Documentation of InformatiCup 2025 Participation

versjon[1] und kro-ma[2]

Team: "team name not licensed under the GPL3"

Abstract: This file documents our participation in the InformatiCup 2025, where we developed a real-time Boids simulation within the Godot engine as both a technical demonstration and a benchmark for the NumDot scientific computing library. By implementing flocking behavior using both native GDScript and vectorized NumDot operations, we compared the expressiveness and performance of the two approaches. While NumDot enables elegant and scalable tensor-based algorithms, our benchmarks showed that native GDScript remains more efficient for small to medium swarm sizes under current conditions. We discuss the challenges of testing emergent behaviors, highlight our collaboration with the open-source community, and provide practical guidance for reproducibility. Our results offer insights for future optimizations and showcase the integration of scientific computation in interactive game engines.

1 Introduction

As part of the InformatiCup 2025, we had the opportunity to actively contribute to the further development of the Godot ecosystem. The Godot Engine is an established open-source game engine that enjoys growing popularity among indie developers and in research. Thanks to its open architecture, it allows for deep extensions via custom modules or add-ons written in GDScript or C++.

After reviewing several suggested topics in the Godot Engine space, we chose to implement a Boids simulation. This is a classical form of swarm intelligence modeling, where agents (“boids”) behave according to simple rules such as alignment, separation, and cohesion [Re87]. The interplay of these local rules results in globally emergent group behavior—an ideal field for algorithmic concepts, data structures, and visual simulation.

Godot’s modular, open architecture provides an excellent foundation for developing such a simulation. Our goal was to implement the boid logic both

[1] TU Berlin, Bachelor student in Computer Science,
[2] TU Berlin, Master student in Computer Science and Attorney at Law (Germany),

with native Godot scripting and with the external scientific add-on library NumDot. The latter enables efficient numerical calculations within Godot using C++ and xtensor. The implementation was carried out as part of Issue 48 in the NumDot repository.

The Boids simulation not only offers an interesting technical challenge but is also well-suited as a demonstrator for interactive, algorithmic visualizations.

2 Motivation

The motivation for our project arose from the desire to apply algorithmic knowledge in a real open-source context—in an established project with an active community. The Godot Engine was a natural choice: open-source, modular, and highly popular among both indie developers and researchers. However, our initial research revealed that contributing directly to the Godot Engine is not easy.

Godot's official development policy is quite strict: Pull requests are only accepted if they are well documented, mature, and consensus-driven within the community. Additionally, there is a huge backlog of open issues and feature proposals, some of which have been pending for years. Without deeper integration into the developer team or long-term experience, it is difficult to make a contribution that gets merged, let alone integrated.

We therefore opted for a more pragmatic approach: Instead of working directly on the engine, we chose a project that is both technically and community-wise close to Godot but with lower entry barriers—the scientific computing framework NumDot. This project is under active development, open to contributions, and offers exciting use cases.

The decision to implement a Boids simulation as part of InformatiCup 2025 comes from the desire to experience algorithmic concepts not only theoretically but also practically. Swarm behavior as simulated by Boids is a well-known example of an emergent system. The underlying rules—alignment, cohesion, and separation—are simple, yet they lead to complex, realistic movement patterns.

The topic became especially exciting due to the possibility of implementing the calculations not only in GDScript but also using the scientific extension NumDot. This project provides tensor math and scientific computing functions, similar to NumPy or SciPy in Python, but fully embedded in Godot. This allowed us to analyze and compare performance differences and implementation options between traditional engine logic and a specialized mathematical library.

The Boids simulation was thus identified as an attractive visual project and as a suitable benchmark for evaluating the performance and flexibility of NumDot.

3 Project Selection

During the topic selection process, we explored a range of possible entry points in the Godot ecosystem: from feature proposals and open add-on issues to experimental PRs and community forums. It quickly became clear that many of these suggestions are discussed long-term but rarely implemented—not least due to Godot's strict governance structures.

