphere radius around the boid where it is not attracted to any other boids and outside this radius there is a larger radius for considering its neighbours, in which the distance from the boid to its weighting in the final separation force follows a halved Gaussian distribution. [11]. Whereas another model created at University of Sheffield modelling murmurations in Mason, calculates the separation force using the negative reciprocal of the distance for each boid in its neighbourhood [39]. Both models achieve separation, however the StarDisplay model's hard sphere where it does not consider the boid into the separation calculation may result in a more natural murmuration, for example, if a starling managed to come within the radius of the wingspan of another starling, this would result in them separating in unnatural directions.

2.1.3.2 Cohesion

Cohesion is necessary to form a murmuration as it is the force that brings the starlings together. The StarDisplay model [11] defines this as the force to the centre of mass of the boids which are located in its topological neighbourhood. Unlike separation, they avoid boids in its blind angle at the back and the cohesion is stronger if the boid is at the border of the flock. Comparing this with the Mason model [39], they have only calculated the force to the average neighbour position.

2.1.3.3 Alignment

In a murmuration the starlings are seen to be aligned with each other, the alignment force will weigh the overall force in a direction towards the average direction of its neighbours. Both models simply apply a force to the boid to align it with the average forward direction of its neighbours, with the StarDisplay model still avoiding neighbours at its blind angle similar to cohesion.

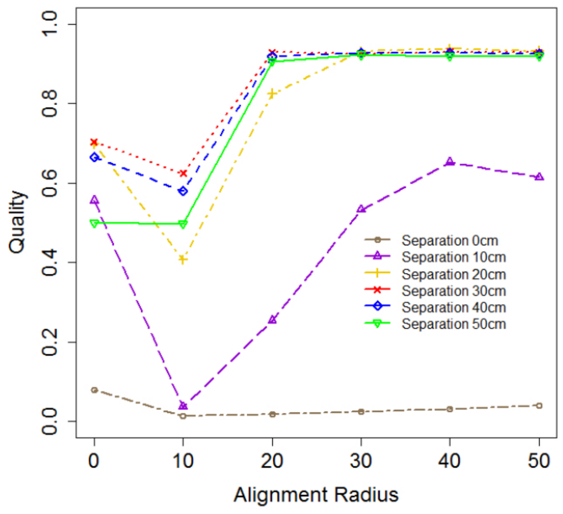


Figure 2.4: A graph showing performance metrics of separation and alignment radii [7]

Figure 2.4 suggests that having an alignment radius of 30-50cm with a separation radius of 20-50cm achieves the best performance of a simulated murmuration using Reynolds model with the frequency of collisions as the quality metric [7].

2.1.3.4 Focal Point

The focal point is an additional force of attraction alternative to the boids steering behaviours, this force is necessary as starlings murmurations have been seen to remain more-or-less static with respect to a focal point on the ground, which is generally a roosting site [3]

2.1.3.5 Weights

To conclude the steering behaviours, each behaviour generates a vector to move in, this is normalised and multiplied by a weight corresponding to the type of behaviour. The weights are used to find a balance between each steering behaviour which results in a natural murmuration. In StarDisplay, separation and cohesion are weighted as 1 and alignment is weighted at 0.5, whereas in the Mason model the weights of separation, cohesion and alignment are 6.1, 5.3 and 6.6 respectively. There are other factors with may affect the reason these weights were chosen such as a force of attraction towards a focal point, and the turning rate and speed of boids.

2.2 The Peregrine Falcon

Figure 2.5: The peregrine falcon [36]

A peregrine falcon is a large, crow-sized falcon with a blue-grey back, barred white underparts and a black head, named after the Latin ”Peregrinus” which means traveller and they can be found on all continents except Antarctica [36]. In research using citizen science to monitor murmurations it found birds of prey are only present at 29.6% of murmurations and the peregrine falcon makes up 12.6% of the predators present in their recordings.

2.2.1 Catching Prey

The peregrine falcon captures its prey mid-air by performing a high-speed dive manoeuvre where it can reach speeds of 242mph [26], making the peregrine falcon the faster than any other animal on the planet when performing the stoop [36]. Research has been conducted on the success rate of the peregrine falcon catching prey by looking at the prey morphology and sexual size dimorphism, it was found that although the female peregrine falcons are 50% heavier, the rate of success mostly depended on the variety of prey's species rather than the different sex of peregrine falcons [28]. An article on physics-based simulations of aerial attacks by peregrine falcons reveals that stooping at high speed maximises the catch success against agile prey [29], a model was formed that focuses on a single starling which reacts to a single peregrine falcon. Instead of the predator only flying around the starlings as carried out in the Mason simulation [39], the model simulates a more natural behaviour of the peregrine falcon which is to perform a controlled dive towards the prey where the speed of the peregrine falcon is controlled when chasing the starling. The peregrine falcon would fly at a specific altitude above the murmuration and then dive towards the centre of the murmuration, as the peregrine falcon nears the murmuration it would single out a starling and capture it. The model described by the article [29] would benefit my work by helping create more natural behaviour of the peregrine falcon in my model.

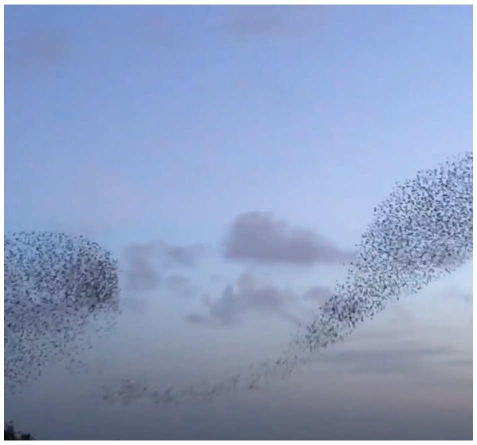
The change a falcon successfully captures a starling in a murmuration is directly correlated to the size of the murmuration. If the size of the murmuration is larger than the vision of the predators cognitive ability, it reduces the rate of attacks and success of catching prey as well as decreasing the predator's cognitive ability for survival behaviour such as staying alert [8]. Research found that if a predator hunted the prey randomly, the probability of an individual being captured on the outside of a murmuration is based on the ratio of the density of the edge compared to the centre of the murmuration [8]. Additionally, it was found that if the speed the starlings can pass along information in the murmuration is faster than the rate of motion of the predator, then the threat from the predator can be reacted to in advanced from the starlings at the edges of the murmuration [1].

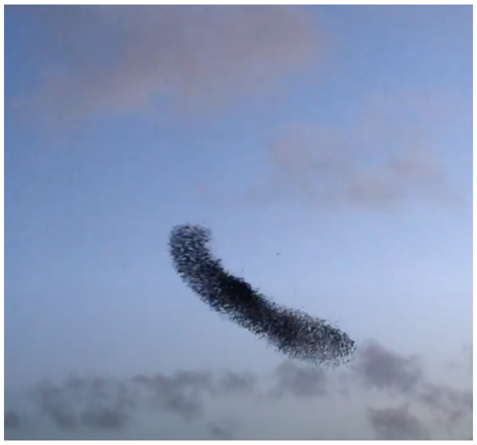
2.2.2 Murmuration patterns created by interacting with a predator

(a)

(b)

(c)


(d)


(e)

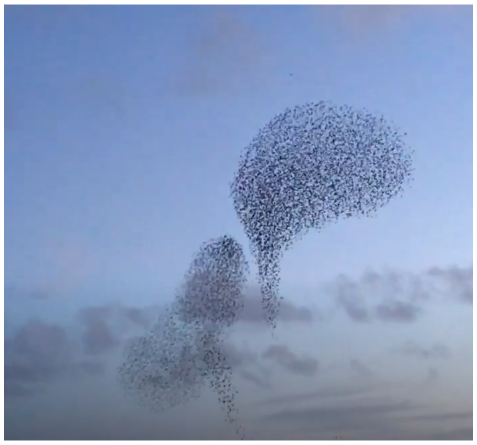

(f)

Figure 2.6: Patterns of a starling murmuration with a predator [23] [6]

As shown in the figure 4.1, when a murmuration is introduced to a predator, the murmuration creates clear patterns, which displays that the individual birds will fly away from the predator. When the predator swoops in, the murmuration often separates and forms two different groups in the opposite direction to where the predator was, shortly after the predator is a distance away from the murmuration, the two groups are formed together again. It is also noticed during the separation of the murmuration there is a thin trail of birds joining them together as shown in figure 4.1 (b) and (d), with the centre of the trail having less density of birds. Figure 4.1 (e) shows another distinct pattern where the predator is close the murmuration, as the birds are flying away from the predator, the murmuration forms a curve shape with the density of birds being highest at the inner side of the curve. These images will be useful later in modifying parameters to achieve a natural murmuration with a predator and they will allow me to compare my model to these natural murmurations.

Another way of determining if the model accuracy represents a natural murmuration is by using performance metrics of collective coordinated motion. An article proposed this in 2016 [7] suggests the performance metrics are the criteria that determine success in the behaviour of a system by using Reynolds model of boids, they proposed three quantities indicators for features exhibited by a fish school which are extension, polarisation and frequency of collision. The extension aims at finding a balance with the radius of separation to minimising the values of extension and the frequency of collisions as this directing affects the flocks cohesion. The polarisation performance metric is for finding a balance in the orientation of the flock by minimising the polarisation and frequency of collisions as the flock should be moving co-ordinately with uniformly aligned boids which do not collide. They combine these results to evaluate the global performance of the flock.

Relating to my model, it will be useful to count the frequency of collisions when modifying the parameters to create a natural murmuration as it will help to achieve a more realistic model, as the results from the metrics will allow feedback on the weights applied to separation, alignment and cohesion, a machine learning approach could be implemented here for optimise the quality of the metrics by modifying the weights.

2.3 Murmuration Models

2.3.1 Star Display

The first article which will be surveyed is Star Display [11], a model involving thousands of starlings in a murmuration. In the model, they combined Reynolds Boids rules of separation, alignment, and cohesion with three additional rules:

1. Simplified aerodynamics of flight, particularly rolling during turning. The aerodynamics involved with each starling experiences lift, drag and the force of gravity, whilst flying along a curve, they roll into the direction of the turn to place them at a certain angle to the horizontal plane.
2. Movement over the focal point ("roosting area") at a certain height
3. Using a topological range to obtain a fixed number (6-7) of neighbours for each starling to interact with

The aerodynamics involved with each starling experiences lift, drag and the force of gravity, whilst flying along a curve, they roll into the direction of the turn to place them at a certain angle to the horizontal plane.

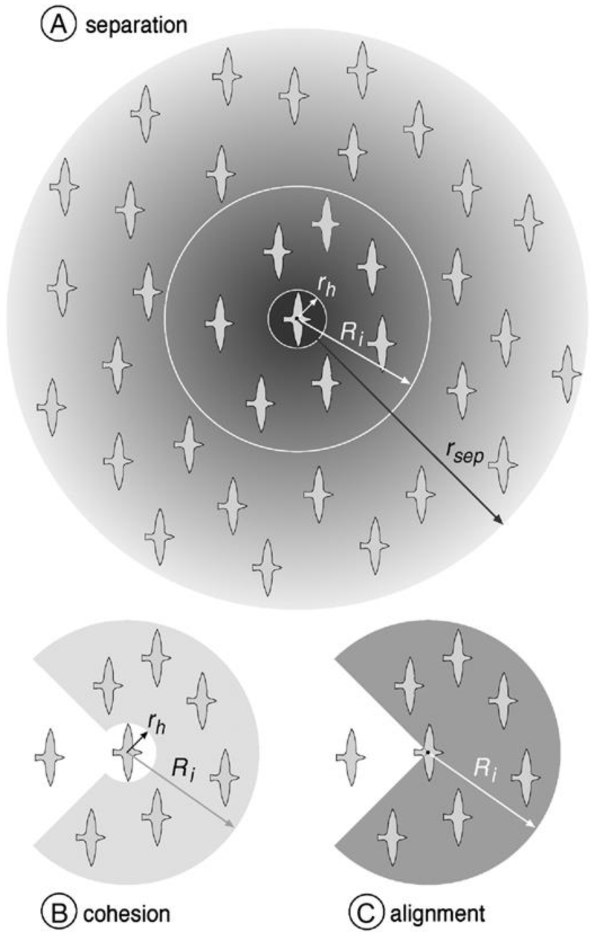


Figure 2.7: The interaction ranges for the boids rules of separation (A), cohesion (B), and alignment (C)

For separation, it is to move an individual starling away to avoid local crowding of its neighbours. In between distances r_h and r_{sep}, they follow a halved Gaussian distribution to determine the effect each neighbouring starling has on the impact of the separation. As for cohesion, they have followed previous models to avoid the selection of neighbours in its blind angle at its back, and any starling within the hard radius r_h is not considered in the calculation of cohesion, otherwise the starlings are used to calculate a force towards the average position of each starling neighbour. As for alignment, this creates a force to align with the average forward direction of its neighbours. Each of these forces are assigned a weight and the sum of them result in a "social force". The force of attraction towards to focal point is calculated using a horizontal and vertical force of attraction proportional to the starlings' distance, so the greater the distance from the focal point, the larger the force of attraction. Additionally, they have included a random force calculated using a random vector associated with a weight. Together, all the forces are summed together to form a "steering force".

2.3.2 Mason Model

The mason model is a previous dissertation at the University of Sheffield by Yu-Hao Chang [39], he modelled a murmuration of starlings interacting with a predator in Mason, a Java multi-agent simulation library to generate 3D or 2D models. In this model, he incorporated 1,000 starlings in 3D space flocking around a focal point with a falcon moving through and around them to create a natural murmuration. The individual starlings' speed and direction to form a murmuration was calculated using Reynolds Boids rules of separation, alignment, cohesion as well as additional rules which were nesting and predator proximity with varying weights applied to each individual calculation.

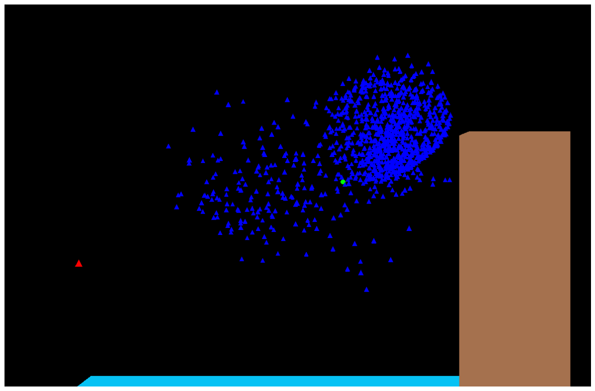
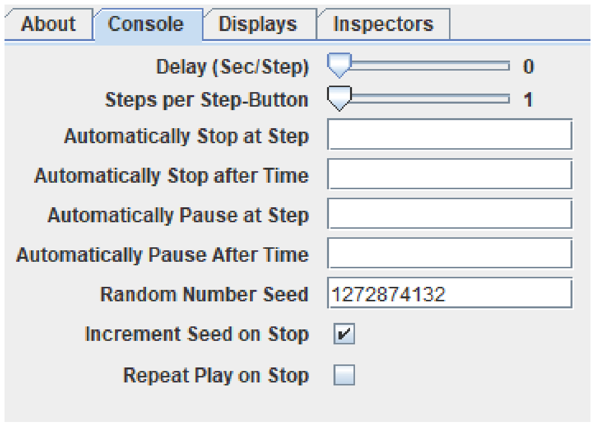


Figure 2.8: Yu-Hao Changs' model. The red triangle is the predator, the blue triangles are the starlings, the green circle is the focal point and the brown rectangle is an object

The falcons' rules are to fly towards the centre of the flock or, if a starling is within a certain range, chase a single starling. Once a starling is caught, that starling will remain motionless, and the predator will choose another target. This can be improved on as the falcon is not demonstrating the behaviour and manoeuvres of a natural falcon, according to an article on simulations of peregrine falcon attacks [31], the falcon is renowned for attacking its prey from high altitude in a fast controlled dive called a stoop to increase the success rate of catching prey.

