# Simulation Exercise for a System of Self-Driven Particles

PHYS-571 report

Alejandro van Breukelen García[a], Fatemeh Nouri[a], Pierre-Louis Wasmer[a]

Supervised by Professors Ferrero and Couairon

[a]*IP Paris, M1 Physics*

**Abstract**

In this report, we present our swarm simulator code, whose purpose is to study the phase transitions of systems of interacting boids under various initial conditions, in an attempt to replicate the model of Vicsek et al. 1995 [2]. In all of our simulations, the boids move at a constant speed, and update their directions of motion based on that of their neighbors, plus a random noise term. We report that our model produces organic-looking animations and produces systems that exhibit a phase transition around a noise level, $\eta$, of 4 radians. This phase transition is sharper for larger systems, indicating that our model can approximate case of the thermodynamic limit $N$ and $L \to \infty$. Lastly, we show that for large systems, a binning algorithm indeed performs notably better than its brute force counterpart.

Our code is publicly available at: `https://github.com/Avanbreukeleng/BirdSwarmInternationalGang`

## 1. Introduction

The study of systems of many interacting boids has captivated researchers on various fronts. In particular, the study of their behaviour during phase transition provides valuable insights into the dynamics of biological subjects such as schools of fish, herds of quadrupeds, or flocks of flying birds. In these systems, each member tends to move as other subjects do in their neighbourhood, resulting in a phase transition from an inital configuration of randomly moving boids to a net flow in a certain direction [1]. This emergent behaviour is not restricted to flocks, but it is in this context that we take a closer look at how individual particles with simple rules, follow well-defined dynamics.

In this project, we aimed to study the phase transition of self-driven particles by using the model introduced in Vicsek et al. [2]. The model is as follows: at each time step a given particle driven with a constant absolute velocity assumes the average direction of motion of the particles in its neighbourhood of radius $R$ with some added random perturbation. Our simulation was carried out in a square-shaped cell of size $L$ with periodic boundary conditions. At time $t = 0$, $N$ particles' initial position and direction are determined by uniform distribution and after each time step the position and direction are updated according to:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t)\Delta t, \tag{1.1}$$

$$\theta(t+1) = \langle\theta(t)\rangle_r + \Delta\theta, \tag{1.2}$$

Where $\langle\theta(t)\rangle_r = \arctan(\langle\sin(\theta(t))\rangle_r / \langle\cos(\theta(t))\rangle_r$ denotes the average direction of the velocities of particles, including the particle $i$, within a circle of radius $r$ surrounding the given particle and $\Delta\theta$ is the noise, chosen with a uniform probability from the interval $[-\eta/2, \eta/2]$. In order to study the transition properties as a function of density $\rho = N/L^2$ and noise $\eta$, we set $r = 1$, $\Delta t = 1$, and $\|\mathbf{v}_i\| = 0.033$.

We expect to observe the following situations: 1) For small densities and noise the particles tend to form groups moving coherently in random directions. 2) At higher densities and noise the particles move randomly with some correlation. 3) perhaps the most interesting case is when the density is large and the noise is small; in this case, the motion becomes ordered on a macroscopic scale and all of the particles tend to move in the same spontaneously selected direction.

As in any system with phase transition possibility, we need to introduce an order parameter. In our model, we used the absolute value of the average normalized velocity

$$v_a = \frac{1}{Nv}\left|\sum_{i=1}^{N}\mathbf{v}_i\right|. \tag{1.3}$$

This parameter quantifies the dispersion of the bird's velocities. In a fully decoupled system, all birds move in random directions, and the average normalized velocity is thus close to zero. On the other hand, a value of $v_a$ close to one indicates that the birds are moving as a unit, along the same direction. As such, we can think of $v_a$ as an order parameter.