A promising candidate was the open-source project NumDot, which brings scientific computing directly to Godot. We were particularly drawn to Issue 48, where a Boids simulation was suggested as a demo—a classic example of swarm behavior and an excellent testbed for numerical algorithms.

Since the project is actively maintained and has a clear, beginner-friendly contribution guideline, it offered a realistic opportunity to make a meaningful pull request. Moreover, the Boids simulation provides an accessible way to demonstrate central computer science topics such as scientific computing, software development techniques, and algorithmic design.

By implementing two versions—one with classic Godot logic, one with NumDot—we were able to evaluate and document both the practical suitability and performance of each approach.

4 Development Phase

This section describes the concrete implementation of the project, with emphasis on the following points:

4.1 Planning and Conceptualization

At the outset, we defined clear requirements for our Boids simulation implementation:

- **Agent behavior:** Implementing the fundamental rules of alignment, cohesion, and separation.
- **Visualization:** Real-time rendering and control in Godot.
- **Performance analysis:** Comparative evaluation of GDScript and NumDot approaches.
- **Community involvement:** Adhering to contribution guidelines and coordinating with the NumDot community.

4.2 Technical Implementation in GDScript

The first implementation was carried out entirely in GDScript within the files `demo/demos/boids_simulation/boids_model.gd` and `demo/demos/boids_simulation/gd_solver.gd`. The boid logic implemented in GDScript is found in `gd_solver.gd`, and `boids_model.gd` contains the model behavior also used by the NumDot implementation (`demo/demos/boids_simulation/nd_solver.gd`).

The rules for alignment, cohesion, and separation were programmed using Godot's standard functionality. The following code excerpt from our implementation illustrates the calculation of these rules per boid:

```
# Calculate separation, alignment and cohesion per boid
var separations: Array[Vector2] = []
var alignments: Array[Vector2] = []
```

```
var cohesions: Array[Vector2] = []
for i in range(params.boid_count):
    var separation := Vector2(0, 0)
    var alignment := Vector2(0, 0)
    var cohesion := Vector2(0, 0)
    for j in range(params.boid_count):
        var distance = (positions[i] - positions[j]).length()
        if distance < params.range * 0.5 and distance != 0:
            separation += (positions[i] - positions[j]) / (distance ** 2)
        if distance < params.range:
            alignment += directions[j]
            cohesion += positions[j] - positions[i]
```

This code shows how each of the three fundamental rules is calculated for each boid. *Separation* ensures a safe distance between boids, *alignment* aligns movement directions among neighboring boids, and *cohesion* keeps the flock together by moving each boid toward the centroid of its neighborhood.

The mathematical basis for these three rules can be directly inferred from the code above.

Separation causes boids to move away from each other if they are too close, mathematically achieved by dividing the vector between two boids by the square of their distance—so the closer two boids are, the stronger the effect.

For *alignment*, we sum the movement directions of all boids within a defined radius (`params.range`), resulting in each boid aligning its own movement with the average direction of its neighbors.

The *cohesion* rule makes each boid move toward the center of its neighbors by calculating and summing the relative vectors.

Together, these calculations produce emergent, natural-looking flocking behavior, as illustrated in Figs. 1 and 2.