The interaction between the peregrine falcon and the starlings could also be improved upon. In the mason simulation, the starlings only react to the predator if it is one of a fixed number of entities around the starling. This issue is more noticeable when the starlings are grouped up close and the predator is near them however just out of range to be considered by the starling, therefore causing unnatural behaviour. This can be improved on by using inspiration from the StarDisplay modal [11], in this model instead of the starlings using a fixed radius of other starlings they interact with, they use a topological range to obtain a fixed number of interaction neighbours such as 6 or 7 starlings, this would also reduce the strain on the simulating the model as there can be less starlings to consider in the calculations of separation, alignment and cohesion, it would need to adapt to always keep track of the predator and use the starlings cone of vision and find a maximum distance to which the starlings would react to the predator.

Whilst running the simulation, you can use your mouse to change the angle the murmuration is viewed at as well as zooming in and out. The model has various parameters you can set, as shown in figure 2.9. The most useful option is configuring the delay as you can increase the speed and pause the simulation allowing you to capture patterns of the murmuration and manoeuvres caused by the predator, with these you can compare the simulation with a real-world murmuration to validate the model.



Delay (Sec/Step)	0
Steps per Step-Button	1
Automatically Stop at Step	
Automatically Stop after Time	
Automatically Pause at Step	
Automatically Pause After Time	
Random Number Seed	1272874132
Increment Seed on Stop	✓
Repeat Play on Stop	☐

Figure 2.9: Adjustable parameters of the Mason murmuration model

The model is a good attempt of natural murmuration; however it is not visually pleasing. there are no textures and only one colour associated with each entity and no shadows to assist with the depth perception of the model, rendering it two-dimensional until you move the view around. The models background properties such as the brown cube and blue floor does not reassemble the natural scenery of where starling murmurations take place. In this work, it should attempt to achieve more natural scenery by including a sky, shadows, and textures to the landscape as well as the models of the starling and peregrine falcon.

2.4 Modelling Swarms

This section will be outlining an overview of the modelling of swarms and critiquing the most up to date swarm models developed specifically using Unity as this will be the software which this project will use.

2.4.1 Overview of Swarms

Swarming is inspired from the behaviour of self-organized and decentralized systems that cooperate to do a special task. Decentralization means that the swarm has no central leader or boss and each member does its work with a kind of imitation [22].

2.4.2 Bogdan Codreanu's Model

In Bogdan Codreanu's Unity model of a murmuration with a predator [5] he used Unities Entity Component System (ECS) to handle the instantiating of boids, the ECS is a Unity Data-Oriented tech stack, it contains entities, the data associated with the entities and systems – the logic that transforms the component data from its current state to its next state [35].

The murmuration involves 100,000 boids at once as shown in Figure 2.10 and follows Reynolds boids algorithm by calculating the forces of separation, alignment and cohesion for every boid. In Figure 2.10 due to the larger number of boids, having realistic 3D models would not be required if the simulation was viewed from a distance, however in Figure 2.11 it shows the boids closer up where the quality of program diminishes as they have very simple shapes, as such it wouldn't be practical to use this model in any games or movies, unless from a far distance.

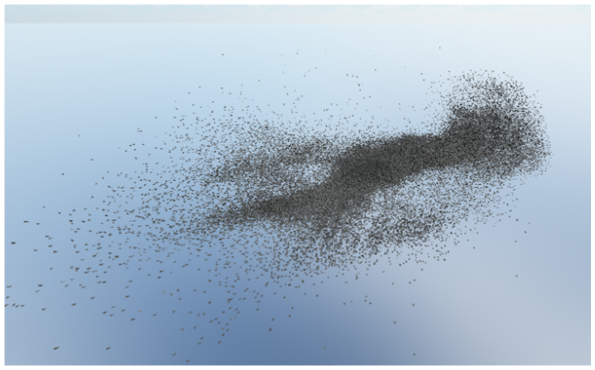


Figure 2.10: Image to show Bogdan Codreanu's Unity model of a murmuration with 100,000 boids 2.10

Bogdan Codreanu's Unity model's predator is controlled by the user, the predator itself is in the shape of a red pointer as shown in Figure 2.11, you can move the predator by using your mouse to change its direction with pressing the W key to confirm the change in the direction, this works well as you can demonstrate different escape manoeuvres of the murmuration accurately, however it would be better if the predator movement was automatic so you could view the entire murmuration to capture the escape patterns as a whole instead of being restricted to the predators' view.

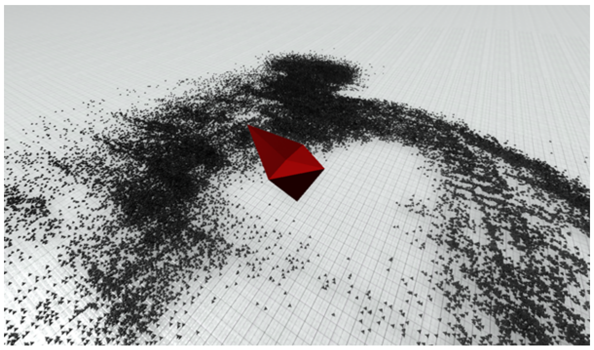


Figure 2.11: Image to show Bogdan Codreanu's Unity model of a the predator interacting with boids 2.10

In order to simulate 100,000 boids, Bogdan made use of a cell space partitioning algorithm which organised the boids into cells which can be used to obtain the closest neighbours around a boid faster than compared with an O(n^2) algorithm which looks at every boid find the closest boids, this literature survey will later review a paper on a cell space partitioning algorithm. Additionally, the program uses multi-threading to further increase the performance.

2.4.3 Skooters' Model

Similarly to Bogdan Codreanu's Unity model, Skooters' model [32] also makes use a cell space partitioning algorithm as well as multi-threading to reduce the performance cost of running the algorithm. Skooters' model aims to create a screensaver using Reynolds boids algorithm, the boids themselves are shaped as snakes and are in multiple colours which create moving patterns with a snapshot of the patterns show in Figure 2.12.

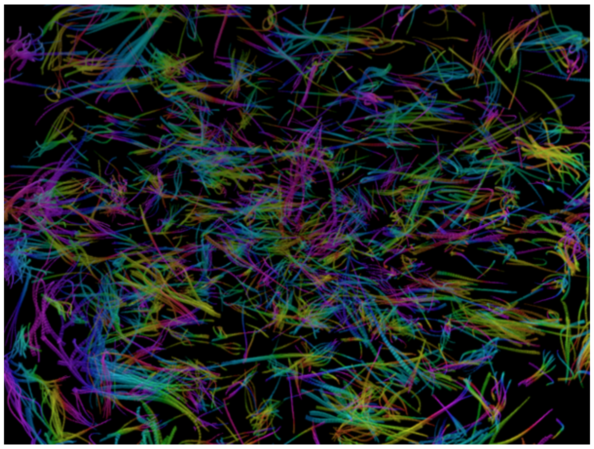


Figure 2.12: Image to show Skooters' boids screensaver

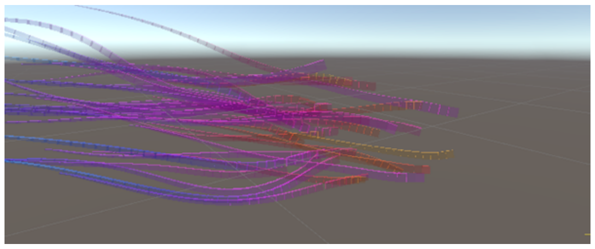


Figure 2.13: Image to show Skooters' boids in a local group [32]

The boids move with each other in accordance to Reynolds boids algorithm, however, the boids appear to move in many different groups as shown in Figure 2.13 and not together as you would see a flock of birds, this is due to the program having no focal point of attraction to ensure the boids stay around each other. The programs demonstrates a great use of Reynolds algorithm as a swarm with good optimisation techniques which this project should make use of.

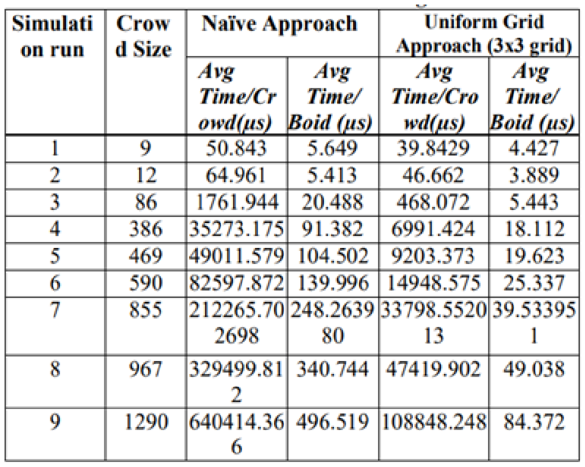
2.4.4 Swarm Optimisation

Both up to date Unity models of swarms make use a cell space partitioning algorithm to improve their performance, a paper by Mavhemwa Prudence explains how a cell space partitioning algorithm can be used to improve Reynolds Boids Algorithm in a gaming environment [19]. The problem arises from when trying to find the closest neighbours to every starling as a naive approach would be to evaluate the whole crowed for every agent, resulting in a computationally expensive O(n^2) algorithm.

The cell space partitioning algorithm, also known as spatial subdivision works to subdivide a given geometrical space into smaller sub-spaces or cells, which can then be used to speed up proximity or locality queries [19]. This report will focus on the uniform spatial subdivision, the idea is to partition the 3D space into many uniform cells and then to obtain the nearest neighbours with boids within the same cell, compared with other spatial subdivision techniques such as object dependant spatial structures which considers the distribution of the Boids which generate irregular subdivision, the uniform spatial subdivision's advantage is the grid does not change over the simulation process making it easier to implement, however it does not allow the structure to adapt with the distribution of Boids.

The algorithm works as follows: at the beginning of the simulation as the Boids are created, they will be distributed into their cells depending on their location. Every time a boid moves the program will perform a check to see if the Boid has moved into a new cell and update the cell if necessary, then when obtaining the nearest neighbours, it will fetch all the boids within the same cell, which avoids the O(n^2) complexity.

Results were taking from running 5000 samples of the naive algorithm against the uniform grid algorithm which are shown in Figures 2.14 and 2.15. With increasing number of Boids, the uniform grid is shown to be much more computationally efficient.



Simulation run	Crowd Size	Naïve Approach		Uniform Grid Approach (3x3 grid)	
		Avg Time/Crowd(μs)	*Avg Time/ Boid (μs)*	*Avg Time/Crowd(μs)*	*Avg Time/ Boid (μs)*
1	9	50.843	5.649	39.8429	4.427
2	12	64.961	5.413	46.662	3.889
3	86	1761.944	20.488	468.072	5.443
4	386	35273.175	91.382	6991.424	18.112
5	469	49011.579	104.502	9203.373	19.623
6	590	82597.872	139.996	14948.575	25.337
7	855	212265.702698	248.263980	33798.552013	39.533951
8	967	329499.812	340.744	47419.902	49.038
9	1290	640414.366	496.519	108848.248	84.372

Figure 2.14: A table of results comparing a naive and uniform grid approach to Reynolds Boid algorithm [19]

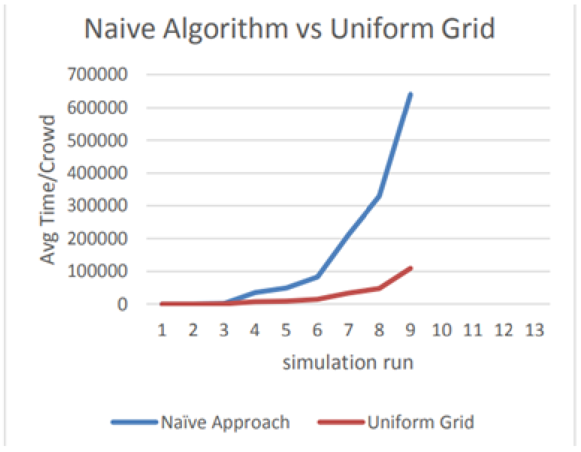


Figure 2.15: A graph comparing a naive and uniform grid approach to Reynolds Boid algorithm [19]

2.5 Summary

The literature survey has been important to understand all the elements involved in the project, for example understanding the behaviours of starling murmurations and predators in addition to different techniques used in previous Unity simulations to optimise the Reynolds Boids algorithm for performance which will be applied to this projects' simulation. By reviewing previous models this project can avoid the mistakes they have made and use the best parts of their model in this model. The next step is outlining the requirements and analysis of the project to form objectives which will need to be completed in order for the model to be successful.

Chapter 3

Requirements and analysis

This projects' main objective is to create an accurate and visually pleasing 3D model of a predator interacting with a starling murmuration. To achieve this, this requirements and analysis section will detail all the aims and objectives in order for this project to be successful and will analyse the individual parts in detail.

3.1 Models

Accurate 3D models of the starling and falcon will need to be created in order make the project look realistic and it would also provide this project with addition real-world applications such as using this murmuration in games or movies. During the simulation, the models will need to have basic animations to bring the models to life.

3.2 Starlings

Once the models have been created, the simulation will need to randomly place all the starlings together with random directions and then the following rules can be applied for the starlings to create the flocking motion.

3.2.1 Starling Rules

- Steer away from the falcon
- Separation - steer away from its neighbours
- Alignment - steer in the same direction as its neighbours
- Cohesion - steer towards the centre of its neighbours
- Focal point - steer towards the focal point

The starlings will follow boids rules of separation, alignment, and cohesion, in addition to a force to keep them over the focal point and to move away from the peregrine falcon. The different weights corresponding to the different steering forces will need to be fine-tuned to reassemble a natural murmuration with the use of videos and images from real starling murmurations.

3.3 Predator

Once a natural murmuration is achieved, the peregrine falcon is introduced to the simulation. The peregrine falcons' main objective is to catch a starling like it would in real life, to achieve this the falcon needs to fly above the murmuration, select a random starling and stoop towards the starling in order to catch it, this should cause explosive patterns in the murmuration as a result with will be used in this model's evaluation to determine if this project was successful.

3.4 Visuals

A main aspect of this project is having a model that is visually pleasing, to achieve this, the implementation phase should consider adding the following to the 3D simulation:

- A landscape
- A sky
- Objects such as trees
- Shadows from the models
- Weather such as clouds, rain or mist

Additionally, another realistic condition is wind, this would be a force effecting both the starlings and falcon by moving them in the wind's direction adding randomness to the simulation, this could be visually represented by white streams of air moving in the direction of the wind.

The starling and falcon should be modelled in 3D with accurate proportions and colours of their real-life counterparts along with animations to supplement the realism.

3.5 Performance

The simulation should aim to achieve 1,000 starlings at once in a murmuration, in order to achieve this, the following performance optimisations may be required.

- The basis of Reynolds Boids algorithm is to first find the closest starling, a brute force approach to this is comparing every starling with every other starling to find the closest starlings, which has a complexity of O(n^2). Instead, the simulation should implement a uniform spatial grid to keep track of where each starling is in order to obtain the closest starlings in a more computationally efficient way.

- Multi-threading will allow the algorithms to run simultaneously on different cores which will help greatly to improve the performance.
- The models of the starlings and Predator must have a low polygon count to allow many of them to be used in the simulation without affecting the performance too much.

