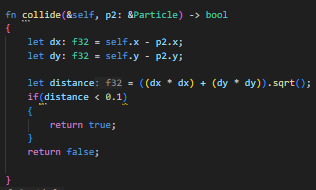
# Week 2 - Rust Lab G:

## Q1. Colliding particles

### Solution:

Collide checks if two particles are below a certain distance apart:







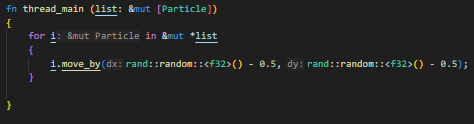
My simulation will be like so:

1. Move particles with the threadpool
2. Check collisions with the threadpool
3. Repeat

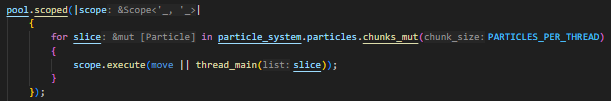
To move:

A subset of the particles is borrowed and moved, distributed between the threadpool:

Moves each particle:



Chunks the particle list between the threadpool



To check collisions:

It takes an immutable read only reference to the whole particles list

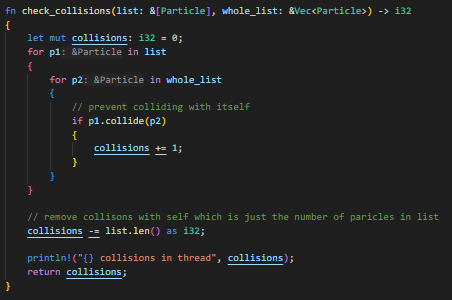
And an immutable subset of the particles.



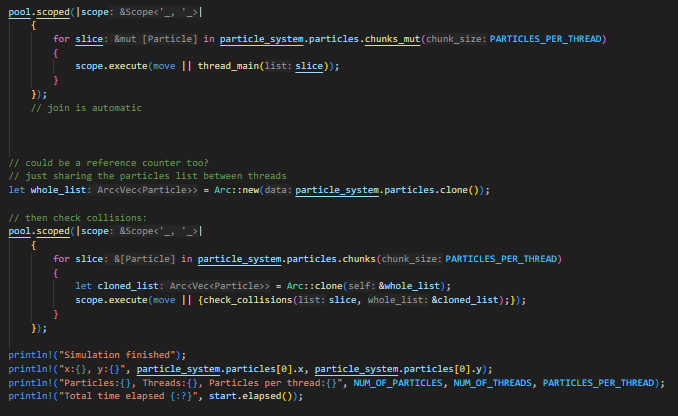
And checks if each particle pair collides.

It subtracts the self-collisions which we know is just the number of particles we check.

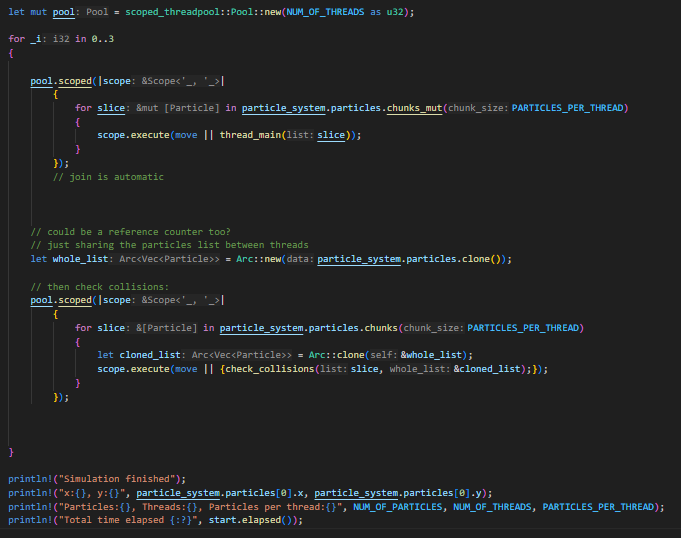
It also prints the collisions for now.



