Progress Report

James Hamilton

registration: 100340006

1. Background and Motivation

I have been tasked with designing and modelling an iteration upon Conway's Game of Life (Gardner, 1970). My implementation is an interactive educational tool, consisting of a 2D space in which particles engage in physical interactions, somewhat alike to gravity, albeit much simpler. The idea being, complex behaviour emerges from an accumulation of simple interactions. I model a collection of n particles of t types, of which a matrix of t values manipulates the applied attraction/repulsion on particles during interaction, then to avoid infinite velocity, applying friction. I've drawn heavy inspiration from existing systems, like (Mohr, 2023) and the collaborative works of (Schmickl et al., 2016) and (Schmickl and Stefanec, 2019).

Having experience using OpenGL (KhronosGroup, 2023) and abstractions for creating small simulations and games, I want to further develop my skill-set in the area. I've diverged to a physics-based model, designing a system not bound by a discrete domain. Though measuring the extent to which a model displays complex behaviour is notoriously difficult to measure, in my preliminary research, I found my human interpretation of such, to be higher in continuous models.

2. Planning

2.1. Aims and Objectives

Primarily, I aim is to convey complex behaviour. A measurement I'll be treating as qualitative; quantitatively measuring such behaviour is not within the projects scope. Though I can confidently my model displays it, as seen in C.

Real-time simulation is also a primary aim; the user is to observe as it unfolds. To ensure a quality user experience, I'll subject my model to heavy optimisation. Intermediate optimisation is seen in prototype 1 3.2.

Statistical analysis is also a key objective. This is to provide comparative results of simulations ran with different parameters and consequently facilitate educational user interaction with the model. Development has not yet begun in this area.

Keeping in line with the educational aspect of my solution, user friendly interaction with the model is key. Prototype 1 sees limited user interaction.

See my Moscow statement D for a feature list representing my aims and objectives.

2.2. Tools and Resources

My chosen programming language is C++17 (ISO, 2020). Providing efficient compiled code, object orientated paradigms and an extensive list of modern programming language features, its an ideal choice for my project.

For handling graphics and user input, I'm using a new upcoming open-source OpenGL (KhronosGroup, 2023) abstraction written in C (ISO, 2018), called raylib (raysan5, 2023). It has wrappers for C++ and is specifically made for my purpose. It also grants me low level access to graphics, which I'll need to utilise for optimisation, while also saving me from the usual OpenGL (KhronosGroup, 2023) boilerplate.

My IDE is vscode (Microsoft, 2023a) I did use a raylib (raysan5, 2023) project template (educ8s Nick Koumaris and frotzen Elwin Beall, 2023), featuring its own makefile. In the past, I've used tools to handle linkers and compilers, doing so manually is something I want to learn, however, is not a focus for the project. For prototype 2, I'll experiment with writing my own CMake (Cedilnik et al., 2023) file.

Version handling is achieved using Git (Torvalds, 2023) and Github (Preston-Werner et al., 2023). Both of which are industry standard, hence my prior experience using them.

I'm developing my solution on Windows 11 (Microsoft, 2023b), I considered using my Ubuntu subsystem, however, decided against it, to ensure compatibility with the lab computer I'll be demonstrating on.

2.3. Methodology

I've taken an iterative prototype-based approach to developing my solution, following SCRUM guidelines outlined in (Schwaber, 1995), scaled down to an individual context.

My standup meetings are every two weeks with my project supervisor Vincent Moulton, where we discuss my progress on proposed development. Rough handwritten minute notes are taken so I have a reference of what to do that fortnight.

There will be two prototypes, the feature list for each of them can be seen in E. The first one is primarily for getting comfortable with the codebase and raylib (raysan5, 2023), whilst experimenting with the model ruleset. The second one is to apply lessons learned in the previous one, whilst implementing features that require less experimentation and merging it all into a user-friendly package.

The iterative approach cascades down to implementation of individual features. Beginning with a rough plan of how I am to implement a feature, creating a new branch in the GitHub (Preston-Werner et al., 2023) repository and experimentation begins of the feature. Once I'm satisfied the feature matches its requirements, I merge it into the main branch.

As of the writing of this report, prototype 1 is completed.