3.6 Constraints

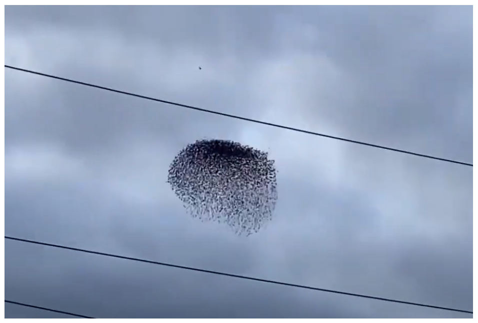
Due to time constraints, it may not be possible to cover some of the additions to improve the visuals of the simulation. Moreover, there are other additions to the project which could improve the model's realism which I have not included in this requirements and analysis section, for example, using physics to calculate the banking angles of the starlings and rolling in and out of turns, I have not included these due to them being complex and it may take too long to implement.

3.7 Model Evaluation

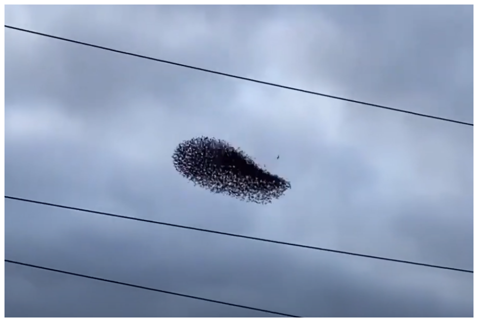
In order to evaluate the work to establish how successful the work is, the simulation will need to be compared and analysed with real-life data.

Figure 3.1 can be used to evaluate the murmurations interaction with a predator, it outlines the different stages in how the predator can shape the murmuration creating an explosive pattern, these images were taken from a video in Marin County [10]. Replicating an explosive pattern, with the densities of the murmuration changing with the position of the predator, similar to the snapshots taken in the video, will allow a successful evaluation of the interaction between the murmuration and predator.

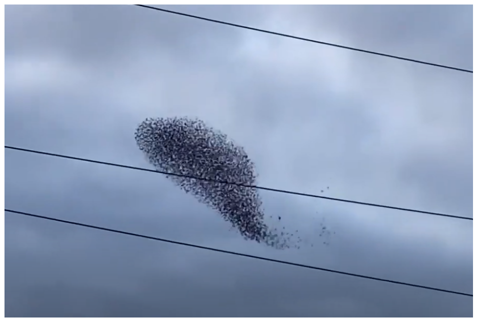
(a) Murmuration



(b) Predator stooping down from above causing a high density of starlings on the top side



(c) Predator targeting a starling on the right, the murmuration is closer together with a higher density of starlings on the side of the predator



(d) Murmuration expands towards the left and tries to stay together in a curve shape but some are separated

Figure 3.1: Images taken from a video of a starling murmuration with a predator [10]

Additionally, the model should be evaluated with using the metrics of consistency of extension and consistency of polarisation, as these metrics will provide an idea of how well the murmuration is performing in regard to minimising number of collisions with how far apart each starling is and the collective directions of starlings.

Chapter 4

Design

The design section expands on parts of the project described in the requirements and analysis section, explaining the requirements in more detail and developing suitable designs which satisfy these requirements.

4.1 Models & Animations

Creating accurate 3D models of the starling and falcon will allow further practical uses for the simulation, such as use in movies or video games. This project will use Blender to create the models and animations as the modelling software is compatible with Unity, alternatives would be paid software such as Maya, 3ds max or 3DTopo, or Sketchup, a free 3D modelling software, however Sketchup only works well for simple shapes unlike the modelling of birds.

With Blender chosen as the software, the sizes and weight of the starling and falcon are shown in Table 4.1.

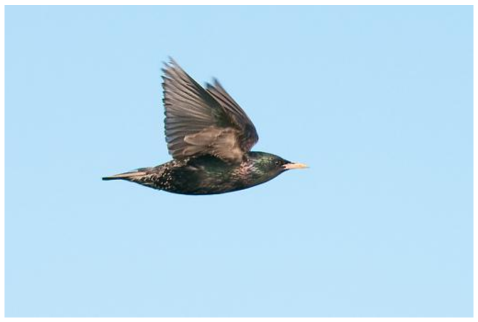
Model	Size (cm)	Wingspan (cm)	Mass (g)
Starling	21.5	37 - 42	75 - 90
Peregrine Falcon (Male)	34 - 58	74 - 120	330 - 1000

Table 4.1: Size, wingspan and weight of the starling and peregrine falcon [34][36]

The shapes and colours of the starling and peregrine falcon will be taken from images found from the internet. For the starling, Figure 4.1(a) shows the colours and shape of the starling, most notably, the beak is bright yellow, the body has a green tint with white dots and the body size appears large compared with the starlings small wings. Figure 4.1(b) describes the peregrine falcon as having a white/yellow beak, white feathers on its bottom with black feathers on its back, the falcon as long wings as well as a long body.

(a) Starling [37]

(b) Peregrine Falcon [36]


(c) Starling in forward flight [2]

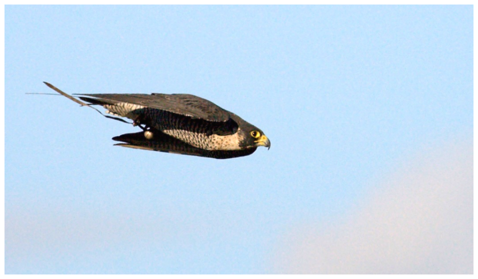

(d) Peregrine Falcon in a stoop [21]

Figure 4.1: Images of a starling and peregrine falcon

Figures 4.1(c) and 4.1(d) will be used to create animations for the starling and peregrine falcon, the most important animations which should be present in the simulation are the flapping of wings of both models and the peregrine falcon folding in its wings as it stoops on the prey. In regards to the wing-flapping rate of each bird, the starling has 4.5 wing beats per second [33] and the Peregrine falcon has 1.5 wing beats per second.

4.2 Starlings

The starlings will need to be placed into the simulation and distributed evenly around the focal point with random rotations as this would ensure the simulation is not deterministic. Every tick of the simulation, each individual starling will need to calculate its next position and move there, it will do so by accelerating towards the overall direction force given by the summation of the normalisation of these individual forces:

- A force of repulsion from the falcon, if the falcon is within a distance of 10 metres from the starling, the repulsion force will use the inverse of the distance in the calculation of the force so that the closer the falcon is, the greater the repulsion force will be.
- A force of separation to steer away from its neighbouring starling birds, this force is similar to the repulsion force from the falcon, as the force of separation needs to calculate the repulsion force from all its neighbours.
- A force of alignment to steer in the same direction as its neighbours, this force will need to calculate the average velocity of all its neighbours.
- A force of cohesion to steer towards the centre of its neighbours, this force will need to calculate the average position of all its neighbours and create a force to point towards that direction.
- A focal point force to steer towards the focal point, the force will need be a force of attraction towards the focal point by also using the inverse distance between itself and the focal point so this force is greater as the starling is further away from the focal point.

Figure 4.2 shows how each individual starling will operate, at first it calculates the five forces explained above, normalises each force so their magnitudes are equal and then multiply each by their respective weights, finally adding these together and normalising again will give the overall direction force the starling should move towards.

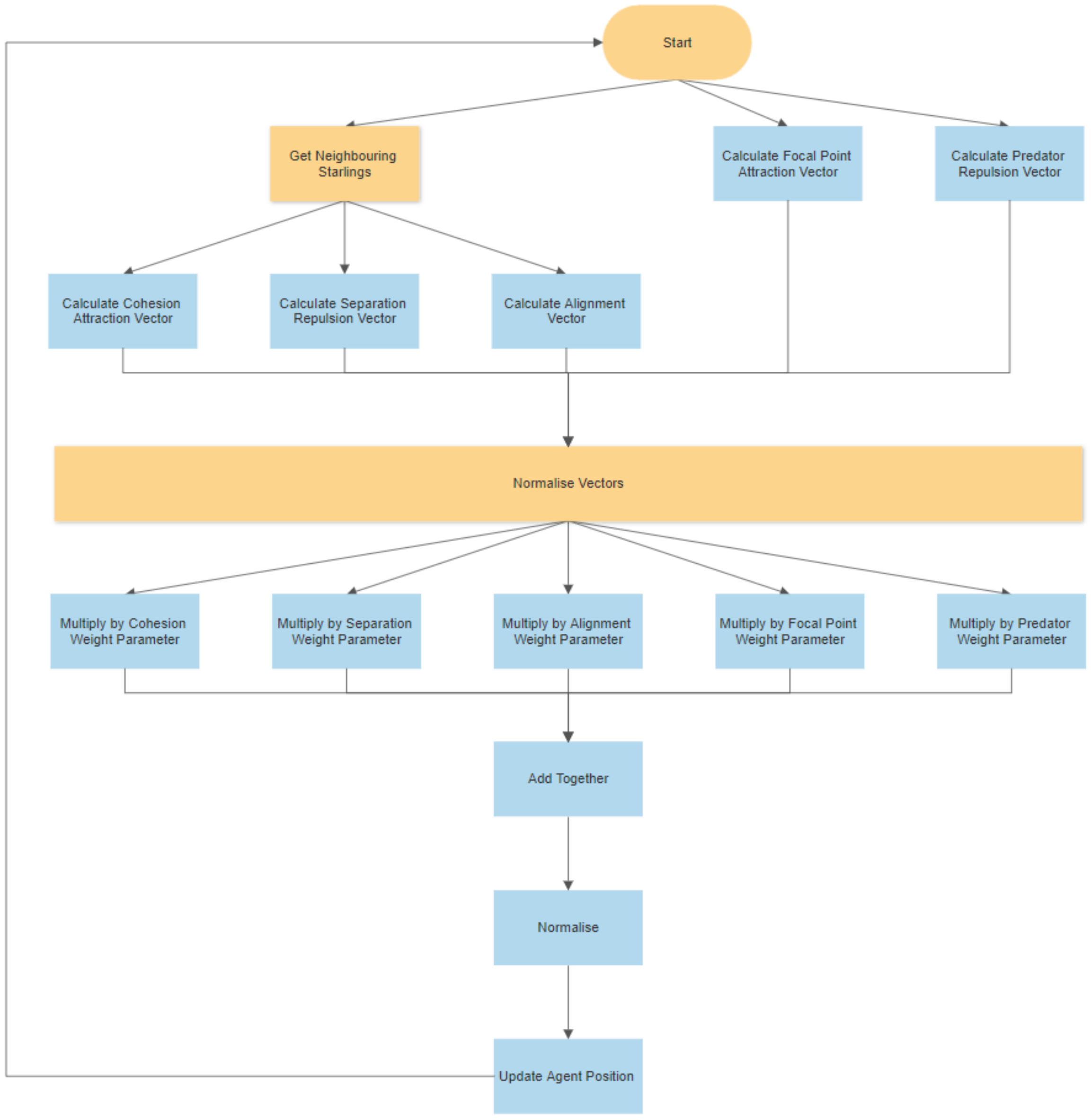

Start
Get Neighbouring Starlings
Calculate Focal Point Attraction Vector
Calculate Predator Repulsion Vector
Calculate Cohesion Attraction Vector
Calculate Separation Repulsion Vector
Calculate Alignment Vector
Normalise Vectors
Multiply by Cohesion Weight Parameter
Multiply by Separation Weight Parameter
Multiply by Alignment Weight Parameter
Multiply by Focal Point Weight Parameter
Multiply by Predator Weight Parameter
Add Together
Normalise
Update Agent Position

Figure 4.2: Flow chart of the starling

4.2.1 Starling Neighbourhood

The neighbours in which the starling rules are applied to will be found using the nearest 6-7 closest neighbours within a radius, outside of its current position and within its cone of vision, as shown in figure 4.3. The ideas of the neighbour selection process were taken from Reynolds Boids model [30] and Stardisplay [11], Stardisplay used a fixed number of closest starlings with a small hard sphere around the starling and a fixed radius to consider as neighbours and Boids introduced the cone of vision to exclude neighbours in its blind angle.

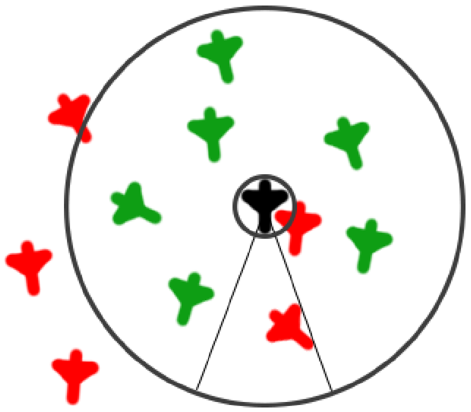


Figure 4.3: Diagram to show how the 6 closest neighbours are selected around a starling. The green birds are the chosen neighbours and the red birds are not counted as the neighbours.

4.2.2 Starling Parameters

Parameter	**Description**	**Example Value**
Neighbour Radius	Size of the radius to obtain neighbours	2m
Separation Weight	Value to scale the separation vector	1
Cohesion Weight	Value to scale the cohesion vector	1
Alignment Weight	Value to scale the alignment vector	1
Focal Point Weight	Value to scale the focal point vector	1
Predator Weight	Value to scale the predator repulsion vector	1
Predator Detection Distance	Maximum distance between the starling and predator for the predator repulsion force to be calculated	10m
Neighbours to Consider	Number of neighbours used to find next position	7
Maximum Speed	Maximum speed of the starling	4m/s
Acceleration	Acceleration of the starling	3m/s^2
Blind Angle	Size of the angle behind the starling to not consider as a neighbour	9°
Hard Sphere Radius	Radius of the sphere around a starling to not consider as a neighbour	0.2m

Table 4.2: Parameters required for the starlings

4.3 Peregrine Falcon

When the simulation starts, the peregrine falcon will need to be placed in the simulation away from the murmuration as it ensures the murmuration can be formed properly before any predator-murmuration interaction.

The predator will be constantly stooping into the murmuration to create the predator-murmuration patterns which is what this project aims to simulate. In order to achieve this, the predators' behaviour will need to repeat a cycle of stooping onto a starling, moving above the murmuration and waiting at above the murmuration for some time before stooping again. These three parts of the peregrine falcon' behaviour are explained in further details below:

- The peregrine falcon will move to a certain height above the centre of the murmuration.
- The peregrine falcon will select a random starling as a target after a certain amount of time after the falcon has been gliding over the murmuration.
- The peregrine falcon will stoop on the targeted starling with a greater acceleration in order to pass through the murmuration to replicate the predator-murmuration patterns.

The three stages can be better visualised by Figure 4.4, showing the each stage described above, in a flow chart. After stooping on a starling, the peregrine falcon will start the process again, therefore, in an automatic cycle which should continuously make murmuration-predator patterns.

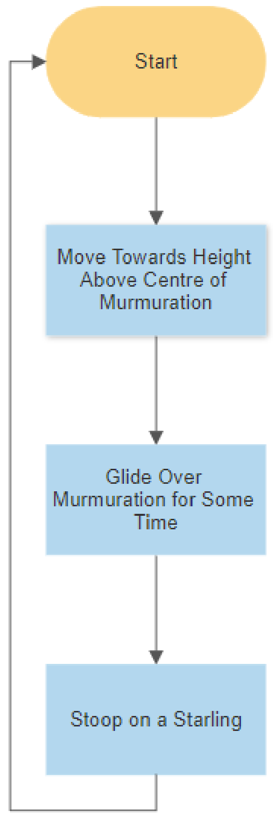

Start
Move Towards Height Above Centre of Murmuration
Glide Over Murmuration for Some Time
Stoop on a Starling

Figure 4.4: Flow chart of the peregrine falcon

4.3.1 Peregrine Falcon Parameters

For each stage of the peregrine falcons' behaviour, there will need to be parameters such as the speeds at each part and the time the Peregrine falcon will remain gliding for before stooping down on a starling. These parameters are shown in Table 4.3.1 with example values to help understand which each parameter will be.