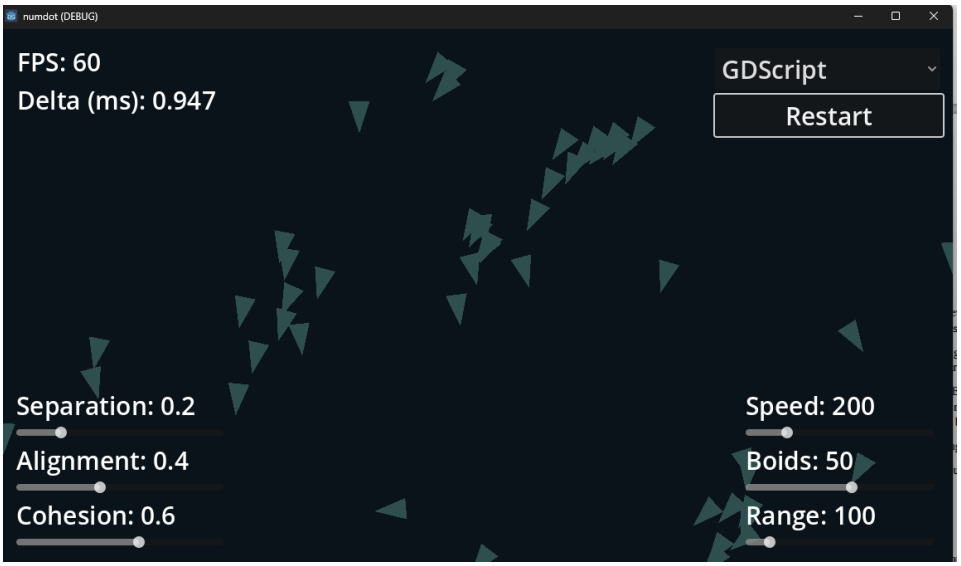

Fig. 1: A loosely moving flock resulting from moderate settings for separation, alignment, and cohesion.

4.3 Integration of NumDot – Comparison to Godot Implementation

In the next step, we implemented the Boids simulation using the scientific add-on library NumDot. Whereas the classic GDScript variant computes each boid rule (separation, alignment, cohesion) in explicit nested loops over all agents, NumDot enables a much more compact and efficient solution using vectorized array operations. Instead of iterating for each agent, all position and direction differences and the resulting computations are applied to entire tensors at once.

For example, the neighborhood calculation in GDScript looks like this:

```
for i in range(params.boid_count):
    for j in range(params.boid_count):
        var distance = (positions[i] - positions[j]).length()
        # further calculations
```

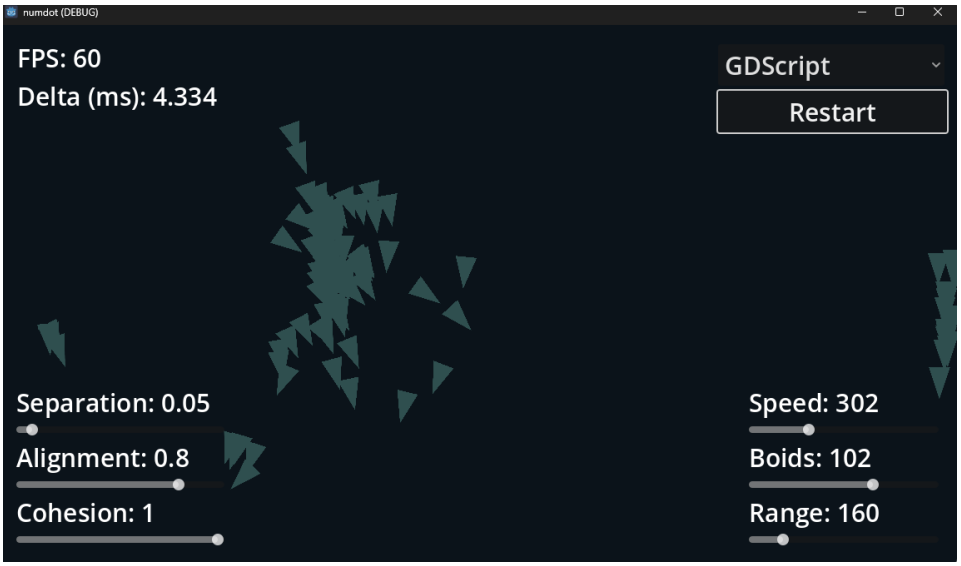

Fig. 2: Rapid and dense flocking behavior with high settings for cohesion and alignment.