In the thermodynamic limit, where $N$ and $L \to \infty$, we expect that the order parameter $v_a$ in terms of noise $\eta$ should behave like a step function around a critical value of $\eta$. This behavior is typical of systems undergoing kinetic phase transitions[2]. By observing this property in our model, we get a strong indication that the corresponding physical system undergoes a transition. For values of $\eta$ below the critical noise, the value of $v_a$ should be close to one, since the noise is small enough to allow the system to undergo phase transition. For values of $\eta$ above this critical point, $v_a$ should converge to 0; indicating that noise is large enough to prevent the system from reaching phase transition.

However, in our simulation with finite lattice size $L$ and periodic boundary condition, we expect to observe a smoother func-

tion for $v_a(\eta)$. By gradually increasing the number of particles and lattice size while keeping the density $\rho$ constant, we can check wether our system behaves like as predicted above. This scaling analysis also allows us to pinpoint and investigate interesting values of $\eta$ and $\rho$, and understanding their influence on the system's level of order.

## 2. The code

The bird flock is simulated in a python programme [1] which implements the dynamics expressed in Section 1.

### *2.1. Input Parameters*

The main simulation revolves around a class named *Bird()*, which takes a set of eight initial parameters as input:

- Seed: An integer value for the *np.random.seed()* random number generator, which is then used to set the initial positions of the boids. By providing the same seed (1 in our simulations), one gets the same set of random numbers, which ensures the reproducibility of the simulation.
- Vel: The constant speed of the boids, chosen to be 0.33.
- N: The total number of birds in the simulation
- R: The Radius of influence of each bird. It defines the distance within which a bird considers the presence of its neighbours.
- L: Length of the square lattice. This parameter sets the boundaries of the simulation, and periodic boundary conditions are applied to create a toroidal (wrap-around) world.
- $\eta$: Determining the range of the noise, a random perturbation in the direction of the birds ($\theta$) at each time step.
- dt: The length of time steps in the simulation.
- Nstep: The number of time steps which determines how many times we run the simulation.

These parameters collectively define the initial conditions and behaviour of the simulated flock of birds. By varying their values, we later perform a scaling analysis for order parameters in Subections 3.2 and 3.4.

### *2.2. Main.py in the unoptimised branch*

In this section, we describe the first version of our Bird Simulation, which is available in the *Unoptimised* branch of our repository. We call this version unoptimised because it uses a straightforward pairwise approach to compute the distances between all birds, which results in a $O(n^2)$ time complexity. The bird class is first initialized, ensuring that all birds are assigned a random (but reproducible) position and initial direction within the LxL world. The simulation is then evolved in two steps:

First, the code calls the *evolve* function, which updates the position of each bird in the flock based on their current position and direction $\theta$, according to Equation 1.1. After updating the positions, the function applies periodic boundary conditions to keep the birds within the LxL world. For the first timestep only, the birds' positions are evolved according to their randomly assigned $\theta_0$, without considering their neighbours. This behaviour is a byproduct of the fact that our code calls the evolve function before computing the bird's new direction according to their neighbours. This has virtually no effect on the output of our code (one can think of this as the actual simulation starting at timestep 1).

In a second step, the code calls the *new_theta* function, which computes the birds' new direction according to Equation 1.2. Note that even in this unoptimised version, we make use of the fact that the relationship of being a neighbour is symmetric, and once a pair of birds is considered, there is no need to consider them again in the opposite order. Therefore, for each bird i, the code only looks at birds with indices greater than i, and by the time it reaches the last bird (N), all of the neighbours are already known. This optimization eliminates redundant calculations, which in turn makes the process more efficient.

These elements are sufficient to build a working boid simulation. However, the main code has one additional purpose, which is to calculate a kinetic order parameter. Our first intuition was that a good way to check whether our system reaches a steady state would be to check whether the value of $\langle\theta(t+1) - \theta(t)\rangle$ is close to that of the noise $\Delta\theta$. The logic was that, if the difference in the average directions of the birds between two timesteps is close to the noise term, then all birds should be moving in an ordered way. However, we soon realized that this approach would not allow for a proper identification of a steady state. Indeed, the dynamics of a bird flock can exhibit temporal variations in their order. Even in a steady state, the birds can show fluctuations in their collective behaviour due to the inherent stochasticity in the system. This idea was thus abandoned, and its remnants can be found as a hashed function *check_transition* in the bird class of the unoptimised branch.