Parameter	**Description**	**Example Value**
Glide Time	How long the peregrine falcon will glide for over the murmuration until it stoops	5s
Glide Speed	Speed of the glide	3m/s
Re-position Maximum Speed	Maximum speed for moving back to the stop of the centre of the murmuration	4m/s
Re-position Acceleration	Acceleration for moving back to the stop of the centre of the murmuration	2.5m/s^2
Stoop Maximum Speed	Maximum speed of the stoop	10m/s
Stoop Acceleration	Acceleration of the stoop	5m/s^2
Target	The starling to stoop on	starling object

Table 4.3: Parameters required for the peregrine falcon

4.4 Evaluation Metrics

Once the model has been implemented, metrics will need to be generated to give an understanding of how the model is performing and provide feedback on the weights of the forces. The metrics which will be used are the consistency of extension and the consistency of polarisation [7], together these metrics can be combined into a metric of quality which gives an overall score of the murmuration between 0 and 1, with 1 being the optimal score.

Figure 4.5 shows the consistency of extension equation, where t is the time step, $cen(t)$ is the average position of all the starlings, $p_i(t)$ is the position of starling i, $\cdot col(t)$ is the number of colliding starlings and max_e is the maximum extension of any particular starling. This calculates a normalised value of extension between 0 and 1. As the result gets closer to 1, it aims to minimise the extension and number of collisions.

$$cns_{ext} = 1 - \frac{\sum_{i=0}^{m} \|cen(t) - p_i(t)\| + max_e \cdot col(t)}{max_e \cdot n}$$

Figure 4.5: Consistency of extension equation

Figure 4.6 aims to find the consistency of polarisation, where it first finds the difference between the average movement direction of the murmuration between the individual direction of every starling, likewise with the extension equation, polarisation also penalises the collisions between starlings.

$$cns_{pol} = 1 - \frac{\sum_{i=0}^{m} \left(\frac{180}{\pi} arccos \left(f(t) \cdot \left[\frac{\mu_p(t)}{\|\mu_p(t)\|} \right] \right) \right) + \rho + col(t)}{180 \cdot n}$$

Figure 4.6: Consistency of polarisation equation

Finally, the equations of extension and polarisation are combined into a metric for quality as shown in Figure 4.7 where it takes the weighted sum of each using the coefficients σ and γ as weights for the consistency of extension and polarisation respectively where $\sigma + \gamma = 1$.

$$qlty(t) = \sigma \cdot cns_{ext}(t) + \gamma \cdot cns_{pol}(t))$$

Figure 4.7: Quality metric equation

These metrics will provide a quantitative method of evaluation to analysis the model as it provides an indication of how well the murmuration is performing. The metrics can be used to feedback into the weights of separation, cohesion and alignment in order to achieve a higher quality metric, a machine learning algorithm could be used here to maximise the metrics by modifying the weights however, due to time constraints this will not be implemented.

4.5 Uniform Spatial Grid

A uniform spatial grid is an important aspect of this project, as it will allow a greater number of starlings to be present in the simulation at one time due to its efficiency in obtaining each starlings neighbours to perform the calculations on starlings to find out where the starlings next position will be.

In order to implement a grid, there will need to be a data structure which can store a list of starlings with a grid reference, a cell size and grid size, with these, the grid must implement the following functions:

- Given a starling and its position in 3D space, calculate its grid reference and store it. This is calculated shown in Figure 4.5, for this to work the grid size must be larger than the cell size.
- When a starlings position is updated, the grid will need to calculate its grid reference at its new position using the equation in Figure 4.5 and adjust its grid reference if it has changed.
- The grid will need to return the the neighbours given a starling. A naive approach for this would be to just simply return all the starlings in the same grid reference as

its neighbours, however if the starling we are fetching the neighbours for is on the edge of a grid, then it may miss potential starlings which should be its neighbour. To overcome this issue, the program should return all the starlings in the grid cells around the starling as well as the cell the starling currently occupies.

$$round(pos.x/cellSize)$$
$$+(round(pos.z/cellSize)) * gridSize$$
$$+(round(pos.y/cellSize)) * gridSize * gridSize$$

Figure 4.8: Equation to get the grid reference of a position in 3D space

To determine values for the cell and grid size, we must try to minimise the cell size to optimise performance also while having a large enough cell size to satisfy the neighbour radius distance. For example, if the neighbour radius distance is 2m, a cell size of 2m will ensure all starlings within 2m of its position will be considered as its neighbours, as if the starling was located on the edge of a cell, the grid would return the starlings in the surrounding cells therefore every starling within 2m will be found.

4.6 GUI

A graphical user interface will be necessary as to allow quick adjustments to the parameters of the starling and peregrine falcon whilst the simulation is running, this also allows a potential user to demonstrate how different parameters can affect the murmuration.

The interface should be accessible while the simulation is running and should not obscure the view the simulation. The types of parameters which can be adjusted can be split into two separate menus, one for flock settings and one for predator settings, these menus should be accessed by buttons in the corner of the screen as the simulation is running. Inside the flock settings menu, in addition to the flock parameters, there should also be an option to change the number of starlings in the simulation which will reset the simulation with the desired number of starlings when applied. When adjusting parameters, it will be useful to reset them to how they were before any changes were applied, as well as the option of saving a set of parameters locally on the machine so you do not lose any settings which were found to be ideal for the simulation. Another option to have in a menu would be to pause the simulation, this will allow you to inspect the positions of all the birds and take screenshots showing the patterns created.

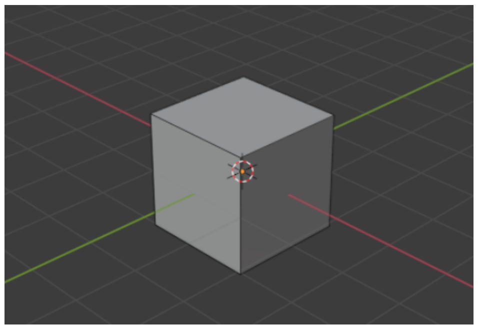
Chapter 5

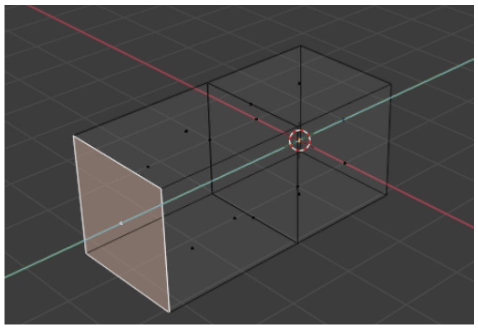
Implementation and Testing

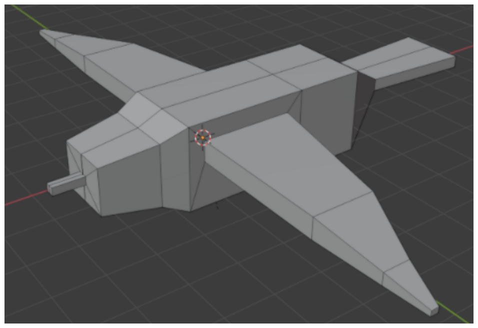
5.1 Models & Animations

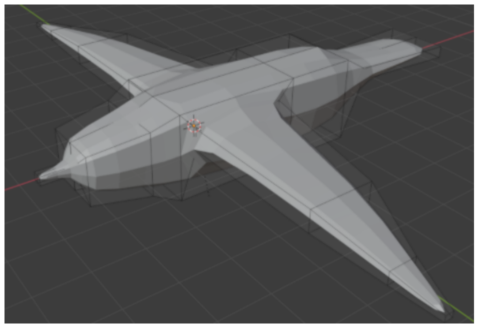
5.1.1 Models

With the use of Blender, a 3D model of a starling and peregrine falcon were created to use in the simulation. Outlined in Figure 5.1 were the steps in creating the 3D model of a starling, it began with a cube and then shaping the cube by extruding its faces along with a symmetry tool in the middle to keep the model symmetrical. After the model was similar to the general shape of a starling, a smoothing effect was applied to make the model appear more realistic and then colours were added to match those of a starling, which were most notably, a bright yellow beak, followed by dark purple and dark green with some small white dots.


(a) The cube blender gives you to start modelling


(b) Extruding faces of the cube to form a starling


(c) The final shape of the starling


(d) Applying a smoothing effect

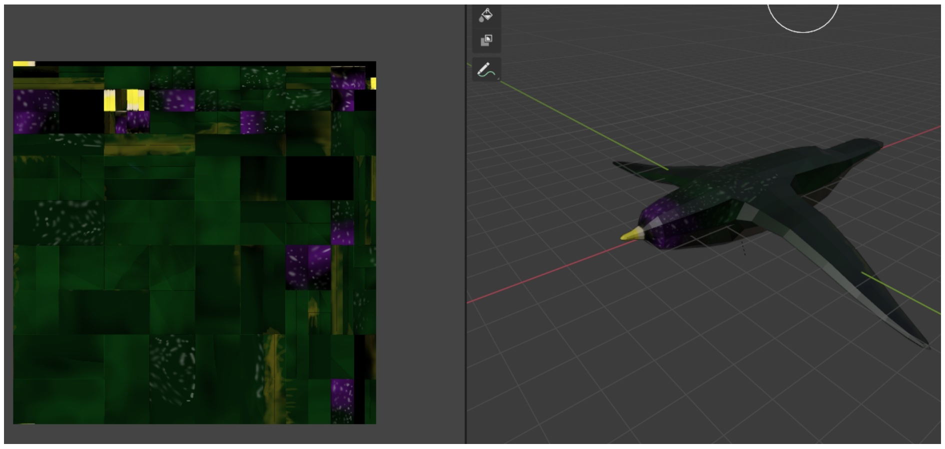
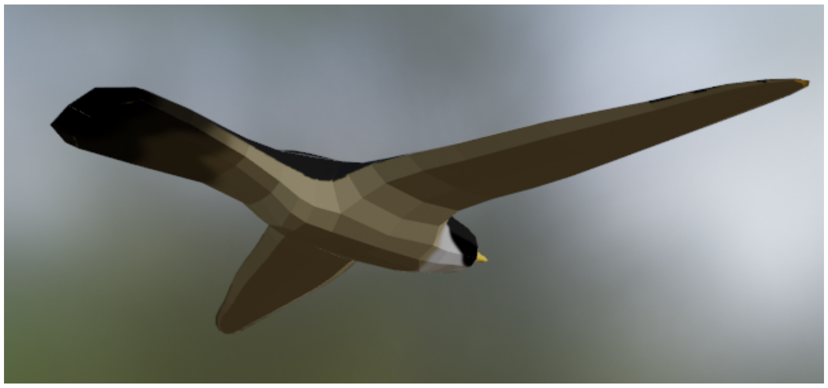
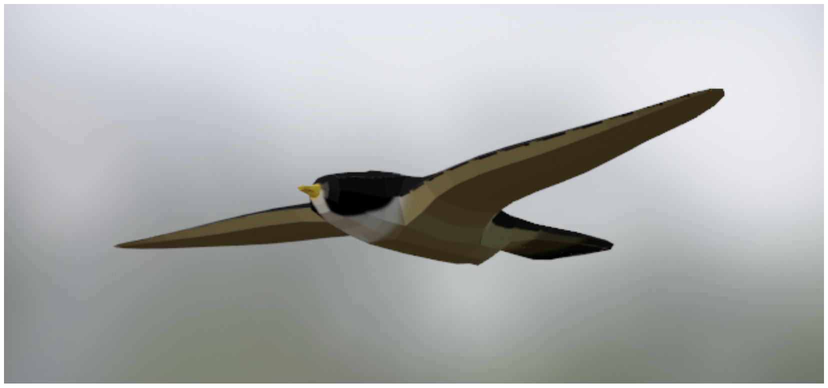

(e) Adding a texture using a brush tool

Figure 5.1: Images of a creating a 3D model of a starling in Blender

In Figure 5.2 you can see the final model of the peregrine falcon. Compared with the starling model, its wingspan and tail is larger and the colours are different to represent a falcon.



(a)



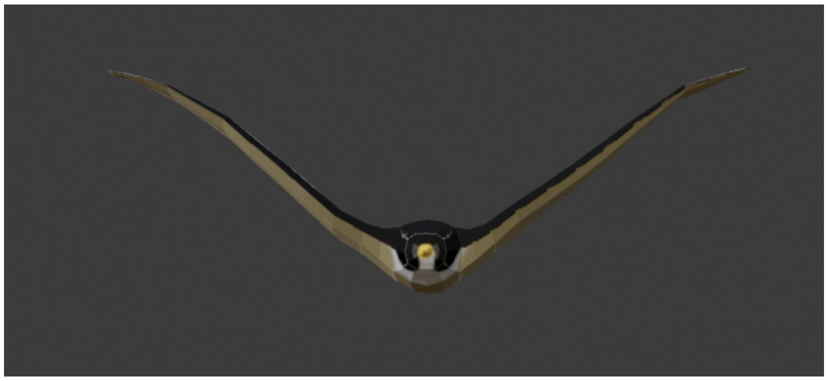
(b)

Figure 5.2: 3D model of the falcon created in Blender

5.1.2 Animations

As for the animating the models, the starling and peregrine falcon both have animations to flap their wings and the peregrine falcon has an additional animation to fold in its wings when it performs a stoop. Figures 5.3(a) and 5.3(b) show snapshots from the animation of the peregrine falcon flapping its wings, the animation was created by mapping the wings rotations into key frames using Blenders animation tool, in the case for the wings flapping there were three sets of key frames inserted at different times, one for the wings at their lowest point, another for the wings at their highest point and finally one set of key frames for the wings as their default state shown in Figure 5.2. For the peregrine falcon wings folding in, as shown in Figure 5.3(c), there were only two sets of key frames for this animation, one of the default state and one for the wings folded in.