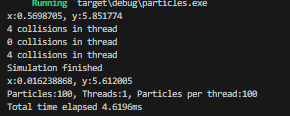
Whole code:



Make it loop:

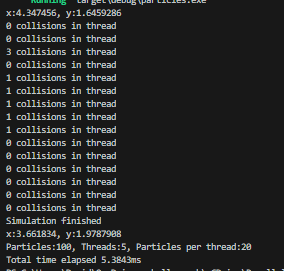


For 1 thread, 3 iterations:

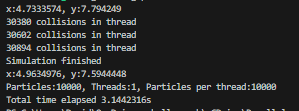


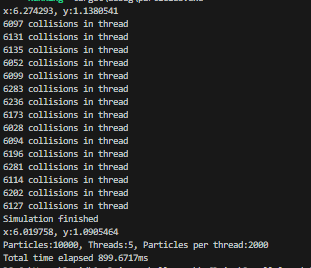
For more threads the work is split.

It takes longer probably to spawn so many threads.



With many more particles parallel is noticeably better:





### Reflection:

There are no locks needed in this solution since we do not do moving and collision checking at the same time. Instead each step is parallelised.

There isn’t any race condition for the same reason.

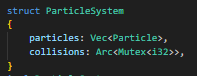
I used the same thread pool since the solution doesn’t do them at the same time.

Copying the arc saves copying the whole particle list each time.

It could be faster if we could move and check collisions at the same time but it would require careful and complicated management of the data.

## Q2. Recording collisions using Atomic

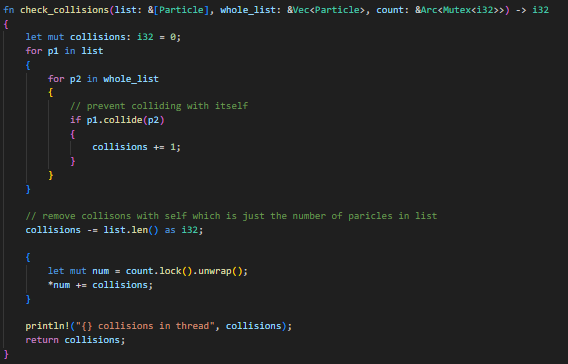
I include an atomic reference to integer guarded by a mutex:



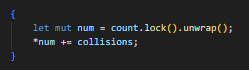
I passed the Mutex as a reference to the check\_collisions function



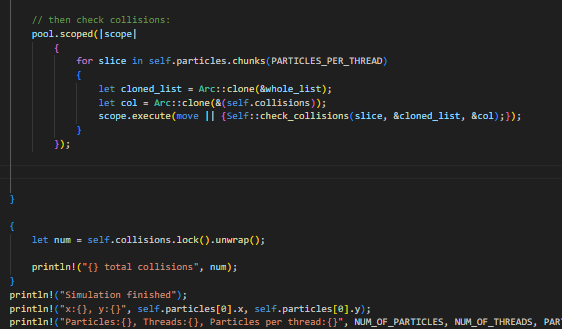
It is the same as before:

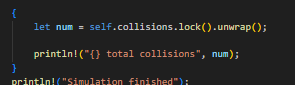


But now we lock the mutex and accumulate the count after counting all the collisions.

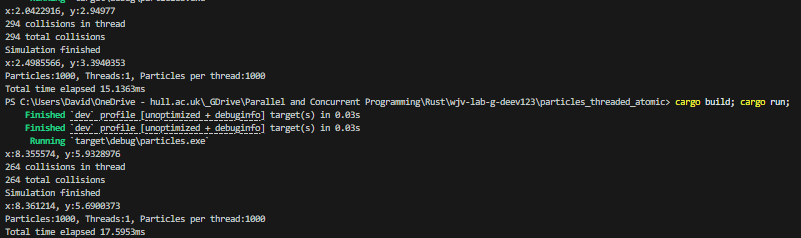


After all iterations we print the count:

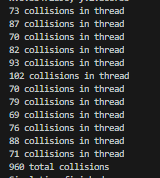


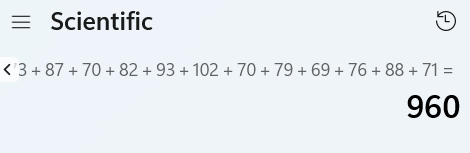


With one thread and iterations the counts match:



And with many the sum is right:





### Reflection:

An Arc and mutex can be used to accumulate the count of the collisions.

We’ve made an atomic counter in the particleSystem class.

## Q3. Ownership (optional)

Cant think of a way to do moving and collisons at the same time with continuous thread pools.

Maybe if all individual particles were locked when they were read but we still need to sync the move threads and collision threads between frames somehow so that the whole threadpool finishes and allows the other threadpool to continue if it already finished the frame before the other was finished. Something like a shared lock than stops progress to the next thread