Instead, the code computes the kinetic order parameter $v_a$, as described in Equation 1.3. As explained in section 1, This parameter provides a quantitative measure of the collective motion of the bird flock. We use the order parameter to detect phase transitions in collective behaviour. In a phase transition, there may be a sudden change in the order parameter, indicating a transition from a disordered to an ordered state. One key point is that, although $v_a$ is computed at each timestep, we later define a mean $v_a$, for which we only average the $v_a$ values for the last fifty timesteps. This choice is justified in Subsection 3.3.

### *2.3. Main.py in the main branch*

In the main branch, a bin sorting system was added to the simulation to decrease the computational intensity of the code, especially for large numbers of birds and size of the world. The

[1] `https://github.com/Avanbreukeleng/BirdSwarmInternationalGang.git`

bin system works as follows: The initial $LxL$ sized world is divided into a grid of square bins of length R. We thus have $(L^2)/R^2$ bins. In cases where this quantity is not an integer, the code chooses an integer for $L_{bin}$, whose value is the next closest value of R, but with $L_{bin}$ always bigger than R. This ensures that the whole world space is covered in bins.

Birds are assigned to bins based on their coordinates. Interactions are then calculated only with birds in the same or adjacent bins. As a result, the algorithm avoids unnecessary distance calculations for pairs of birds that are more than one bin apart, which significantly improves the upward scalability of the model. A step-by-step explanation of the binning process is available in the extensively commented *new_theta* function of the *Bird* class on the main branch of our repository. The runtime decrease derived from using this method is described in Subsection 3.1.

One final consideration is that, because the world is divided into bins of length R, the optimized model yields the exact same output as the unoptimised even in cases where the world size is low as compared to the radius of interaction of birds. In such cases, however, the binning method actually results in a slight increase in runtime, as setting up the bins is not necessary for small world sizes. The binning method thus actually decreases the downward scalability of our model, at least as far as computation time goes, which is of no consequence for our purposes.

### 2.4. Parameter_variation

In addition to the main simulation, we use a parameter_variation file, whose purpose is to mass-produce results for various initial conditions.

This file calls the *Bird* class from main.py using different sets of initial parameters. computes some desirable quantities, such as the density $\rho$, $\eta$, and the order parameter $v_a$, and saves them using the dill library. The code works using the following two classes:

- ParameterModifier: This class is designed to modify a set of input parameters for the simulation. It takes as input an initial set of parameters and allows for modifications of two specific parameters: N and eta. For each value in *new_N*, the class creates a modified parameter set with the updated N value. Identically, each value in *new_eta* is incorporated into a parameter set. The components of these parameter sets are described in Subsection 2.1.

- Bird_Simulator: This class is responsible for running simulations with the different sets of parameters outputted by the previous class and collecting results for further analysis. For each distinct set of parameters, the class initializes a *Bird* instance from main.py, and runs the simulation. The class then computes the density of birds $\rho$, and the order parameter mean $v_a$ (i.e., the average of $v_a$ over the last 50 timesteps). For each instance of the *Bird* class, these values, along with $\eta$, are saved into one of two different arrays. If the values of these three parameters were obtained by varying N and keeping eta constant, then they are saved into the *N_matrix* array. If the opposite is true, then the values are saved in *eta_matrix*. Lastly, the *va_matrix* array simply contains the $v_a$ values for all timesteps and parameter sets, which are all outputed from instances of *Bird*.