(a) The peregrine falcon with its wings at its lowest point


(b) The peregrine falcon with its wings at its highest point

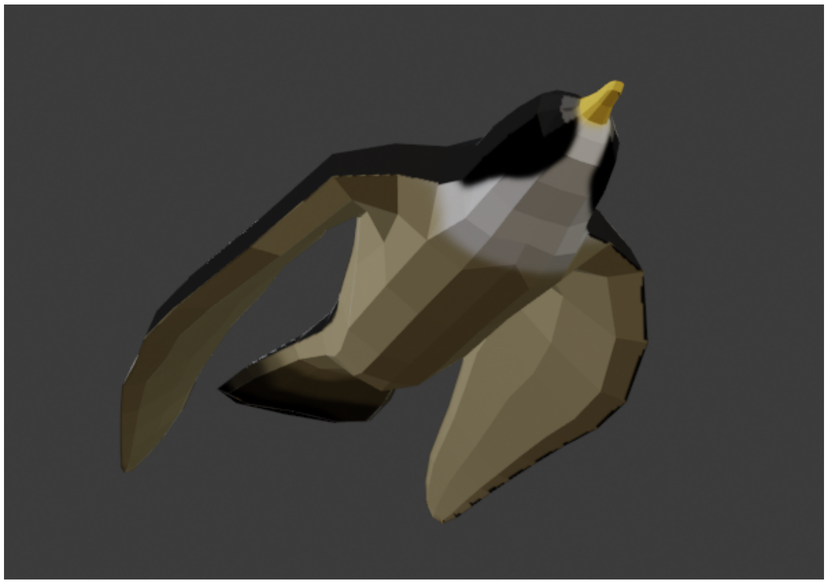

(c) The peregrine falcon folding its wings

Figure 5.3: Images showing snapshots in the animation of the peregrine falcon

5.2 The Murmuration

Once the models were created, imported into Unity and scaled to their size, the murmuration could now be implemented. At first, three starlings were placed into the simulation manually in order to begin implementing the forces of separation, alignment, cohesion and a force towards the focal point in a controlled environment, normalising the addition of these forces created an overall force which is used to tell the starling which direction to head too next. This overall force is then multiplied by the starling's acceleration parameter and added to its previous velocity to obtain a new velocity to move the starling with, finally, it updates the direction the starling is looking at to the new velocity. Once the mechanics of the movement and forces were satisfied, more starlings were introduced to the simulation to test obtaining the nearest neighbours. To include more starlings in the simulation, a flock initialising algorithm was created to add any number of starlings into the simulation, it works by placing every starling inside a sphere at a random point with the focal point being the centre, in order to ensure all the starlings were evenly distributed about the sphere, the sphere's radius expands depending on the number of starlings and makes use of a density constant to scale the radius. Now more starlings were in the simulation, an algorithm to obtain the nearest neighbours was created, this algorithm took inspiration from the StarDisplay model [11], in that it makes use of a blind angle which does not consider neighbours in a certain angle behind the starling, as well as a hard sphere, which does not consider starlings within a certain radius of itself. The algorithm to obtain the nearest neighbours at first was very trivial, it will loop through every other starling and keep record of the closest 7 starlings and when it is finished, it outputs a list of the closest 7 starlings to be used to calculates the forces of separation, cohesion and alignment.

At first, only 200 starlings could be simulated without any performance issues, as the algorithm was $O(n^2)$ every tick had to check 40,000 distances between every starling. This figure was greatly improved to 1,000 starlings being simulation at once with no performance issues by implementing the uniform spatial grid. By keeping record of which cells contained which starlings, the algorithm turned into a complexity of O(kN) where k is the number of starlings fetched from the cells surround it. Finally, implementing multi-threading in Unity on the O(kN) nearest neighbour function and the calculation of the forces, allowed the simulation to run 2,000 starlings at once, which further satisfies the requirement for the number of starlings found in real murmurations where the minimum is 1,000 starlings.

5.2.1 Testing the Murmuration

To test the murmuration once the forces are correctly implemented, the weights need to be fine-tuned to make the murmuration move together with the separation and alignment found in real murmurations, such as in Figure 5.4, before introducing the predator into the simulation. The process in finding the best weights manually are shown in Figure 5.5 in which the best weights found for separation, cohesion, alignment and the focal point were 1.5, 3, 2.5 and 5 respectively. The focal point's weight is the highest which is because the other weights will often cause the starlings to fly away in one direction which will lead them away from the focal point which is shown in Figure 5.5(b) where doubling the cohesion weight caused them to fly away from the focal point as two separate murmurations, so it is important the focal point weight is high so the starlings do not leave the murmuration.

In testing the murmuration, an unusual behaviour was found where the murmuration would move in a vertical circle around the focal point, to amend this, the angle where they could move directly upwards and downwards was limited by 80°.

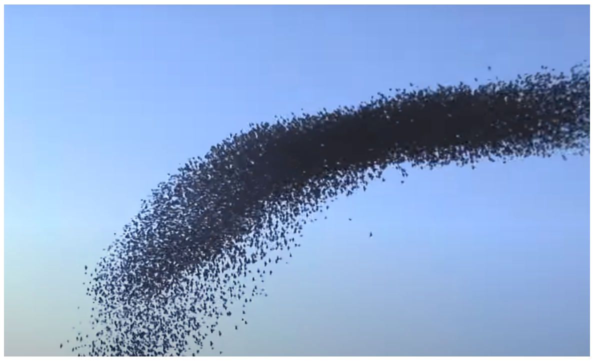
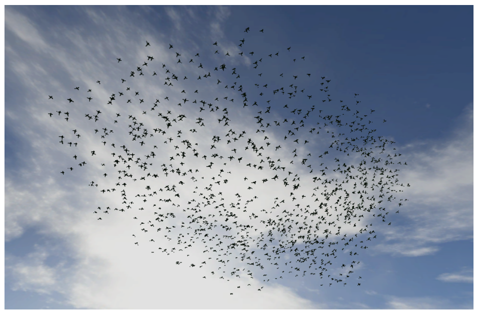
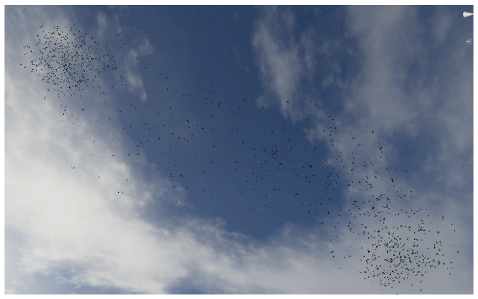
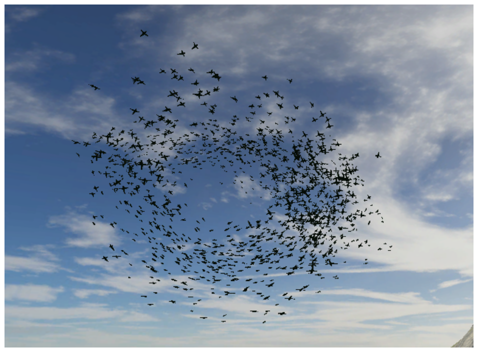
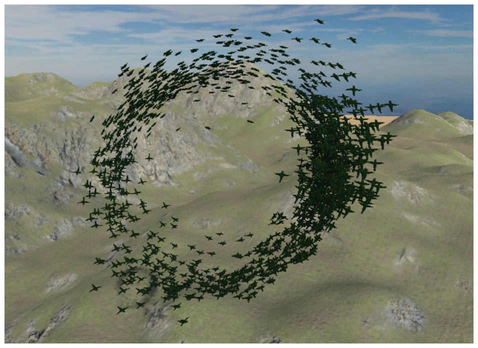


Figure 5.4: Image of a starling murmuration without a predator [38]


(a) Murmuration with the same weights for separation, cohesion, alignment and focal point, the starlings are separated and not moving together (1.5, 1.5, 1.5, 1.5)


(b) The Cohesion weight has been doubled, groups of starlings split off and fly too far from the focal point (1.5, 3, 1.5, 1.5)


(c) The focal point weight is doubled, the starlings rotate around the focal point but are still too far separated (1.5, 3, 1.5, 3)


(d) The focal point is increased again, the starling rotate around the focal point together in a circle (1.5, 3, 1.5, 5)

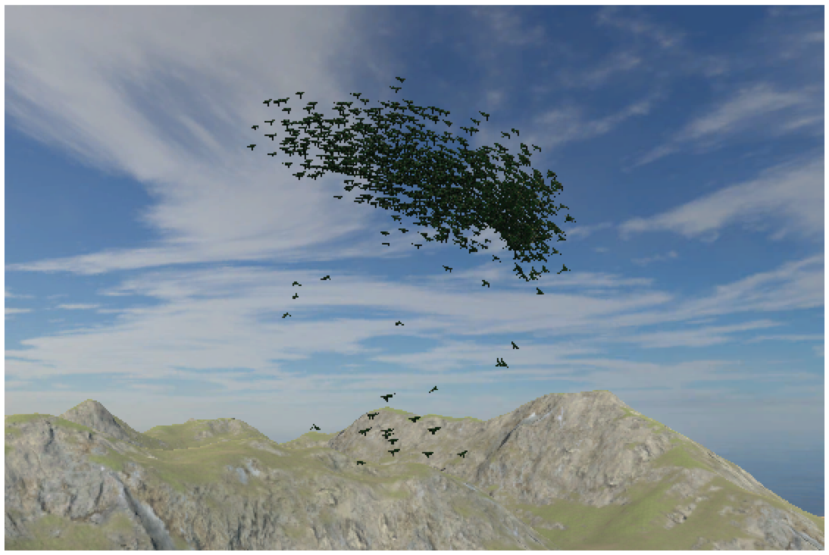

(e) Alignment is increased and the murmuration is flying together as one organism around the focal point (1.5, 3, 2.5, 5)

Figure 5.5: Images of different weights applied to a murmuration of 1,000 starlings, weights are separation, cohesion, alignment and focal point, each shown inside brackets in the description of each image with their weights respectively.

5.3 The Peregrine Falcon

The peregrine falcon was designed to work in three stages: move above the murmuration, glide for some time and stoop on a starling, and repeat these stages in an infinite loop to create patterns of interaction between the murmuration and Peregrine falcon. Each of the stages was implemented so every frame, the program will check which stage the falcon by a check with an enumeration containing the falcon stages, depending on the stage, the program will run appropriate actions to complete the stage so it can move onto the next. The specific stages are outlined below.

5.3.1 Move above the murmuration

In this stage, the peregrine falcon has a 3D vector which is above the murmuration and the falcon simply moves towards this vector with its animation of the wings flapping. Once the falcon is within a close distance to the vector, it will switch stages to the glide stage.

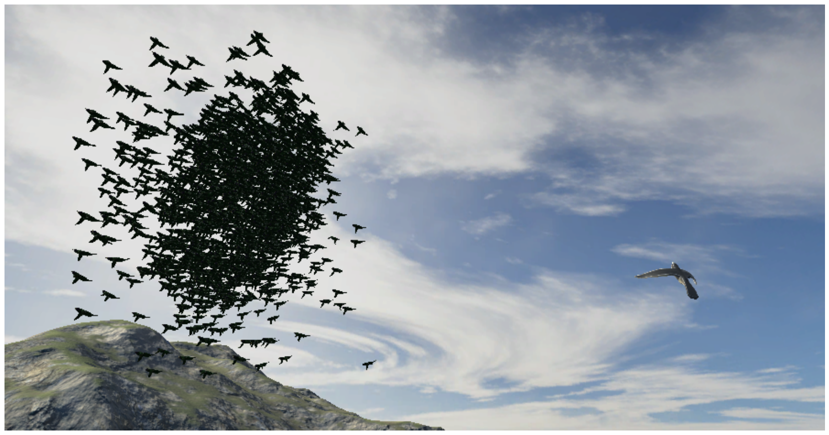


Figure 5.6: Falcon moving above the murmuration

5.3.2 Glide

In the glide stage, the falcon has no wing flapping animation as when birds glide, their wings are extended. To make sure the falcon does not wonder too far from the murmuration whilst gliding and looking for prey, its glide movement is implemented so it glides in a horizontal circle above the murmuration, the circle movement was implemented by moving the falcon towards a vector on a circle's perimeter whilst the vector moves around the circle's perimeter every tick to guide the falcon around the circle. After 5 seconds of gliding, the stage is changed to stooping.

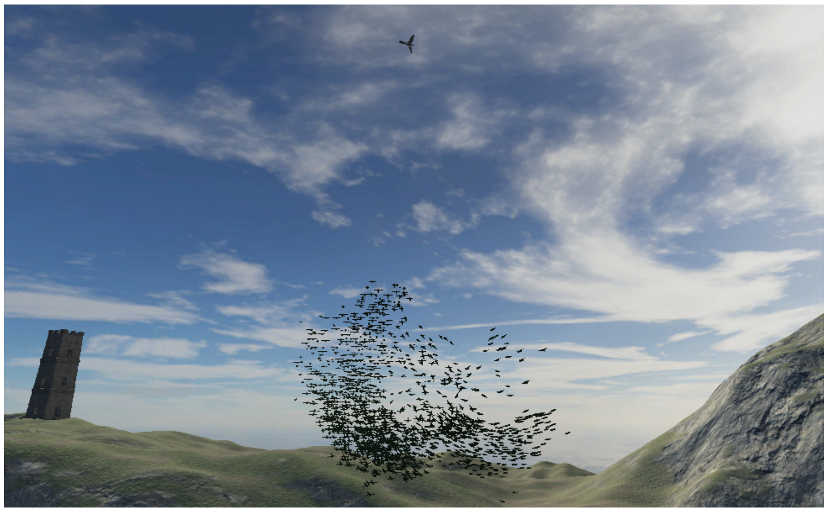
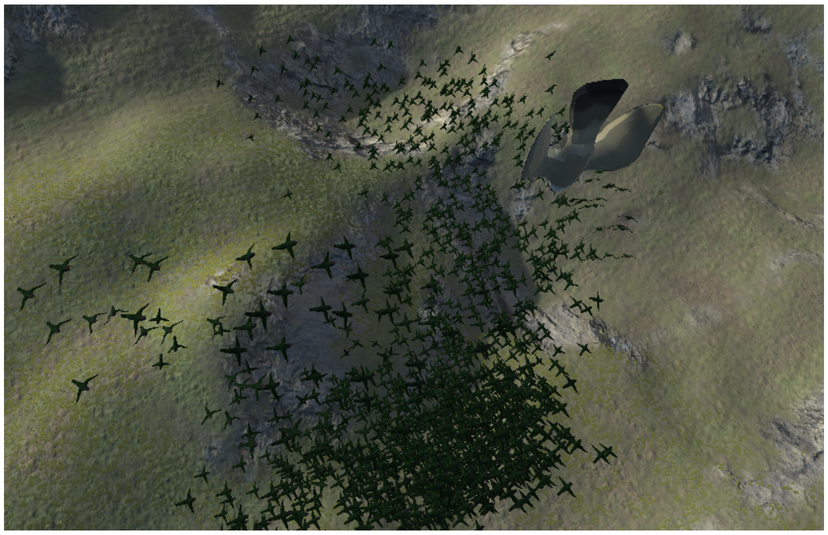


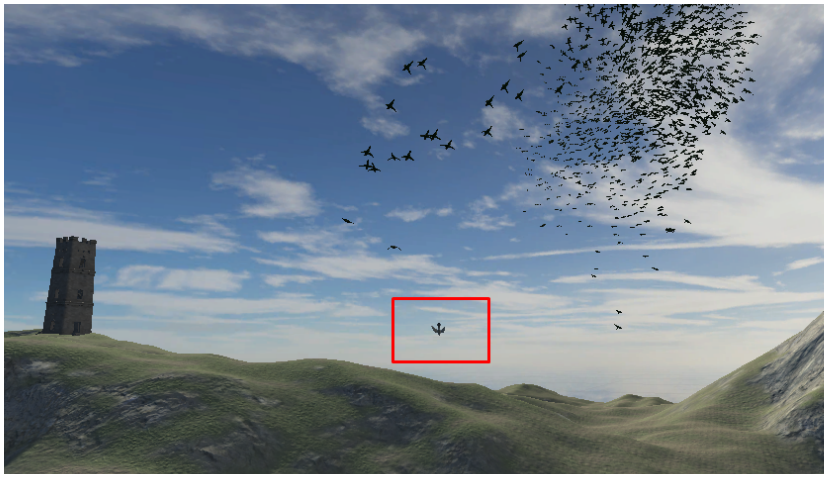
Figure 5.7: Falcon gliding above the murmuration

5.3.3 Stoop

When stooping, the program selects a random starling for the falcon to stoop on, the falcon accelerates with a high acceleration towards the starling with the animation of its wings folded in, once the falcon reaches the starling, it will simply pass through the starling as the functionality of catching a starling was not implemented due to time constraints. After the falcon has passed the starling, it will continue stooping for 5 metres (a suitable distance to slow down from its high velocity stoop) before switching stage to move above the murmuration again.



