

# 2D Brownian Motion Simulation: Mathematical Foundations and Implementation

Andrea Simone Costa

October 11, 2025

## 1 Introduction

Brownian motion, first observed by botanist Robert Brown in 1827, describes the random movement of particles suspended in a fluid. This phenomenon arises from collisions between the suspended particles and the fast-moving molecules of the surrounding medium. This project implements a computational simulation of 2D Brownian motion, demonstrating key physical principles such as random walks, diffusion, and Mean Squared Displacement (MSD) behavior.

The simulation manages 4000 particles performing random walks in a bounded 2D domain, with optimized collision detection and real-time visualization of both particle trajectories and statistical properties.

## 2 Mathematical Background

### 2.1 Random Walk

A random walk is a mathematical formalization of a path consisting of a succession of random steps. In 2D, at each time step  $t$ , a particle moves by a fixed distance  $\Delta r$  (step size) in a random direction  $\theta \in [0, 2\pi)$ . The displacement at each step is:

$$\begin{cases} \Delta x = \Delta r \cos \theta \\ \Delta y = \Delta r \sin \theta \end{cases} \quad (1)$$

where  $\theta$  is uniformly distributed:  $\theta \sim \mathcal{U}[0, 2\pi)$ .

The position of particle  $i$  at time  $t$  is obtained by accumulating all previous displacements:

$$\mathbf{r}_i(t) = \mathbf{r}_i(0) + \sum_{k=1}^t \Delta \mathbf{r}_k \quad (2)$$

### 2.2 Mean Squared Displacement

The Mean Squared Displacement (MSD) is a key observable in diffusion theory. It measures the average squared distance traveled by particles from their initial positions:

$$\text{MSD}(t) = \langle [\mathbf{r}(t) - \mathbf{r}(0)]^2 \rangle = \frac{1}{N} \sum_{i=1}^N [(x_i(t) - x_i(0))^2 + (y_i(t) - y_i(0))^2] \quad (3)$$

where  $N$  is the total number of particles and  $\langle \cdot \rangle$  denotes ensemble averaging.

For ideal 2D Brownian motion in an unbounded domain, the MSD grows linearly with time:

$$\text{MSD}(t) = 4Dt \quad (4)$$

where  $D$  is the diffusion coefficient, related to the step size and time interval by  $D = \frac{\Delta r^2}{4\Delta t}$ .

## 2.3 Boundary Effects and MSD Plateau

In a bounded domain with reflective boundaries, the MSD cannot grow indefinitely. After sufficient time, particles explore the entire available space, and the MSD reaches a plateau. The plateau value depends on the initial particle distribution:

- **Center initialization:** Particles start near the center with radial distribution. The plateau approaches  $\text{MSD}_\infty \approx L^2/6$ , where  $L$  is the domain characteristic length.
- **Random initialization:** Particles start uniformly distributed. The plateau approaches  $\text{MSD}_\infty \approx L^2/3$ .

The factor-of-two difference arises because randomly initialized particles are already spatially distributed, while center-initialized particles must diffuse outward first.

## 2.4 Collision Physics

Real Brownian particles experience collisions with both fluid molecules (microscopic) and other suspended particles (macroscopic). In this simulation, we implement a simplified collision model:

When two particles approach within distance  $d < r_1 + r_2$  (where  $r_1, r_2$  are particle radii), a collision is detected. The collision response uses a simplified bounce mechanism:

$$\phi_{\text{bounce}} = \phi_{\text{collision}} + \pi \quad (5)$$

where  $\phi_{\text{collision}} = \arctan 2(\Delta y, \Delta x)$  is the angle from the moving particle to the colliding particle. This results in a 180-degree reversal from the collision direction.

Note that this is not a physically accurate elastic collision (which would require momentum and energy conservation), but it effectively prevents particle clustering while maintaining computational efficiency for large particle systems.

## 3 Implementation

### 3.1 Architecture Overview

The codebase is structured into modular TypeScript files:

- `brownianModel.ts`: Main simulation controller extending AgentScript's `Model` class
- `collisions.ts`: Spatial grid system for optimized collision detection
- `msd.ts`: MSD calculation and Chart.js visualization
- `simulation.ts`: Canvas-based particle rendering
- `index.ts`: Application entry point and animation loop

### 3.2 Spatial Grid Optimization

A naive collision detection algorithm checks all particle pairs, resulting in  $O(N^2)$  complexity. For  $N = 4000$  particles, this means  $\sim 8$  million distance calculations per time step, which is computationally prohibitive.

The simulation implements a spatial grid to reduce this complexity. The domain is divided into cells of size  $2r$  (twice the particle radius). Each particle is assigned to the cell containing its position. To detect collisions for a given particle, we only check particles in the same cell and the 8 neighboring cells (a  $3 \times 3$  grid).

This reduces complexity to  $O(N \times k)$ , where  $k \approx 9$  cells is constant, independent of  $N$ . For evenly distributed particles, this provides approximately a 400-fold speedup for 4000 particles.

The spatial grid is implemented as a hash map:

```
Map<"cellX,cellY", Particle[]>
```

where the key is a string representation of cell coordinates and the value is an array of particles in that cell.

### 3.3 Boundary Conditions

Reflective boundary conditions are implemented at the domain edges. When a particle crosses a boundary, its position is reflected back:

$$x_{\text{new}} = x_{\text{boundary}} - (x - x_{\text{boundary}}) \quad (6)$$

This preserves the distance from the boundary, implementing a perfect mirror reflection. This approach ensures particles remain within the simulation domain while maintaining realistic diffusive behavior.