### 2.5. Data analysis, Final Plots.ipynb

The last step in the code was to analyse the data. This involved mainly animating the particle motion and plotting the parameter variation with respect to various parameters. For this purpose, the mean $v_a$ of each run was computed as the mean of the $v_a$ of the last 50 timesteps (see Subsection 3.3). And for estimating the errors, the standard deviation over those steps was computed and then it was divided by the squared root of the number of uncorrelated steps (out of the 50). Which should equal to 50 divided by the correlation time. This time was estimated first by computing the correlation of those steps, and looking at which timestep this correlation goes below a certain threshold, which in this case is 0.

## 3. Results and Discussion

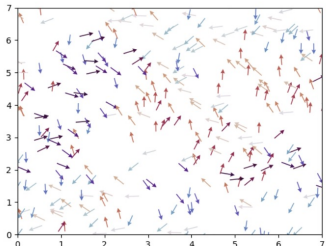
In this section, we present the data obtained from our simulation and discuss how they compare to the theory an the physics described in the literature.

### 3.1. Animation of boid behaviour

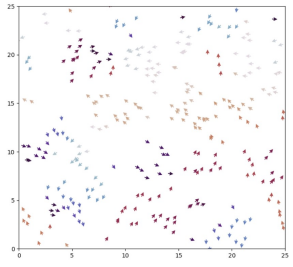
The first result of the animation described hereinabove was the visualisation of the movement of the boids in the box. Figure 3.1 shows four different sets of animations at different points in time. Each boid is represented as an arrow. The colour is included only to better distinguish the different directions. We indeed see the behaviour expected and outlined in Section 1, and it evolves in an organic fashion.

### 3.2. Behaviour of $v_a$ in terms of $\eta$

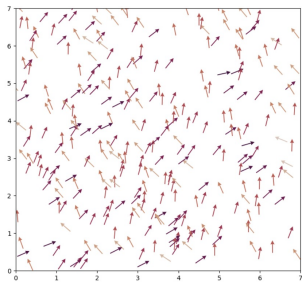
It is crucial to study the behaviour of order parameter as a function in order to investigate the properties of phase transition. For a fixed density $\rho \approx 4$, we ran the simulation for six different system size. As discussed in Section 2, in our model we calculate the $v_a$ at each time step. After sufficiently long simulations, we anticipate minimal variations in the $v_a$ value, indicating that the system has reached the steady state. Figure 3.2 shows the behaviour of the mean value of the last fifty steps of $v_a$ in terms of noise $\eta$ for all the systems. The error bars calculated as described in Subsection 2.5 As indicated by the plot, there are two regimes in our model: before and after the critical value $\eta \approx 4$. For larger $\eta$, the particles follow random directions and do not correlate to each other, in this way the order parameter is very close to 0. With a reduction of the noise comes an increase in the value of the average order parameter. This comes as expected for a system experiencing phase transition, indeed, the particles evolve freely, and after some time reach a steady state in which particles are correlated to each other and follow mostly the same direction. Nevertheless, their order parameter does not reach exactly 1 due to the



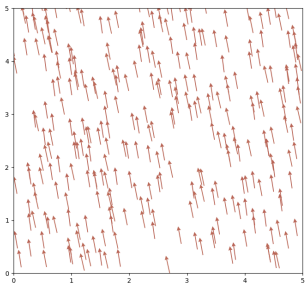
(a) t = 0, L = 7, $\eta = 0.1$: in the first timestep, all the birds acquire a random position and direction.



(b) For small densities and noise the particles tend to form groups moving coherently in random directions, here L = 25, $\eta = 0.1$.



(c) After some time at higher densities and noise (L = 7, $\eta$ = 0.1) the particles move randomly with some correlation.



(d) For higher density and small noise (L = 5, $\eta = 0.1$) the motion becomes ordered.

Figure 3.1: Four steps of different animations are shown. All have N = 300.