(a) Start of the stoop

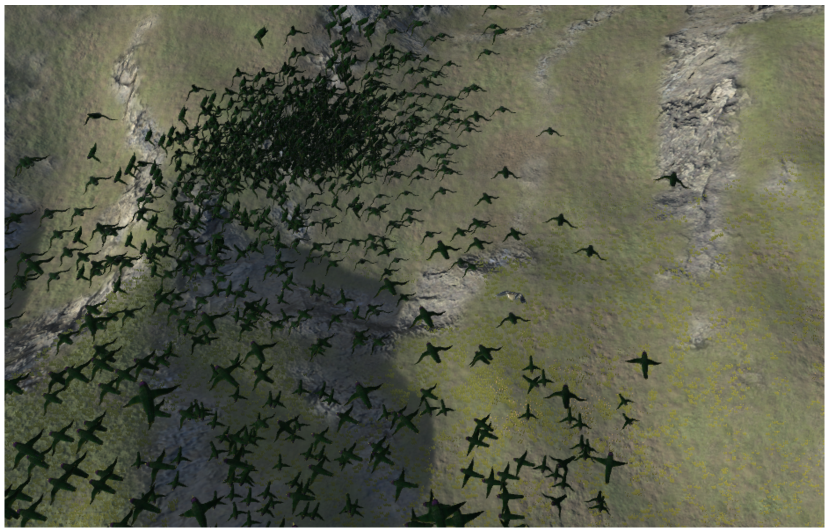


(b) End of the stoop, showing the falcon inside the red rectangle

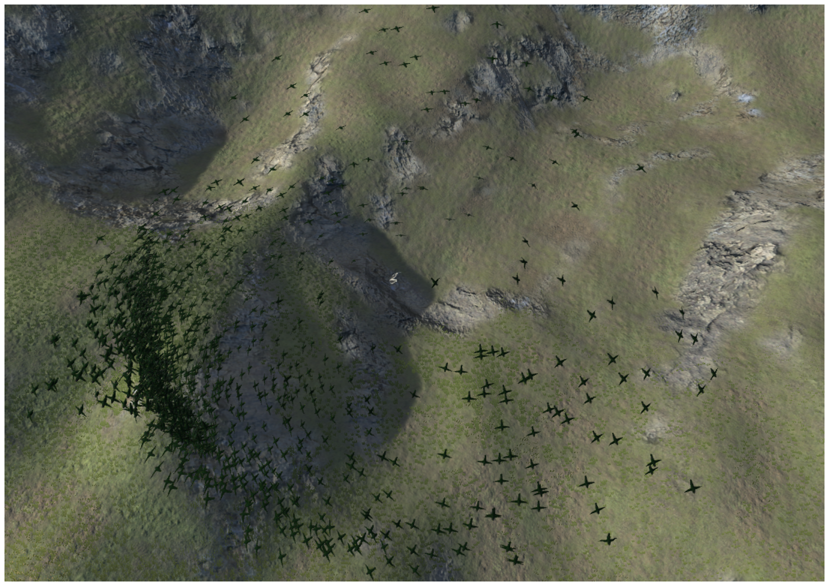
Figure 5.8: Images to show the stooping stage of the peregrine falcon

5.4 Murmuration-Predator Interaction

Once the predator was implemented, the predator repulsion force was added to the starlings, this force was simply a normalised vector in the opposite direction to the falcon from the starling. A distance of 6 metres was used for the starlings to detect the predator as from video analysis the predator is approximately 6 metres away for any starlings to react. To find a suitable weight for the predator repulsion force, in order to replicate an explosion effect found in real videos of murmurations with a predator, some trial and error was used testing different weights, Figure 5.9 shows examples of the difference between a low and high weight used for the repulsion force, the higher weight force creates the properties from real interactions as it recreates an explosion of the starlings separating as well as causing the main murmuration to have an area of higher density as the predator is closer.



(a) A low predator repulsion weight, there is no explosive effect



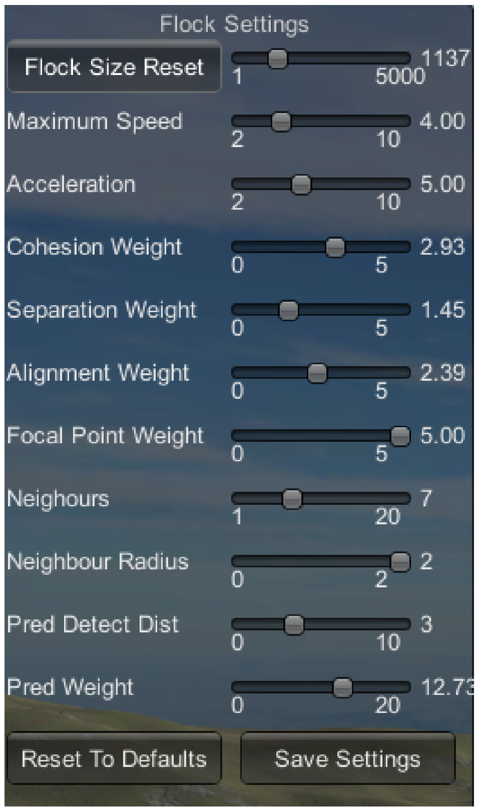
(b) A high predator repulsion weight, there is an explosion effect and some starlings are separated from the main murmuration

Figure 5.9: Images demonstrating different predator repulsion weights

5.5 GUI & Controls

In order to modify weights and parameters of the simulation easier and inspect the simulation with the use of pausing, a graphic interface and the options are shown in Figure 5.10 and each section is accessible by buttons in the top left of the screen while running the simulation. Most importantly, there is a run metrics button which allows the metrics to be calculated at any time the simulation is running as well as the ability to reset the metrics so you can modify the weights and check the metrics as the simulation runs.

Controls for moving the camera around the simulation were included as this allows any potential users of the simulation such as animal conversationalists to run and inspect the patterns and shapes produced by the murmuration with the predator.

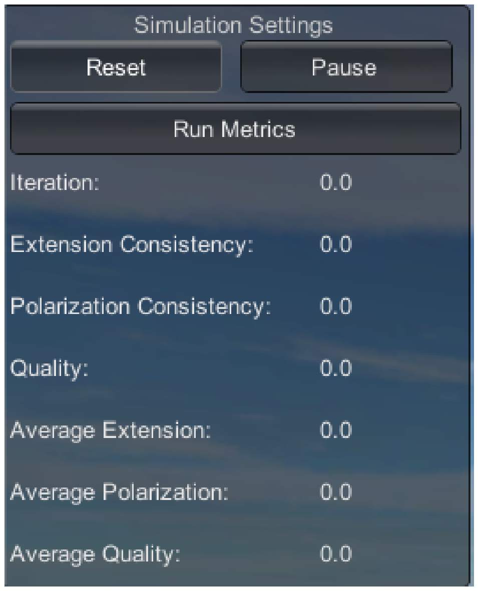


Flock Settings

Flock Size Reset	1	5000	1137
Maximum Speed	2	10	4.00
Acceleration	2	10	5.00
Cohesion Weight	0	5	2.93
Separation Weight	0	5	1.45
Alignment Weight	0	5	2.39
Focal Point Weight	0	5	5.00
Neighours	1	20	7
Neighbour Radius	0	2	2
Pred Detect Dist	0	10	3
Pred Weight	0	20	12.73

Reset To Defaults | Save Settings

(a) Parameters associated with the flock



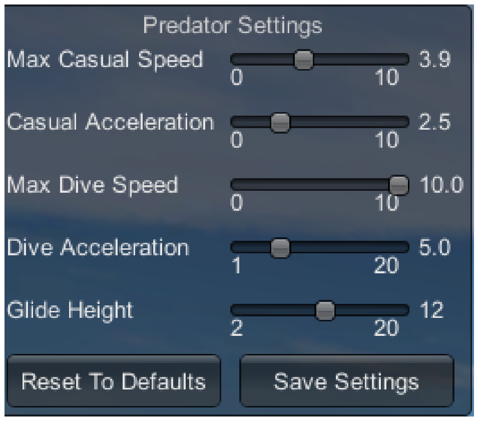
Simulation Settings

Reset | Pause

Run Metrics

Iteration:	0.0
Extension Consistency:	0.0
Polarization Consistency:	0.0
Quality:	0.0
Average Extension:	0.0
Average Polarization:	0.0
Average Quality:	0.0

(b) Overall simulation settings and metric results

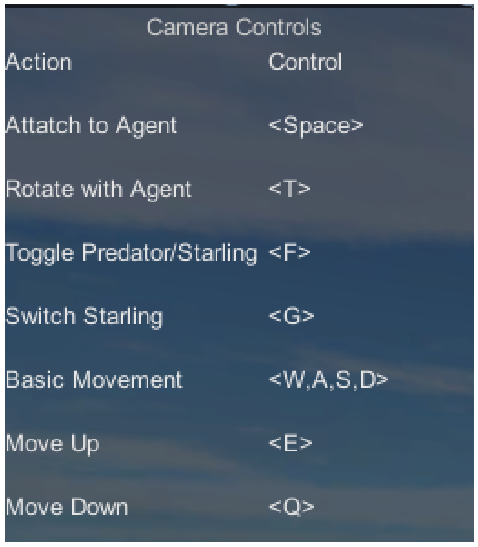


Predator Settings

Max Casual Speed	0	10	3.9
Casual Acceleration	0	10	2.5
Max Dive Speed	0	10	10.0
Dive Acceleration	1	20	5.0
Glide Height	2	20	12

Reset To Defaults | Save Settings

(c) Parameters associated with the predator



Camera Controls

Action	Control
Attatch to Agent	<Space>
Rotate with Agent	<T>
Toggle Predator/Starling	<F>
Switch Starling	<G>
Basic Movement	<W,A,S,D>
Move Up	<E>
Move Down	<Q>

(d) Camera Controls

Figure 5.10: Images of the GUI

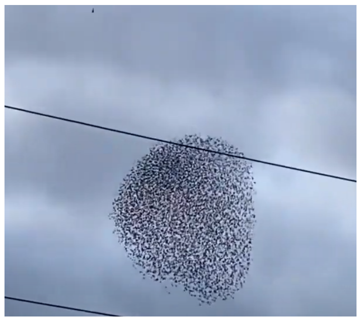
Chapter 6

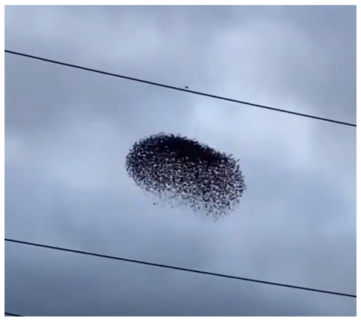
Results and Discussion

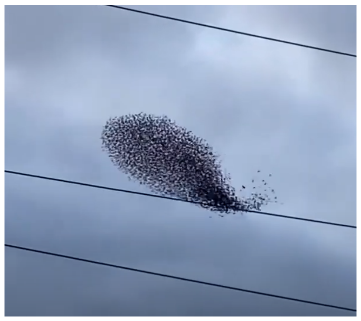
Evaluating the murmuration and peregrine falcon model will use qualitative and quantitative analysis. The qualitative analysis will involve analysing and comparing features and behaviours of images of the simulated model with images of real interactions between a predator and a murmuration. The quantitative analysis will involve metrics of extension and polarisation discussed in the literature survey [7], the consistency of extension measures how close every starling is from the average position of all starlings with a penalty for collisions, and the consistency of polarisation measures the consistency of the direction of every starling which also has a penalty for collisions. Together, the two metrics are combined to give a quality metric which tells you how well the murmuration is performing. Different weights for separation, alignment, cohesion, and the focal point are used for calculating the metrics to find out which weights produce a higher quality metric.

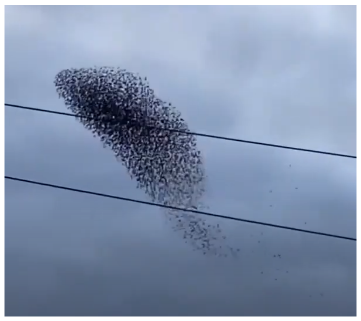
6.1 Qualitative Analysis Results

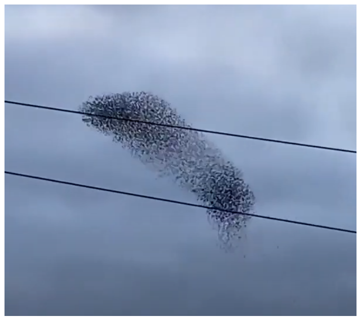
The qualitative analysis results section shows 8 images for a real murmuration with a predator and 8 images for the simulated one, each image is one second apart from the next so in total the images span 8 seconds of predator-murmuration interaction. The first image is from when the predator starts to stoop down onto the murmuration.

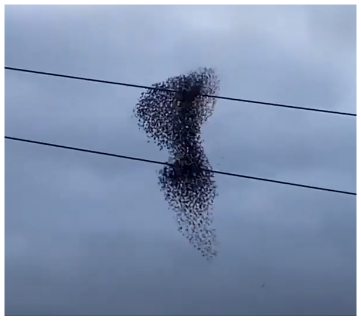

(a)

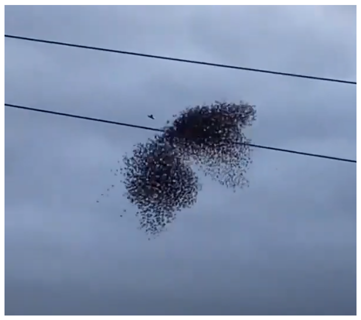

(b)


(c)


(d)


(e)


(f)


(g)

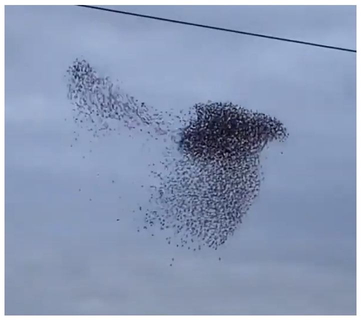

(h)

Figure 6.1: A series of images of a real starling murmuration interacting with a predator with 1 second between each image taken from a video [10]