In NumDot, these calculations are achieved through array broadcasting, which improves readability and significantly boosts performance for large numbers of boids:

```
var position_differences := nd.subtract(
    positions.get(&":", &"newaxis", &":"),
    positions.get(&"newaxis", &":", &":"))
)
# Further operations are performed directly on these arrays.
```

This vectorized approach allows for efficient and simultaneous application of the behavioral rules to all boids. Compared to the classic Godot implementation, this should yield much better performance and scalability for large flocks, while the overall simulation process remains identical, just more compact and mathematically elegant with NumDot.

4.4 Challenges of Unit Testing

Throughout development, we carefully considered how unit tests might be integrated into the Boids simulation, especially for the underlying mathematical operations. Our aim was to formally verify expected behaviors—especially cohesion, alignment, and separation—using unit tests. In practice, we encountered several fundamental challenges:

- **Difficulty testing emergent systems:** Boid swarm behavior is based on local interactions, which produce complex and emergent global patterns. Such systems are hard to test with conventional unit tests, which typically check deterministic, clearly scoped functions. Emergence only becomes evident in the interaction of many agents over multiple iterations and is thus not easily captured by isolated test cases.
- **Lack of established test frameworks:** We explored the feasibility of using existing unit test frameworks for Godot (e.g., `godot unit testing 4`) or framework-independent approaches. While a comparison of basic functions like `nd.add` against established libraries such as NumPy would be conceivable, there is currently no comprehensive test suite for tensor libraries that can be readily transferred to NumDot.
- **Lack of automation for complex tests:** Fully automated, reproducible tests for complex simulations would require additional tools or reference data. Developing a systematic test infrastructure (e.g., a test plugin or custom test suite) was beyond the scope of this project.

In summary, classic unit testing—and especially integration testing—was only possible to a limited extent in this context. Instead, functionality was mainly ensured via interactive demos and manual comparison of simulation results to established references. Visual feedback from the Boids demo was especially helpful in quickly identifying and correcting deviations or unexpected behavior.

A successful completion of the automated build process using GitHub Actions (see build log) provides formal verification of the technical integrity of our implementation. The platform compiles NumDot for all target platforms and ensures that the extension can be built without errors under various system

architectures. This guarantees that the core functionality is fundamentally operational and integration-ready across environments.

As part of continuous integration (CI), our implementation is automatically built for all supported platforms. A successful build demonstrates the technical consistency and portability of the project; any errors or incompatibilities are detected early. This automated check is an important formal quality assurance mechanism in the development process.

5 Performance Comparison Based on `delta` Values

To assess the efficiency of the NumDot implementation compared to the classic GDScript variant, we measured the execution time per simulation step (frame). The central metric is the so-called `delta` value:

`delta` = Time in seconds per frame

A low `delta` value is crucial for smooth real-time simulation.

Comparison Tables: Frame Duration for Different Boid Counts

Boid Count	GDScript `delta` (ms)	NumDot `delta` (ms)
10	0,3	3
50	2	6
100	6	13
300	13	100

Tab. 1: Comparison of average frame duration (`delta`) per frame with separation = 0.2, alignment = 0.2, cohesion = 0.6, speed = 200, and range = 100.

In addition to the number of boids, the configured vision range (`range`) has a significant impact on performance. A larger vision range means that each boid interacts with more neighbors simultaneously, which especially leads to noticeable performance drops in the GDScript implementation.

Boid Count	GDScript `delta` (ms)	NumDot `delta` (ms)
10	0,5	4
50	3	7
100	6	13
300	24	106

Tab. 2: Comparison of average frame duration (`delta`) per frame with separation = 0.2, alignment = 0.2, cohesion = 0.6, speed = 200, and range = 1000.