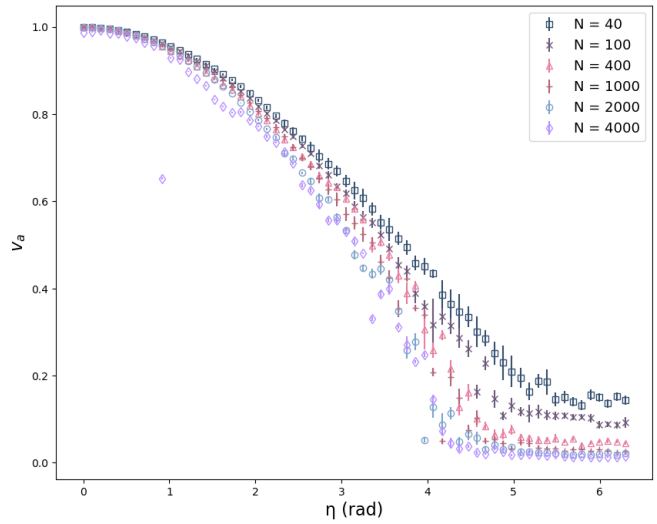



Figure 3.2: The average value of the average velocity ($v_a$) versus the noise $\eta$ in cells of various sizes for a fixed density $\rho \approx 4$.

added noise.

Additional remarks on this plot:

- For each value of $N$, larger noise corresponds to larger error bars, indicating that $v_a$ varies more rapidly. Moreover, for a fixed value of $\eta$, the error bars decrease as $N$ increases.
- For $N = 4000$ near $\eta = 1$, we observed some outlier. This issue is investigated in Subsection 3.3.

### 3.3. *Evolution of $v_a$*

Visualising the order parameter in a system with $N = 4000$ for several values of noise near the outlier ($\eta \approx 1$) implies that if we integrated for a longer time, the system would have reached the steady state (Figure 3.3). Therefore, the mean value of the order parameter plotted does not reflect the true behaviour of the system and should be discarded.

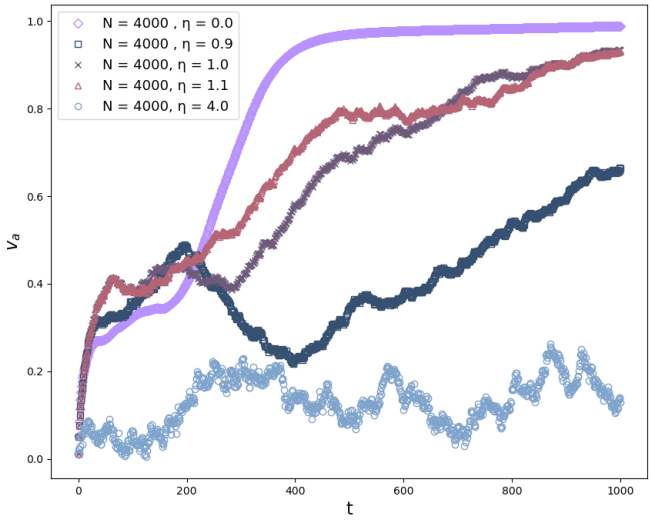



Figure 3.3: The absolute value of the average velocity ($v_a$) versus time steps for a system of $N = 4000$ particles for different value of noise ($\eta$).

It is insightful to compare the above-mentioned system with a system of $N = 1000$ particles (Figure 3.4).

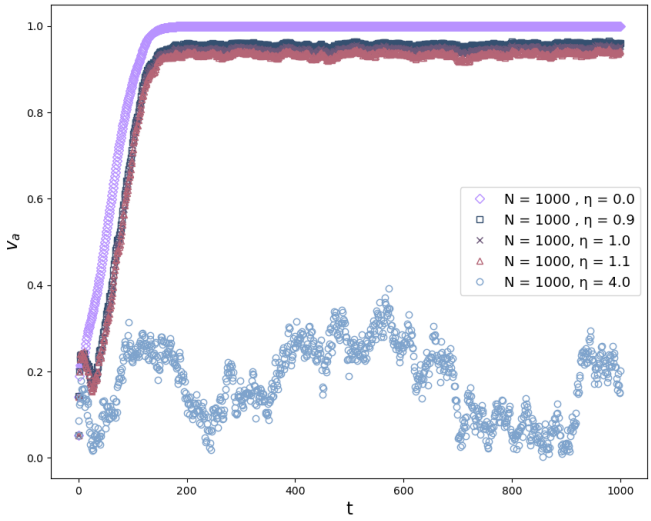



Figure 3.4: The absolute value of the average velocity ($v_a$) versus time steps for a system of $N = 1000$ particles for different value of noise ($\eta$).