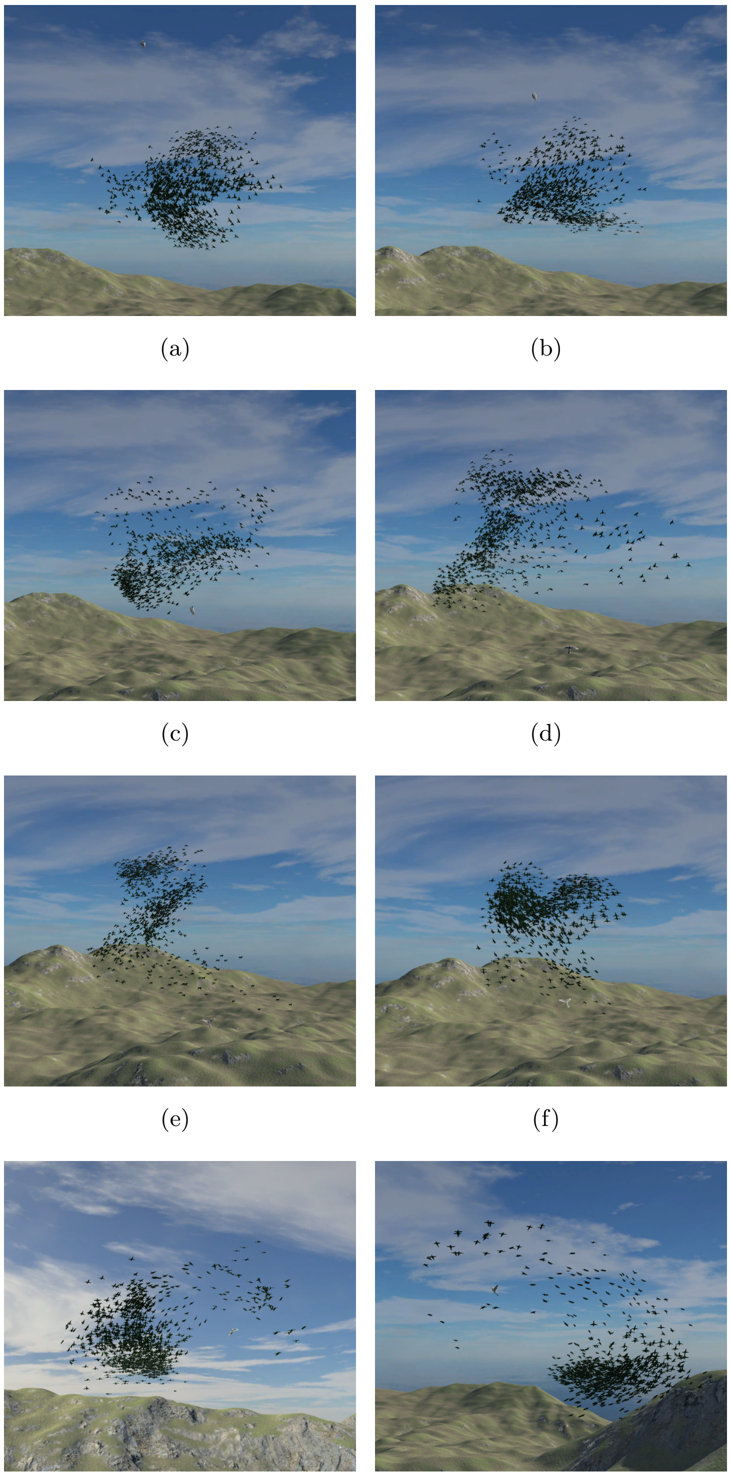

(a) (b) (c) (d) (e) (f) (g) Rotated 180° (h)

Figure 6.2: A series of images of the simulated starling murmuration interacting with a predator with 1 second between each image

6.2 Quantitative Analysis Results

The qualitative analysis results section uses different weights of separation, alignment, cohesion and focal point to experiment which combination of weights gives the highest quality metric (the combination of the average consistency of extension and the average consistency of polarisation).

Separation	Alignment	Cohesion	Focal	Avg Ext	Avg Pol	Avg Quality
2.00	2.00	2.00	2.00	0.5078	0.7503	0.6290
2.00	2.00	4.00	2.00	0.6801	0.7372	0.7086
2.00	2.00	1.00	2.00	0.5058	0.7508	0.6283
2.00	4.00	2.00	2.00	0.5430	0.7344	0.6387
2.00	1.00	2.00	2.00	0.5542	0.7292	0.6417
4.00	2.00	2.00	2.00	0.5441	0.7480	0.6460
1.00	2.00	2.00	2.00	0.5991	0.7390	0.6691
2.00	2.00	2.00	4.00	0.4430	0.6935	0.5683
2.00	2.00	2.00	1.00	0.5791	0.7269	0.6530

Table 6.1: The metrics of average consistency of extension and polarisation and quality caused from different weights for separation, alignment, cohesion and focal point taken over 1,000 frames. Begins with the same weights for each and then each individual weight is doubled and halved showing their metric values.

Separation	Alignment	Cohesion	Focal	Avg Ext	Avg Pol	Avg Quality
2.00	2.00	4.00	1.00	0.7290	0.7380	0.7335
1.00	2.00	4.00	1.00	0.7782	0.7409	0.7595

Table 6.2: The metrics of average consistency of extension and polarisation and quality caused from combining the best weights in table 6.1 to achieve a greater quality metric, each taken over 1,000 frames.

6.3 Discussion

6.3.1 Qualitative

In the qualitative analysis results section, you can see Figures 6.1 and 6.2 have some properties of similarity as the predator stoops down on the murmuration. For example, comparing Figures 6.1(a) and 6.2(a) both murmurations begin together in an uneven circular shape, although the distribution of starlings in 6.1(a) is more even compared with 6.2(a). As the predator approaches the murmuration in 6.1(b) and 6.2(b), both murmurations are more vertically compact, however in the real murmuration there is a higher density of starlings on the top compared with the simulated version, this could be because the predator detection distance is not large enough. Figures 6.1(c) and 6.2(c) show the predator at the same height as the murmuration, the predator in the real scenario has stooped to the right of the murmuration, whereas in the simulated scenario the predator has stooped directly in the centre. As the predator nears the murmuration there is a larger density of starlings on the side closest to the predator in the real scenario which isn't comparable with 6.2(c), however as the predator returns above the murmuration like in Figure 6.2(g) where the predator is on the same height as the murmuration again, you can see there is a larger density of starlings on the side where the predator is. Likewise with the real scenario in Figure 6.1(h), the simulation the predator in the simulation also causes starlings to separate from the murmuration, this is shown through the interaction of the predator in Figure 6.2 and specifically in Figures 6.2(g) and 6.2(h) where the predator has caused a larger number of starlings to become separated from the murmuration. Figures 6.2(f) and 6.1(g) show a similar shape caused by the predator of a murmuration with a curved gap coming into the centre of the murmuration. However, the general outline of the shape of the real murmuration is very smooth unlike the simulated scenario, this could be due to the different numbers of starlings, as in the simulation there are 1,000 starling and the real murmuration looks to be approximately 5,000 starlings and taken from a further distance away.

6.3.2 Quantitative

In the quantitative analysis results section, you can find tables of average quality of the murmuration with different weights. From these tables, conclusions can be made about which values of weights affect the quality of the murmuration in regard to the consistency of extension and polarisation. Having a larger cohesion weight improves the quality the most compared with other weights, while having a smaller cohesion weight decreases the quality the most, this means the cohesion weight is the most important weight to adjust to modify the murmuration. Following this, a lower separation weight is proven to also increase the quality of the murmuration, this is expected as a lowing the separation weight is very similar to increasing the cohesion weight as this will bring the starlings closer together therefore, increasing the average consistency of extension. Table 6.1 also shows when there is a smaller weight for the focal point, the quality of the murmuration is increased, this is unusual as a lower focal point weight caused the murmuration to often split and move in different direction as the other weights took precedence. On the other hand, increasing the focal point weight caused the average consistency of polarisation to be at its lowest, this is expected as when there is a high focal point weight the murmuration appears to circle around the focal point smaller therefore the directions of the starlings are not as aligned if their movement is in a circle. The weights which were found to give the highest quality metric were a separation of 1, alignment of 2, cohesion of 4 and focal point of 1. Overall, the average quality is above 0.6 which is satisfactory enough to conclude that the metrics prove the murmuration is successful.

6.4 Further Work

This projects' work on the simulation of starlings and the interaction with a predator is being continued by a master's student at the University of Sheffield, they aim to apply this research into herding swarms of birds using autonomous drones or unmanned vehicles (UAVs). The new project will replace the predator with one or more drones, assuming the starlings reaction to a drone is similar to how they react to a predator. The project is motivated by issues caused by bird droppings which can create hazards in cities, in particular the centre of Rome [24]. The project aims to explore if using drones can move a murmuration of starlings to another location away from the centre of Rome in a simulation.

If there was more time to complete this work, additional features could be added to improve realism of the project as a whole and to improve the movement of the starlings to make the murmuration more natural. StarDisplay's model [11] used more advanced physics such as simulating all the individual forces on each starling like gravity, drag and thrust, whereas this work's model just had an acceleration in the direction of the overall force calculated using separation, alignment, cohesion, focal point and the predator, and stopped accelerating once it had reached a maximum speed. Applying realistic physics to the model, will show more natural movement such as a banking angle to control the starlings turning rate and may result in a more natural murmuration. To add more realism into the project, the predators behaviour could be adjusted to physically catch the starling with it's feet and take it away or remove the starling from the murmuration, moreover, the predators movement to return to the top of the murmuration could be improved, currently the predator will take a straight path to the top of the murmuration however, in a real scenario the predator would take a route around the murmuration to return to the top to stoop down again. Finally, an additional random force to move the starlings in different directions was not explored in this project, as real murmurations will have some randomness which is caused by factors such as wind.

6.5 Summary

Results have been analysed both qualitatively and quantitatively, revealing this work does exhibit patterns and shapes from real murmurations with a predator and the positions and rotations of the starlings are generally cohesive as proved by the metrics of extension and polarisation. There are still some complex shapes and movement of the murmuration which does not replicate its real-life counterpart, however, the main aims for this project have been completed and the work is successful.

Chapter 7

Conclusions

This project aimed to study the murmurations interaction with a predator and develop a realistic simulation to re-create the patterns and complex shapes which arise from the interactions, this aim was successfully achieved. The simulation was created in Unity, with accurate models of the birds created in Blender and programmed in C. Simulating the murmuration was largely based on three rules inspired from Reynolds Boid algorithm [**?**], which are separation, alignment and cohesion, applying these rules along with two additional rules which are an attraction toward a focal point or nesting area and a repulsion force away from a predator, is all that is required to guide each individual starling in a murmuration. The predator's behaviour in the simulation mostly reassembled its behaviour in real-life, in the simulation, the predator glides above the simulation, stoops down on a starling and then returns above the murmuration and repeats, similar to its behaviour in real-life. The simulation exhibits a range of features which recreates a real life environment, it contains a hill landscape with grass and a tower to add to the ambience as well as a cloudy blue sky. The simulation was evaluated using two methods, quantitative and qualitative analysis, in the qualitative analysis it illustrates several images to compare the shape, density and motion between a real life scenario of a murmuration interacting with a predator and the simulation. The quantitative analysis used metrics of the consistency of extension and the consistency of polarisation [7] to evaluate the directions and positions of the starlings and provided useful feedback in fine-tuning weights for separation, cohesion and alignment.

The results from the quantitative and qualitative analysis reveals the work does replicate most of the patterns and shapes from real murmurations with a predator, as well as metrics which reveal the positions and directions of the starlings are generally cohesive, however there are still features of real murmurations which were not replicated such as the general outline of the shape of real murmurations are very smooth compared to this project's simulation. This could be due to the movement of the starlings, in the simulation they simply accelerate towards the vector obtained by the forces of separation, alignment and cohesion, and will stop accelerating in a particular direction if they have reached a maximum speed. Alternatively, realistic physics could be implemented here such as the forces of gravity, drag, upthrust and banking angles when the starlings turn as this may result in a greater replication of the interactions found in real life. If this were to be improved, it would be easier to compare as the metrics algorithm is already implemented.

One of the main problems to overcome in this project was removing the complexity of the O(n^2) algorithm caused by finding the closest neighbours of every starling, as this would have to run through every starling and compare every other starling's distance to it to obtain the closest neighbours to performance the forces of separation, alignment and cohesion on. With the complexity of O(n^2), only a maximum of 200 starlings could be on the simulation at once, however with the use of a uniform spatial grid which keeps track of where each starling is in a grid structure, it turned the O(n^2) algorithm into O(Kn), a linear complexity. In addition to this, the use of multi-threading allowed 2,000 starlings to be in the simulation without causing any performance issues.

Overall, the aims of this project were met, and the simulation is visually pleasing with accurate models of the starling and peregrine falcon, along with animations to supplement their movements and the overall realism of the simulation. Although addition improvements could be made, the initial aims for the project have been achieved and the work is considered a success.

Bibliography

[1] A. Procaccini, A. Orlandi, A. C. I. G. F. Z. D. S. F. C. C. K. H. E. A. G. P. e. a. Propagating waves in starling, sturnus vulgaris, flocks under predation, 2011.

[2] AgingEyes. European starling in flight, http://forums.photographyreview.com/nature-wildlife/european-starling-flight-54966.html, 2009.

[3] Anne E. Goodenough, Natasha Little, W. S. C., and Hart, A. G. Birds of a feather flock together: Insights into starling murmuration behaviour revealed using citizen science, 2017.

[4] B. G. Hogan, H. Hildenbrandt, N. E. S.-S. I. C. C., and Hemelrijk, C. K. The confusion effect when attacking simulated three-dimensional starling flocks, 2017.

[5] Bogdan. Ecs boids murmuration unity, https://github.com/bogdancodreanu/ecs-boids-murmuration_unity_2019.1, 2020.

[6] Branch, B. I. W. C. Starling murmuration & peregrine falcon attack https://www.youtube.com/watch?v=vhoaxvwvnic, 2018.

[7] C. Gershenson, A. M.-M., and Zapotecatl, J. L. Performance metrics of collective coordinated motion in flocks, 2016.

[8] Duffield, C., and Ioannou, C. C. Marginal predation: do encounter or confusion effects explain the targeting of prey group edges?, 2017.

[9] et al, S. T. Can parks protect migratory ungulates? the case of the serengeti wildebeest, 2004.

[10] Francisco, K. F. . S. Murmuration of starlings react as prey attacks in marin county, https://www.youtube.com/watch?v=j8qhmh52ypa, 2021.

[11] H. Hildenbrandt, C. Carere, C. H. Self-organized aerial displays of thousands of starlings: a model, 2010.

[12] Hemelrijk, C. K., and Hildenbrandt, H. Diffusion and topological neighbours in flocks of starlings: relating a model to empirical data, 2015.

[13] Hildenbrandt, C. K. H. H. Self-organized shape and frontal density of fish schools, 2008.

[14] Jernel, A. Starlings in north america, 2017.

[15] K. M. Tanimu, W. P., and Coghill, G. M. A conceptual framework of starlings swarm intelligence intrusion prevention for software defined networks, 2018.

[16] King, A. J., and Sumpter, D. J. Murmurations, 2012.

[17] L. V. Riters, C. A. K.-N., and Spool, J. A. Why do birds flock? a role for opioids in the reinforcement of gregarious social interactions, 2019.

[18] links open overlay panelPeterde Groot, A. Information transfer in a socially roosting weaver bird (quelea quelea; ploceinae): an experimental study, 1980.

[19] M, M. P. Uniform spatial subdivision to improve boids algorithm in a gaming environment, 2018.

[20] M. Ballerini, N. Cabibbo, R. C.-A. C.-E. C. I. G. V. L. A. O. G. P. A. P. M. V., and Zdravkovic, V. Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study, 2008.

[21] Mills, R. A peregrine falcon in a stoop, https://physicsworld.com/a/falcons-high-speed-dive-generates-forces-needed-to-catch-agile-prey/, 2018.

[22] Mohammad Pirani, Hassan Basirat Tabrizi, A. F. Study of swarm behavior in modeling and simulation of cluster formation in nanofluids, 2013.

[23] Moore, L. Nature education programme - starling murmuration https://www.youtube.com/watch?v=qk_rcpp1uvg, 2019.

[24] O'Sullivan, F. Rome is getting buried in the droppings of a million starlings, 2016.

[25] Pearce DJ, Miller AM, R. G., and MS, T. Role of projection in the control of bird flocks, 2014.

[26] Public Television's Wild Chronicles, f. N. G. M. P. Terminal velocity: Skydivers chase the peregrine falcon's speed, 27 January 2012.

[27] R. F. Storms, C. Carere, F. Z., and Hemelrijk, C. K. Complex patterns of collective escape in starling flocks under predation, 2019.