3. Successful Implementations

3.1. Simulation Rules

3.1.1. Particle Motion Law

With the completion of my first prototype, comes the first version of my particle motion law. In the experimentation leading to this algorithm, I did try implementing equal reactions for interactions, however, this resulted in the particles quickly reaching equilibrium. The results were not very lifelike. Though I have not constricted myself to the laws of physics in my model, it is to be noted that my current algorithm breaks conservation laws of momentum. The ferocity of particles interaction does not consider velocity, mass or momentum, it is completely determined by distance and the arbitrary value assigned within the attraction matrix. See appendix A for a mathematical notation and appendix B for a pseudocode description.

3.1.2. Bounds Handling

In the final version, the scene is to be modelled as a torus square, however, for the first prototype, I just have particles bounce of the scene bounds. I've been careful to handle fringe cases in bounds checking, I calculate the bounds as being the smallest possible float below the whole number I wish to set the boundaries to. This ensures my spatial hash implementation 3.2.1, cannot produce indexes that are out of bounds.

3.2. Optimisation

3.2.1. Spatial Hash

Spatial hashing is the concept of grouping physics object references into cells within a grid representing sections of the simulation space. These cells can then be queries to check for particle collisions within a 3x3 neighbourhood. Seeing as all particles have the same acting range, I set the in-world size of the cells to be squares of the outer particle range dimension.

GameDev.net has some helpful reference material (MacDonald, 2009), though my situation has distinct differences, primarily being that all my objects are of the same size. As consequence, I've applied further optimisation in my implementation.

In prototype 1, I took a procedural approach to gain a level of understanding around the concept. It consists of a 2D standard library vector of pointers to particles, that is dynamically scaled as particles within cells increase in density. It currently only scales upwards and keeps track of the count of particles in each cell, meaning extra memory work is only executed when the capacity of a cell is created.

I plan to put the spatial hash into its own class to have as a member of the simulation class. I also plan to experiment with other data structures, benchmarking them to see which best suits my purpose.

3.2.2. Batched Rendering

When I first started experimentation, to get myself going, I was using raylib's (raysan5, 2023) "DrawCircle" function to draw particles. This quickly became cumbersome, as it draws many triangles to comprise a circle. It also reset the texture and render mode every time. Too many draw calls meant rendering was slow.

I have circumnavigated this issue, by producing a 64x64 circle texture, then when it comes to rendering, drawing all the particles without changing OpenGL's Khronos-Group (2023) allocated texture and without changing rendering mode. This means the render texture is not reset, there is less back and forth communication from the CPU to the GPU.

After implementing batched rendering, the bottleneck became detecting collisions and apply particle motion law.

3.2.3. Algorithmic Optimisations

The code that handles particle motion law is currently the largest bottleneck in my code. The pseudo code version seen in the appendix B, does not see any optimisation.

My optimised implementation avoids calculating square roots until needed, by checking square deltas are within acting range, then once an interaction is determined, calculating the delta by square rooting its square. Position, velocity and attraction matrix row are cached to avoid unnecessary referencing and to improve cache efficiency.

This has fruited visible improvement to performance, however, concrete testing is needed to quantitatively prove its improvement.

3.3. User Interaction

User interaction was kept to a minimum in prototype 1, and what was implemented, was mostly just for debugging. As it stands, user interaction comes in the form of panning around the scene with right-click-drag, pausing the simulation with "SPACE" and toggling draw grid with "G".

The second prototype will introduce more meaningful user interaction, as seen in E.

4. Failed Implementation

4.1. Reproduction

In the first prototype, I spent time experimenting with reproduction laws. However, I quickly came across significant issues and ultimately decided revoke the feature. The idea was to set the conditions in which a set of interacting particles can reproduce.

The problem with implementing a reproduction law in my model, is that without it, particles create somewhat stable structures, where interacting forces maintain a degree of equilibrium, but once a new particle is introduced in such a way that there is not physical interaction before its existence, this equilibrium is broken. As such, reproduction disrupts the formation of self-organising structures, which is an integral part of my model.

There may be a less elegant solution to this problem, but to maintain the flow of development in my project, I have decided to drop reproduction and under/over population laws. The state of prototype 1 proves that I can display complex behaviour without it.

4.2. Game of Life Analysis

I did initially plan to perform a statistical analysis of Conway's game of life (Gardner, 1970), to compare it to results gathered in my own model. However, after reviewing this concept, I've decided that the statistics are simply not comparable. My model is set in continuous 2D space, game of life is set in a discrete 2D grid. Consequently, comparing results from both wouldn't be of use.