- For all values of $\eta$, $N = 1000$ system reach the steady state relatively fast but $N = 4000$ needs more time to hit the plateau. This discrepancy arises because, with a larger lattice length, it takes a longer duration for all the particles to become correlated.

- For all values of $\eta$, the mean value of $v_a$ for $N = 1000$ is higher than $N = 4000$. Because the $v_a$ over a limited number of particles (and system) has a higher tendency towards higher values. That is to say, the average over an infinite system will be lower because any random tendencies of the system will cancel out. Moreover, random fluctuations in larger systems take a longer time to affect all particles, and therefore, during a longer period of time, the system will be out of phase and have lower $v_a$.

Figure 3.4 thus justifies the choice for the mean value of the order parameter, computed from the last 50 of 1000 step simulation. Having a long run as such ensures that the system reaches a steady state (in most cases), whereas taking the average over 50 steps reduces the standard deviation error over this value while avoiding the need for complex data analysis programs.

However, Figure 3.3 shows that the mean value of $v_a$ acquired from simulating 1000 steps for N=4000 is unreliable. The system would need considerably longer times to reach the steady state situation. Therefore, since the data for N=4000, does not represent the steady-state solution of the system, it should only be regarded as qualitative, and any outliers should be ignored.

### *3.4. Behaviour of $v_a$ in terms of $\rho$*

In this part, we show how $v_a$ changes while increasing density for values of noise close to the critical point.

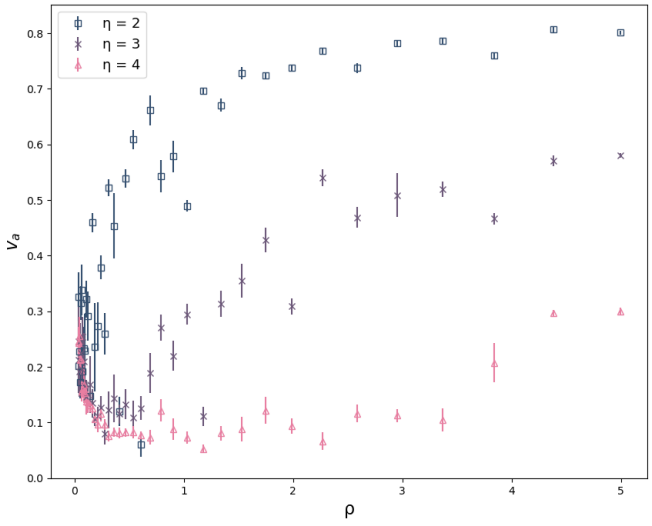



Figure 3.5: The mean value of the average velocity ($v_a$) versus density for different value of noise ($\eta$), but always with L=20.

From Figure 3.5 we can deduce:

- As $\eta$ increase, $v_a$ values decrease.

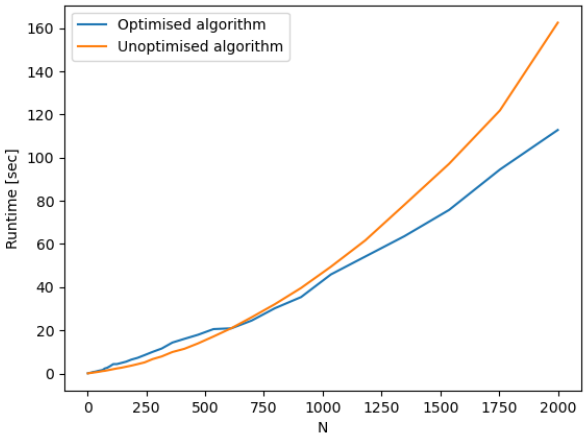



Figure 3.6: Runtime of L=20,$\eta$=1, 500 timesteps for varying N. The two algorithms are shown in the same plot for ease of comparison.

