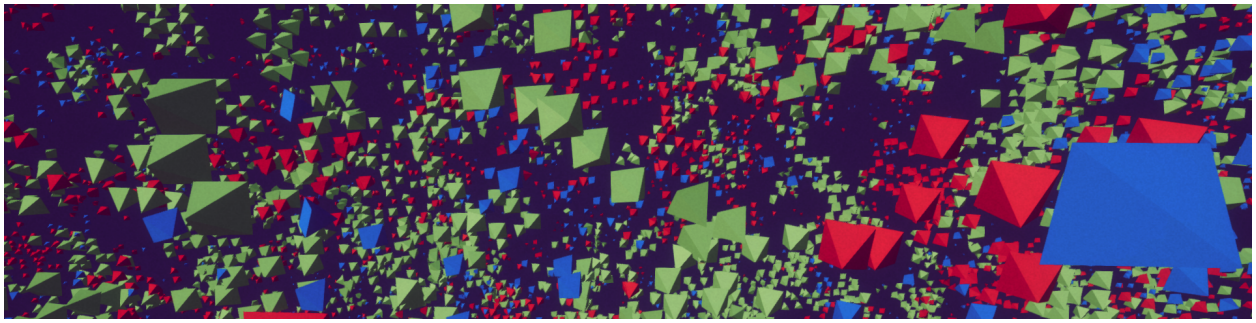

10 Chambers Programmer Test Documentation

Implementation details:

I have developed this project using DOTS in its experimental version 1.0 and requires **Unity 2022.2.0b8** which can be downloaded in the following link (https://unity3d.com/unity/beta/2022.2.0b8) or through the Unity Hub (unityhub://2022.2.0b8/9eb452e2ea43).
For more context, DOTS (Data-Oriented Technology Stack) is a series of packages based on Data Oriented Design, which allows to have efficient code by default and multithreaded. For more information about DOTS you can visit the following link: https://unity.com/dots
For more information about Entities 1.0 experimental visit the following link: https://forum.unity.com/threads/experimental-entities-1-0-is-available.1341065/

In addition to the behavior given in the original project, I added more options to instantiate up to 65k boids with the numbers up to 8. Then the number of boids to instantiate would be as follows: {64, 256, 1024, 4096, 8192, 16384, 32768, 65536}

Besides the above change, the project works in the same way as the original.

Performance investigation

Issues:

- Since everything was running on the *main thread*, worker threads were not being used and were idle the whole simulation.

- All the simulation, especially *Find Neighbours*, was very expensive if we scale up the boid count.
- It is using GameObjects and creation and destruction of GameObjects is very expensive.
- Since boid data is the data that is being iterated the most in each frame, having it in GameObjects makes it so that you can't do as many optimizations, such as using Burst. In addition, the managed objects are scattered around memory, so this can lead to slow performance and cache misses.
- Managed objects like Components or GameObjects are garbage collected, so it can create performance problems and spikes.
- The algorithm used to find neighbours generated *Garbage Collection* because it was adding elements with resizing to the List of `allWithinRadius` and `sameTeamWithinRadius`, also passing these lists as out parameters.

Solution:

- The main solution to most of the problems that arose while analyzing this project is the efficient organization of data. And this is the strongest point of DOTS in Unity. For this reason I decided to take the same behavior of the project but redo it completely using DOTS. This allowed me to have an efficient modeling of all the data I wanted to read and modify.
- Using *jobs* allows me to use the CPU threads to perform several operations concurrently. This way I could take advantage of the whole CPU and not only the Main Thread.
- To further improve performance and reduce the Garbage Collection, I used `ISystem`, which is a struct and allows me to have my systems in unmanaged code and thus be able to use the Burst compiler that improves performance in a very significant way.
- Initially, I had made an algorithm similar to the one in the project to find the neighbors using jobs, but even with this change the performance did not scale so well. Thus, I looked for other ways and found an algorithm to divide the world into quadrants. In this way, the boids only look for neighbors in their quadrants, which allowed me to significantly improve the performance.

Additional Changes

- Some graphical changes were made that do not affect much the performance of the test and looks a more visually pleasing, but feel free to remove them if you wish.

    - I wanted to experiment with another type of boid, so I made a simple model of a prism, which allows to better identify the direction of the boids. However a prefab with the original cubes is provided.
    - A simple interface was created to show the current counter of boids in the simulation.

Potential further performance improvements

- Review different search algorithms and do performance tests to see which one works best for this type of simulation.
- Study the chain of dependencies between the different behaviors more closely to be able to do some operations in parallel and not just wait to do the operations in strict order.
- In-depth study of the input and output data to optimize some processes and hard code some data that doesn't change during the simulation.

If you have any questions or would like more information please do not hesitate to contact me.