This was an oversight on my behalf, during the earlier stages of my design.

5. Evaluative Summation

As discussed in 4, I will no longer be implementing under/over population and reproduction, and I won't be statistically analysing Conway's game of life. I've also decided to take a more object-oriented approach to my program in the second prototype. Doing things procedurally helped quicken the experimentation phase, but I would like a stronger foundational structure within my final product.

Completion of the first prototype was slightly behind schedule, as such so was the retrospective. Retrospective was also more time consuming than originally anticipated, though development has not suffered from this inaccuracy, the timeframe of the second prototype and retrospective has been adjusted.

Any changes in timeframe have been described in the updated Gantt Chart G.

In terms of risk analysis, I believe much risk has been with what I've learnt during experimentation in prototype 1. For the final prototype, I'll be spending more time deciding on architectural decisions and following best practices. I've given myself plenty of leeway for completion of the finished product, I'm confident that development of my solution will be a success.

A. Particle Motion Law Equation

$$\vec{v}_i = \vec{v}_i + \sum_{j=1}^{n} \frac{\vec{\Delta}_{ij}}{\|\vec{\Delta}_{ij}\|}(1-r) \begin{cases} 1 - \frac{s}{r} & \text{if } \|\vec{\Delta}_{ij}\| \leq s, \\ \mathbf{A}_{\mathbf{t}_i \mathbf{t}_j}(\|\vec{\Delta}_{ij}\| - s) & \text{otherwise.} \end{cases},$$

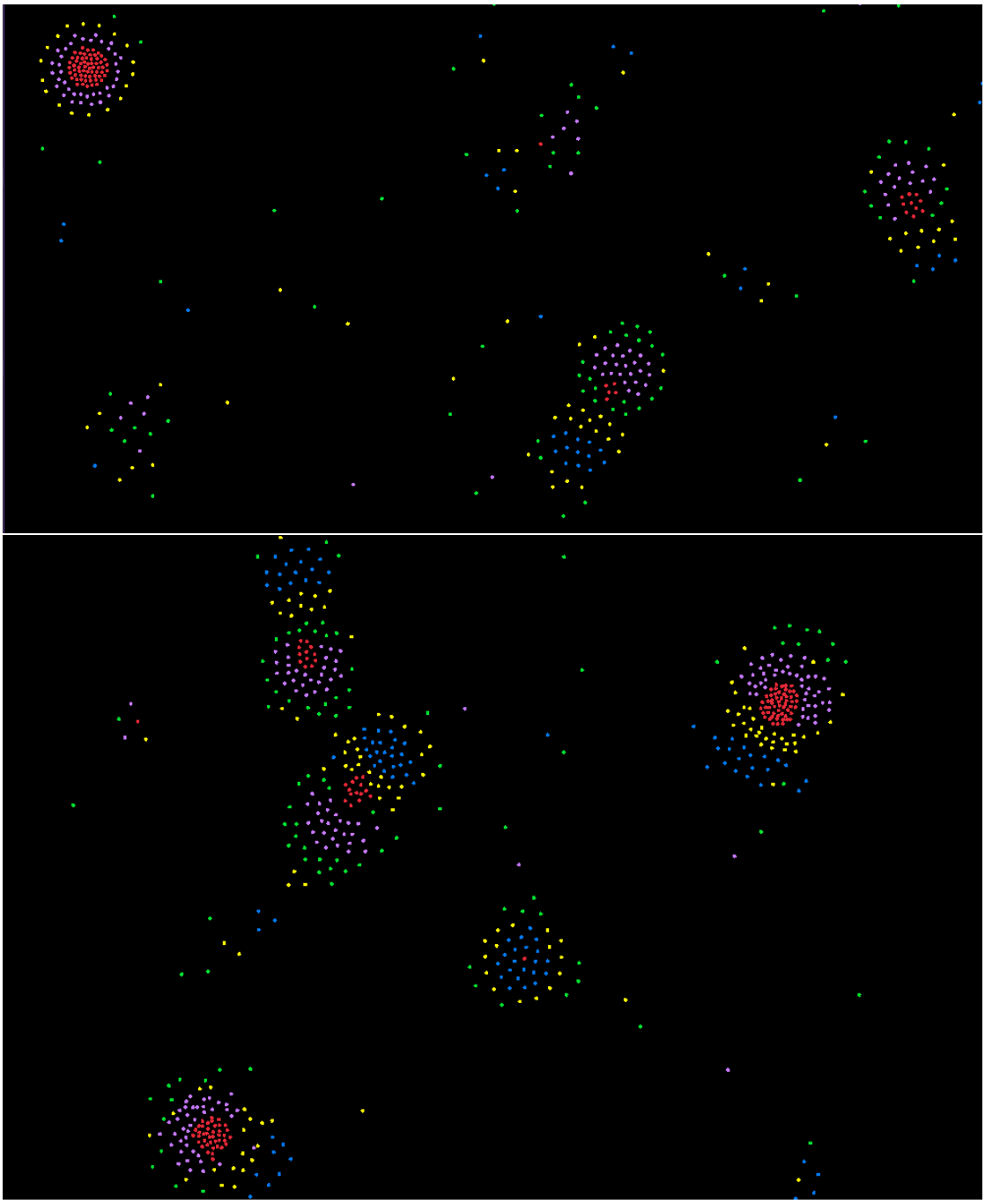
where
- $\vec{v} \in \mathbb{R}^2$ is the velocity vector of a particle,
- $\vec{\Delta} \in \mathbb{R}^2$ is the delta vector between two particle positions,
- $n \in \mathbb{Z}^+$ is the particle count,
- $r \in \mathbb{R}^+$ is the resistance coefficient,
- $s \in \mathbb{R}^+$ is the acting range,
- $\mathbf{A} \in \mathbb{M}$ is the particle attraction matrix,
- $\mathbf{t} \in \mathbb{Z}^+$ is the particle type array.