- As $\rho$ increases, $v_a$ values increase, indicating that the occurrence of a phase transition is influenced not only by noise but also by density. Due to each particle taking the average of more neighbours, and thus, being less affected by noise.

Noting the behaviour for densities lower than unity, it is important to realise that these densities are related to very low numbers of particles. At some point (below $\rho \approx 0.5$), the effect of low particle numbers overcomes every other measurable dynamic. Hence, particles will tend to jump from being correlated to not so in every step, leading to large and different values of the order parameter. Indeed, for a single particle $v_a$ is always = 1, and for two particles it will either have this value if they are grouped or some other random value closer to 0. Regardless, these situations are unphysical and therefore not of interest.

Moreover, similarly to Figure 3.2, there are some outliers in the graph that should be ignored. For instance, this is the case for the simulation at a density value of 2 and $\eta$ 3. These points just need more time to reach to the steady-state solution.

### *3.5. Optimised vs Unoptimised Runtime*

For the purpose of this project, it was also important to study the effect of the optimisation algorithm on the runtime. At first, the bin computation seemed to deteriorate the efficiency of the code, however, once both versions had the correct periodic (and transparent) boundaries implemented, the impact was clear. For a single run with 5000 boids and a 100R length, the optimised code performed 50 steps 1.8 times faster.

Figure 3.6 shows how the binning algorithm (Subsection 2.3) performs better for large N, even at low length scales. If we were to increase the size of the box we would see a shift upwards of the unoptimised algorithm (Subsection 2.2 with respect to the optimised version.

## 4. Conclusion

The purpose of this project was to build a program that would simulate the motion of self-driven particles with the theory proposed by Vicsek et al. [2], and to study their phase transition. The theory and methods behind the Python program have been discussed and the results analysed.

Overall, the program performs as expected and the obtained data is consistent with that of the literature. Our code presents animations that resemble organic systems and are able to simulate the evolution of a system of particles and provide their natural steady-state solution. From its dependence on the noise level, we can conclude that the system indeed experiences phase transition, which is more pronounced with a greater number of boids, and larger length scales. Moreover, the dependence of this solution with respect to the density of particles, indeed shows how particle correlations increase with their number. Lastly, the two versions of the code, with and without a binning algorithm, were compared and the effectiveness of the optimisation was established.

This paper therefore shows how even simple programs following simple mechanics are very effective in simulating physical systems of interesting dynamics. The scope of this project was initially limited to swarms of birds, but similar algorithms could be used to study other emergent behaviours in physics, biology and even crowd dynamics.

The authors therefore would like to encourage future researchers to follow on this idea: the code is in a public repository and future research could take inspiration from it, and make a more extensive study of the case in question. We recommend studying the behaviour of this systems of particles with additional parameters, running the program with different random seeds to make sure the results converge (e.g. with montecarlo), and implementing additional rules to the mechanics of the individual particles (such as attraction, repulsion and acceleration).

## References


[1] Craig W. Reynolds. "Flocks, Herds and Schools: A Distributed Behavioral Model". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 25–34. ISBN: 0897912276. DOI: 10.1145/37401.37406. URL: https://doi.org/10.1145/37401.37406.

[2] Tamás Vicsek et al. "Novel Type of Phase Transition in a System of Self-Driven Particles". In: *Phys. Rev. Lett.* 75 (6 Aug. 1995), pp. 1226–1229. DOI: 10.1103/PhysRevLett.75.1226. URL: https://link.aps.org/doi/10.1103/PhysRevLett.75.1226.