Analysis

Tab. 1 shows that for small boid counts (10–50), both implementations achieve very low frame durations, with GDScript generally performing faster (0.3 ms vs 3 ms at 10 boids, 2 ms vs 6 ms at 50 boids). As the number of boids increases, the difference becomes more pronounced: at 100 boids, NumDot requires 13 ms per frame while GDScript still manages with 6 ms. With 300 boids, the gap widens further, with NumDot taking 100 ms per frame compared to GDScript's 13 ms.

A similar trend is observed with increased vision range (`range` = 1000) as shown in Tab. 2. For small flocks, GDScript is faster (0.5 ms for 10 boids vs 4 ms for NumDot), and as the boid count increases, NumDot's performance degrades more rapidly (24 ms vs 106 ms at 300 boids).

Interestingly, in this setup, the classic GDScript implementation remains faster than the NumDot version for all tested flock sizes—even when vision range increases. The expected advantage of vectorized operations in NumDot does not manifest at the tested scales; in fact, GDScript's lower overhead and native integration allow it to outperform NumDot for all boid counts.

These results indicate that, for small to medium flock sizes, NumDot currently offers no performance advantage over the classic GDScript implementation—in fact, GDScript is usually faster. The benefits of vectorized computation may only become apparent for much larger flocks, more complex neighborhood calculations, or with future optimizations in NumDot. For the scenarios tested here, the native GDScript solution remains the more efficient choice.

6 Community Interaction

After announcing our intention to work on the relevant issue, we closely coordinated with the maintainer. Open questions were clarified early via Discord, and our planned approach was communicated transparently. This laid the foundation for constructive collaboration and ensured that our contribution met the community's technical and conceptual expectations.

During the pull request process, the community was appreciative and supportive. The maintainer explicitly acknowledged the work done and indicated that further improvements or enhancements via additional pull requests are welcome at any time. The current performance situation was also addressed: it was acknowledged that the Godot variant is currently faster than the NumDot version, which was seen as motivation for further optimization in NumDot's core. The openness to further contributions and performance improvements was explicitly emphasized.

There was also a technical discussion on the testability of the implementation. It was jointly noted that classic unit tests are of limited use for complex, emergent swarm behavior. Instead, discussions focused on the possibility of testing basic numerical operations against established reference libraries. Overall, the community confirmed that manual comparison of simulation results and visual feedback from the demo are currently the main means of verifying functionality.

This close, constructive collaboration with the community was formative for the project's progress and significantly facilitated the integration of our contribution.

7 Installation

The pull request for the Boids simulation was successfully merged into the main NumDot repository. As NumDot remains under active development, and setup can sometimes be complex, we recommend referring to the official documentation.

To install and run locally:

1. Clone the repository with all submodules:
   ```
   git clone --recurse-submodules https://github.com/Ivorforce/NumDot
   ```
2. Install and build according to the official instructions: NumDot Build Setup
3. Then open the project with Godot 4.3 and select the **Boids Demo** from the demo directory.

Note: Due to the complexity of the setup and possible dependencies, we recommend checking the latest issues and discussions in the repository to benefit from ongoing developments.

8 Conclusion and Outlook

The implementation of the Boids simulation for NumDot demonstrates how algorithmic concepts can be practically integrated into an open-source ecosystem. The comparison of classic and vectorized implementations not only provides valuable insights into performance aspects but also lays an ideal foundation for future optimization within NumDot.

A visual impression of the developed simulation can be found in this demo GIF: https://github.com/kro-ma/NumDot/blob/main/boids_demo.gif. The Boids demo is also included in the current repository and can be tried out directly after installation.

The work was developed in close coordination with the NumDot community on NumDot Discord.

There are numerous opportunities for further development—for example, further optimization of vectorized algorithms, additional visualization features, or expanded automated tests and benchmarks. NumDot's flexible architecture and the openness of the community provide an excellent foundation for this.

References

[Re87] Reynolds, C. W.: Flocks, herds and schools: A distributed behavioral model. SIGGRAPH Comput. Graph. 21 (4), S. 25–34, 1987, https://doi.org/10.1145/37402.37406.