B. Particle Motion Law Pseudocode

Algorithm 1: Particle Motion Law

for *each particle* p_1 **do**
  for *each particle* $p_1 \neq p_2$ **do**
    $\text{xDelta} \leftarrow \text{p}_2\text{.pos.x} - \text{p}_1\text{.pos.x}$;
    $\text{yDelta} \leftarrow \text{p}_2\text{.pos.y} - \text{p}_1\text{.pos.y}$;
    $\text{dist} \leftarrow \sqrt{\text{xDetla}^2 + \text{yDelta}^2}$;
    if $dist \leq 4$ **then**
      if $dist \leq innerRange$ **then**
        $\text{force} \leftarrow 1 - \frac{\text{innerRange}}{\text{dist}}$;
      else
        $\text{force} \leftarrow \text{attractions}[\text{p}_1\text{.type}][\text{p}_2\text{.type}] \cdot \frac{\text{dist} - \text{innerRange}}{2}$;
      $\text{p}_1\text{.vel.x} \leftarrow \text{p}_1\text{.vel.x} + \text{force} \cdot \frac{\text{xDetla}}{\text{dist}} \cdot \text{invResistance}$;
      $\text{p}_1\text{.vel.y} \leftarrow \text{p}_1\text{.vel.y} + \text{force} \cdot \frac{\text{yDelta}}{\text{dist}} \cdot \text{invResistance}$;
for *each particle* p **do**
  $\text{p.pos.x} \leftarrow p\text{.pos.x} + \text{step} \cdot \text{p.vel.x}$;
  $\text{p.pos.y} \leftarrow p\text{.pos.y} + \text{step} \cdot \text{p.vel.y}$;

C. Emerged Organisms



D. MoSCoW Statement

D.1. Must Have

- Particle motion law
- Particle interaction parameter manipulation
- Variable simulation step coefficient
- Scene pan and zoom
- Deterministic outcome
- Logged statistics of particle type positions and density at a given time

D.2. Should Have

- Taurus scene
- Variable scene size
- Spatial hash map collision optimisation
- Bespoke thread pool implementation
- Batched particle rendering
- Variable particle count

D.3. Could Have

- Over/under population and reproduction rules
- User interaction to add, delete and move particles
- Draw particles into scene
- Launcher GUI for parameter manipulation
- Video recording of simulation