[28] R. Mills, G. K. T., and Hemelrijk, C. K. Sexual size dimorphism, prey morphology and catch success in relation to flight mechanics in the peregrine falcon: a simulation study, 2019.

[29] R. Mills, H. Hildenbrandt, G. K. T., and Hemelrijk, C. K. Physics-based simulations of aerial attacks by peregrine falcons reveal that stooping at high speed maximizes catch success against agile prey, 2018.

[30] Reynolds, C. Boids, 1986.

[31] Robin Mills, Hanno Hildenbrandt, G. K. T., and Hemelrijk, C. K. Physics-based simulations of aerial attacks by peregrine falcons reveal that stooping at high speed maximizes catch success against agile prey, 2018.

[32] Skooter500. Ecs boids, https://github.com/skooter500/ecsboids, 2019.

[33] Society, N. A. The hummingbird wing beat challenge, 2020.

[34] twootz.com. Starling facts - information about starling, https://twootz.com/bird/starling, 2021.

[35] Unity. Entity component system, https://docs.unity3d.com/packages/com.unity.entities@0.17/manual/index.html, 2017.

[36] Wikipedia. A peregrine falcon, 2020.

[37] Wikipedia. Starling, 2020.

[38] Winter, D. Video of a starling murmuration, 2010.

[39] Yu-Hao, C. Modelling a murmuration of starlings interacts with a predator, 2020.

Appendices

Appendix A

Flock code

```
public struct MyJob : IJobParallelFor
{
    public NativeArray<Vector3> newVelocities;
    public NativeArray<bool> escaping;
    [ReadOnly]
    public NativeArray<Vector3> agentPositions;
    [ReadOnly]
    public NativeArray<Vector3> agentVelocities;
    [ReadOnly]
    public NativeMultiHashMap<int, int> boidCellNeighbours;

    public void Execute(int index)
    {
        List<int> context = GetNearbyStarlings(index);
        newVelocities[index] = CalculateBoidForce(index, context);
    }

    public float focalPointWeight;
    public float cohesionWeight;
    public float separationWeight;
    public float alignmentWeight;
    public float predatorWeight;
    private Vector3 CalculateBoidForce(int index, List<int> context)
    {
        Vector3 move = FocalPoint(index) * focalPointWeight;
        if (context.Count > 0)
        {
            Vector3 cohesion = Cohesion(index, context) * cohesionWeight;
            Vector3 separation = Separation(index, context) * separationWeight;
            Vector3 alignment = Alignment(index, context) * alignmentWeight;
            move += separation + cohesion + alignment;
        }
        Vector3 predator = Predator(index) * predatorWeight;
        if (predator != Vector3.zero)
        {
            move += predator;
            escaping[index] = true;
        }
        else
        {
            escaping[index] = false;
        }

        return move.normalized;
    }

    public Vector3 Separation(int index, List<int> context)
    {
        //Separation rule
        //steer away from neighbours
        Vector3 separationMove = Vector3.zero;
        foreach (int otherStarlingIndex in context)
        {
            float dist = Vector3.Distance(agentPositions[otherStarlingIndex], agentPositions[index]);
            float invserseDist = 1 / dist;
            Vector3 repulsion = (agentPositions[otherStarlingIndex] - agentPositions[index]) * invsers
            separationMove += repulsion;
        }

        separationMove = separationMove.normalized;

        return separationMove;
    }

    public Vector3 Cohesion(int index, List<int> context)
    {
        //Cohesion rule
        //move towards the average position of neighbours
        Vector3 averagePos = Vector3.zero;
        foreach (int otherStarlingIndex in context)
        {
            averagePos += agentPositions[otherStarlingIndex];
        }

        Vector3 cohesionMove = averagePos - agentPositions[index];

        cohesionMove = cohesionMove.normalized;
        return cohesionMove;
    }

    public Vector3 Alignment(int index, List<int> context)
    {
        //Alignment rule
        //get in line with the average direction of neighbours
        Vector3 alignment = Vector3.zero;
        foreach (int otherStarlingIndex in context)
        {
            alignment += agentVelocities[otherStarlingIndex];
        }

        alignment = alignment.normalized;

        return alignment;
    }

    public Vector3 focalPoint;
    public Vector3 FocalPoint(int index)
    {
        //move towards focal point
        float dist = Vector3.Distance(focalPoint, agentPositions[index]);
        float invserseDist = 1 / dist;
        Vector3 attraction = (focalPoint - agentPositions[index]) * invserseDist;
        attraction = attraction.normalized;
        return attraction;
    }

    public float predatorDetectionDist;
    public Vector3 predatorPosition;
    public Vector3 Predator(int index)
    {
        //move away from predator if within detection distance
        float dist = Vector3.Distance(predatorPosition, agentPositions[index]);
        if (dist < predatorDetectionDist)
        {
            float invserseDist = 1 / dist;
            Vector3 repulsion = (predatorPosition - agentPositions[index]) * invserseDist * -1;
            repulsion = repulsion.normalized;
            return repulsion;
        }
        else
        {
            return Vector3.zero;
        }
    }
    public float neighborRadius;
    public int neighbourToConsider;
    public List<int> GetNearbyStarlings(int index)
    {
        var closeAgents = new Dictionary<int, float>();

        List<int> context = new List<int>();
        foreach (int agentIndex in boidCellNeighbours.GetValuesForKey(index))
        {
            if (agentIndex == index) continue; //Ignore itself
            float distance = Vector3.Distance(agentPositions[index], agentPositions[agentIndex]);
            if (distance >= neighborRadius) continue;
            if (distance <= hardSphere) continue; //avoid if in hard sphere

            bool added = false;
            foreach (KeyValuePair<int, float> pair in closeAgents)
            {
                if (pair.Value > distance)
                {
                    closeAgents.Remove(pair.Key);
                    closeAgents.Add(agentIndex, distance);
                    added = true;
                    break;
                }
            }
            if (!added && closeAgents.Count < neighbourToConsider)
            {
                closeAgents.Add(agentIndex, distance);
            }
            if (closeAgents.Count == neighbourToConsider) break;
        }

        foreach (KeyValuePair<int, float> pair in closeAgents)
        {
            context.Add(pair.Key);
        }

        return context;
    }
}



public Transform GetRandomStarling()
{
    return agents[Random.Range(0, agents.Count - 1)].transform;
```

```
}

public void DeleteFlock()
{
    foreach (FlockAgent agent in agents)
    {
        Destroy(agent.gameObject);
    }
    agents.Clear();
}

public GameObject FindPredatorTarget()
{
    return agents[Random.Range(0, agents.Count - 1)].gameObject;
}

public List<FlockAgent> GetNearbyStarlings(FlockAgent agent)
{
    List<FlockAgent> cellBoids = new List<FlockAgent>();
    List<int> boidIndexes = grid.GetNeighbours(int.Parse(agent.gameObject.name));

    foreach (int i in boidIndexes) {
        cellBoids.Add(agents[i]);
    }

    var closeAgents = new Dictionary<FlockAgent, float>();

    List<FlockAgent> context = new List<FlockAgent>();
    foreach (FlockAgent a in cellBoids)
    {
        if (agent.name == a.name) continue; //Ignore itself
        float distance = Vector3.Distance(agent.transform.position, a.transform.position);
        if (distance >= neighborRadius) continue;
        Vector3 targetDir = a.transform.position - agent.transform.position;
        float angle = Vector3.Angle(targetDir, agent.transform.forward);
        if (angle >= blindAngle) continue; //avoid blind angle
        if (distance <= hardSphere) continue; //avoid if in hard sphere

        bool added = false;
        foreach (KeyValuePair<FlockAgent, float> pair in closeAgents)
        {
            if (pair.Value > distance)
            {
                closeAgents.Remove(pair.Key);
                closeAgents.Add(a, distance);
                added = true;
                break;
            }
        }
        if (!added && closeAgents.Count < neighbourToConsider)
        {
            closeAgents.Add(a, distance);
        }
        if (closeAgents.Count == neighbourToConsider) break;
    }

    foreach (KeyValuePair<FlockAgent, float> pair in closeAgents)
    {
        context.Add(pair.Key);
    }

    return context;
}
```

Appendix B

Predator code

```
public class PredatorAgent : MonoBehaviour
{
    public Vector3 velocity = Vector3.zero;

    private readonly int hoverTime = 5;
    private float angle = 0;
    private float HoverTimeCounter = 0;
    private bool stooping = false;
    private GameObject target;
    private Animator falconAnimator;

    public float hoverSpeed = 0.3f;
    public float hoverTurnSpeed = 1.5f;
    public float hoverCircleSpeed = 0.5f;
    public float hoverCircleRadius = 1;

    private Flock flock;

    private enum PredState
    {
        MoveAboveMurmuration,
        Hover,
        Stoop,
    }
    private PredState predState = PredState.MoveAboveMurmuration;

    private void Start()
    {
        falconAnimator = this.GetComponentInChildren<Animator>();
    }

    public void Initialize(Flock flock)
```

```
{
    this.flock = flock;
}

void Update()
{
    if (Menu.runSimulation)
    {
        switch (predState)
        {
            case PredState.MoveAboveMurmuration:
                if (Move())
                {
                    predState = PredState.Hover;
                    falconAnimator.SetInteger("flightType", 1);
                }
                break;
            case PredState.Hover:
                if (Hover())
                {
                    predState = PredState.Stoop;
                    GameObject predTarget = flock.FindPredatorTarget();
                    InitStoop(predTarget);
                    falconAnimator.SetInteger("flightType", 2);
                }
                break;
            case PredState.Stoop:
                if (Stoop())
                {
                    predState = PredState.MoveAboveMurmuration;
                    falconAnimator.SetInteger("flightType", 0);
                }
                break;
        }
    }
}

private bool Move()
{
    Vector3 hoverPoint = new Vector3(0, Menu.predHoverHeight, 0) + flock.focalPoint;
    Vector3 targetDirection = hoverPoint - transform.position;
    targetDirection = targetDirection.normalized;

    Vector3 accelerationV = targetDirection * Menu.predMoveAcceleration;
    Vector3 newVelocity = velocity + accelerationV * Time.deltaTime;
    if (newVelocity.magnitude < Menu.predMoveSpeed)
    {
        velocity = newVelocity;
    }
    else
    {
        velocity = newVelocity.normalized * Menu.predMoveSpeed;
    }
    float limit = 0.8f * velocity.magnitude;
    if (velocity.y > limit) velocity.y = limit;
    if (velocity.y < -limit) velocity.y = -limit;
    transform.position += velocity * Time.deltaTime;
    transform.rotation = Quaternion.LookRotation(velocity);

    //return true if its within 1 unit of target position
    float dist = Vector3.Distance(hoverPoint, transform.position);
    if (dist < 3) return true;
    else return false;
}

private bool Hover()
{
    Vector3 hoverPoint = new Vector3(0, Menu.predHoverHeight, 0) + flock.focalPoint;
    angle += Time.deltaTime * hoverCircleSpeed;
    angle = angle % 360;
    float toRadians = angle * Mathf.PI / 180;
    float x = hoverCircleRadius * Mathf.Sin(toRadians);
    float y = hoverCircleRadius * Mathf.Cos(toRadians);
    Vector3 circlePoint = new Vector3(x, 0, y) + hoverPoint;

    Vector3 targetDirection = circlePoint - transform.position;
    targetDirection = targetDirection.normalized;
    float singleStep = hoverTurnSpeed * Time.deltaTime;
    Vector3 direction = Vector3.RotateTowards(transform.forward, targetDirection, singleStep, 0.0
    float limit = 0.2f;
    if (direction.y > limit) direction.y = limit;
    if (direction.y < -limit) direction.y = -limit;
    transform.rotation = Quaternion.LookRotation(direction);

    velocity = direction * hoverSpeed;
    transform.position += velocity;

    HoverTimeCounter += Time.deltaTime;
    if (HoverTimeCounter > hoverTime)
    {
        HoverTimeCounter = 0;
        return true;
    }
    return false;
}

private void InitStoop(GameObject target)
{
    this.target = target;
    keepGoing = false;
}

bool keepGoing = false;
private bool Stoop()
{
    Vector3 targetPos = target.transform.position;
    if (!keepGoing)
    {
        Vector3 targetDirection = (targetPos - transform.position).normalized;

        Vector3 accelerationV = targetDirection * Menu.predStoopAcceleration;
        Vector3 newVelocity = velocity + accelerationV * Time.deltaTime;
        if (newVelocity.magnitude < Menu.predStoopSpeed)
        {
            velocity = newVelocity;
        }
        else
        {
            velocity = newVelocity.normalized * Menu.predStoopSpeed;
        }
    }
    transform.position += velocity * Time.deltaTime;
    transform.rotation = Quaternion.LookRotation(velocity);

    if (transform.position.y < targetPos.y - 6)
    {
        stooping = false;
        return true;
    } else if (transform.position.y < targetPos.y)
    {
        keepGoing = true;
    }

    return false;
}
}
```

Appendix C

Grid code

```
public class Grid : MonoBehaviour
{

   public Dictionary<int, List<int>> cells = new Dictionary<int, List<int>>();
   public Dictionary<int, List<int>> getCells { get { return cells; } }
   public Dictionary<int, int> boids = new Dictionary<int, int>();
   public Dictionary<int, int> getBoids { get { return boids; } }
   public int cellSize = 50;
   public int gridSize = 10000;

   public static int PositionToCell3D(Vector3 pos, int cellSize, int gridSize)
   {
      return ((int)(pos.x / cellSize))
         + ((int)(pos.z / cellSize)) * gridSize
         + ((int)(pos.y / cellSize)) * gridSize * gridSize;
   }

   public static List<int> getSurroundingCells(int cell, int cellSize, int gridSize)
   {
      List<int> neighbourCells = new List<int>() { cell };
      int row = cell / gridSize;
      int col = cell - (row * gridSize);

      neighbourCells.Add(gridSize * (row + 1));
      neighbourCells.Add(gridSize * (row - 1));
      neighbourCells.Add(col + 1 + (row * gridSize));
      neighbourCells.Add(col - 1 + (row * gridSize));
      neighbourCells.Add(col + 1 + ((row + 1) * gridSize));
      neighbourCells.Add(col - 1 + ((row - 1) * gridSize));
      neighbourCells.Add(col + 1 + ((row - 1) * gridSize));
      neighbourCells.Add(col - 1 + ((row + 1) * gridSize));
```

```
        return neighbourCells;
    }

    public void UpdatePosition(int i, Vector3 pos)
    {
        int newCell = PositionToCell3D(pos, cellSize, gridSize);
        int currentCell = boids[i];
        if (newCell != currentCell)
        {
            cells[currentCell].Remove(i);
            AddToCells(newCell, i);
            boids[i] = newCell;
        }
    }

    public void Populate(int i, Vector3 pos)
    {
        int cell = PositionToCell3D(pos, cellSize, gridSize);
        AddToCells(cell, i);
        boids.Add(i, cell);
    }

    public void AddToCells(int cell, int i)
    {
        if (cells.ContainsKey(cell))
        {
            cells[cell].Add(i);
        }
        else
        {
            cells.Add(cell, new List<int>() { i });
        }
    }

    public void Clear()
    {
        cells.Clear();
        boids.Clear();
    }

    public List<int> GetNeighbours(int i)
    {
        List<int> neighbours = new List<int>();
        int thisCell = boids[i];
        List<int> neighbourCells = getSurroundingCells(thisCell, cellSize, gridSize);
        foreach (int cell in neighbourCells) {
            if (cells.ContainsKey(cell)) {
                neighbours.AddRange(cells[cell]);
            }
        }

        return neighbours;
    }
}
```