D.4. Won't have

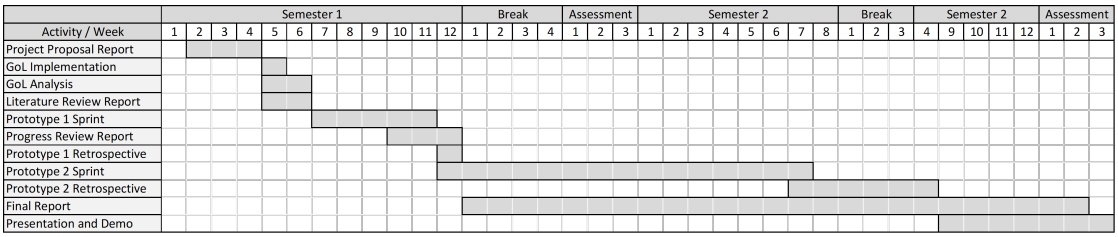
- Delta time coefficient
- 3D implementation
- Hardware acceleration

E. Prototype Feature Tables

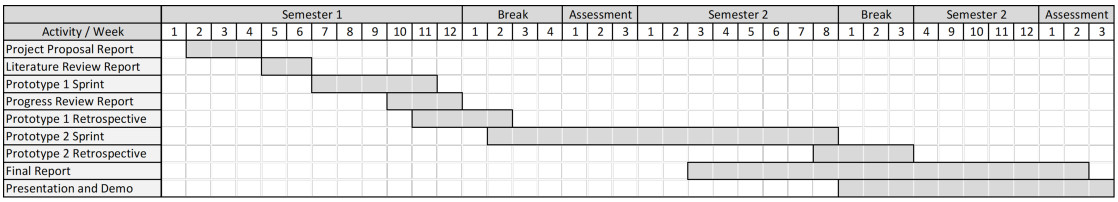
Table 1: Prototype Features

Feature	Implementation Status
Prototype 1	
Particle Potion Law V1	Full
Particle Type Interaction Manipulation	Full
Bounds handling - Bounce on Bounds	Full
Screen Space Rendered Manipulation	Full
Algorithmic Optimisation	Full
Spatial Hash	Partial
Particle Data collection	None
Under/Over Population and Reproduction	Failed
Prototype 2	
Particle Potion Law V2	N/A
Interactive Particle type Interaction Manipulation	N/A
Bounds Handling - Taurus	N/A
Screen Space Rendered Manipulation	N/A
Launcher GUI	N/A
Algorithmic Optimisation V2	N/A
Spatial Hash	N/A
Thread Pool	N/A
Particle Data Collection	N/A
Particle Data Analysis	N/A

F. Original Gannt Chart



G. Updated Gannt Chart



References

Cedilnik, A., Hoffman, B., King, B., Martin, K., and Neundorf, A. (2023). Cmake. *https://cmake.org*.

educ8s Nick Koumaris and frotzen Elwin Beall (2023). Raylib-cpp-starter-template-for-vscode-v2. *https://github.com/educ8s/Raylib-CPP-Starter-Template-for-VSCODE-V2/commits/main*.

Gardner, M. (1970). The fantastic combinations of john conway's new solitaire game "life" by martin gardner. *Scientific American*, 223:120–123.

ISO (2018). Iso/iec 9899:2018 information technology programming languages c. *https://www.iso.org/standard/74528.html*.

ISO (2020). Iso/iec 14882:2020 programming languages c++. *https://www.iso.org/standard/79358.html*.

KhronosGroup (2023). Opengl. *https://www.opengl.org*.

MacDonald, T. (2009). Spatial hashing. *https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/spatial-hashing-r2697/*.

Microsoft (2023a). vscode. *https://code.visualstudio.com*.

Microsoft (2023b). Windows 11. *https://www.microsoft.com/en-gb/windows*.

Mohr, T. (2023). Particle life. *https://particle-life.com*.

Preston-Werner, T., Wanstrath, C., Hyett, P. J., and Chacon, S. (2023). github.com. *github.com*.

raysan5 (2023). raylib. *https://www.raylib.com*.

Schmickl, T. and Stefanec, M. (2019). A primordial particle system in three dimensions.

Schmickl, T., Stefanec, M., and Crailsheim, K. (2016). How a life-like system emerges from a simplistic particle motion law. *https://www.nature.com/articles/srep37969*.

Schwaber, K. (1995). Scrum development process. *https://link.springer.com/chapter/10.1007/978-1-4471-0947-1_11*.

Torvalds, L. (2023). Git –local-branching-on-the-cheap. *https://git-scm.com*.