### 3.4 Rendering and Performance

The simulation renders particles using HTML5 Canvas at 30 frames per second. A centered coordinate system is used: the canvas context is translated so that  $(0,0)$  corresponds to the center of the canvas, with world bounds extending from  $-L/2$  to  $+L/2$  in both dimensions.

The MSD chart is updated every 10 ticks (not every frame) to reduce computational overhead while maintaining sufficient temporal resolution for visualizing diffusion dynamics.

## 4 Software Architecture

### 4.1 System Overview

The application follows a modular architecture with clear separation of concerns:

```
index.ts (Entry Point)
|
v
BrownianModel (AgentScript Model)
|-- turtles.create() -> Particle initialization
|-- step() -> Main simulation loop
|
+-- collisions.ts -> Spatial grid & collision detection
|
+-- msd.ts (MSDChart) -> Chart.js visualization
|
+-- simulation.ts (Simulation) -> Canvas rendering
```

### 4.2 Core Modules

**BrownianModel** (`brownianModel.ts`): Extends AgentScript's `Model` class to manage the simulation state. Uses `turtles.create()` to instantiate 4000 particle agents, each storing position  $(x, y)$ , step size, and initial position for MSD tracking. The `step()` method orchestrates the simulation loop.

**Spatial Grid** (`collisions.ts`): Implements `createSpatialGrid()` which builds a hash map of cell coordinates to particle arrays. `moveParticleWithOptimizedCollisions()` performs random walk movement, queries the grid for nearby particles, and handles collision response.

**MSDChart** (`msd.ts`): Wraps Chart.js to visualize MSD over time. Calculates  $MSD(t) = (1/N) \sum_i [(x_i - x_{i,0})^2 + (y_i - y_{i,0})^2]$  every 10 ticks and updates the chart canvas.

**Simulation** (`simulation.ts`): Manages HTML5 Canvas rendering. Uses `ctx.translate()` to center the coordinate system, then iterates through particles with `turtles.ask()` to draw each as a circle using `ctx.arc()`.

### 4.3 Execution Flow

Each animation frame (30 FPS via `requestAnimationFrame`):

1. **Grid Construction:** `createSpatialGrid()` assigns each particle to a cell based on  $[x/\text{cellSize}]$
2. **Movement Loop:** For each particle, `turtles.ask()` calls `moveParticleWithOptimizedCollisions()`:
  - Generate random angle  $\theta \sim \mathcal{U}[0, 2\pi]$
  - Calculate displacement  $(\Delta x, \Delta y) = (\text{stepSize} \cos \theta, \text{stepSize} \sin \theta)$
  - Query  $3 \times 3$  cell neighborhood for collision candidates
  - If collision detected: reverse direction by  $\pi$  radians
  - Apply reflective boundary conditions
  - Update position via `turtle.setxy()`
3. **MSD Update:** Every 10 ticks, calculate MSD and push to Chart.js dataset
4. **Rendering:** Clear canvas, apply coordinate transform, draw all particles as circles

### 4.4 AgentScript Integration

AgentScript provides the agent-based modeling framework. Key usage patterns:

- Model class: Base class providing world bounds, turtles collection, and tick counter
- `turtles.create(n, initFn)`: Batch-create particles with initialization function
- `turtles.ask(fn)`: Iterate over all particles, applying function to each
- `turtle.setxy(x, y)`: Update particle position (handles coordinate clamping)

### 4.5 Canvas Rendering

The Canvas 2D API renders particles at each frame:

```
ctx.save()
ctx.translate(canvas.width/2, canvas.height/2) // Center origin
turtles.ask(turtle => {
  ctx.beginPath()
  ctx.arc(turtle.x, turtle.y, turtle.size, 0, 2*Math.PI)
  ctx.fillStyle = turtle.color
  ctx.fill()
})
ctx.restore()
```

This coordinate transform maps physics space (centered at origin) to screen space (top-left origin).

## 5 Results and Discussion

The simulation successfully demonstrates key features of 2D Brownian motion:

1. **Linear MSD growth:** In the initial phase, before boundary effects dominate, the MSD grows linearly with time, consistent with the theoretical prediction  $\text{MSD}(t) = 4Dt$ .
2. **Plateau behavior:** After sufficient time, the MSD reaches a plateau whose value depends on the initialization strategy, confirming the theoretical expectations ( $L^2/6$  for center initialization,  $L^2/3$  for random initialization).
3. **Collision effects:** The simplified collision model prevents particle clustering while maintaining the random walk character of the motion.
4. **Computational efficiency:** The spatial grid optimization enables real-time simulation of 4000 particles with collision detection at 30 FPS.

### 5.1 Limitations and Extensions

The current implementation uses a simplified collision model that does not conserve momentum or energy. A more physically accurate simulation would implement elastic collisions with velocity memory, allowing particles to maintain momentum between steps.

Additionally, the simulation uses discrete time steps and fixed step sizes, while real Brownian motion is a continuous-time process. A more sophisticated model could implement the Langevin equation or solve the stochastic differential equations governing Brownian dynamics.

## 6 Conclusion

This project provides an accessible yet rigorous computational implementation of 2D Brownian motion. By combining agent-based modeling, optimized collision detection, and real-time visualization, it offers an interactive tool for understanding fundamental concepts in statistical mechanics and diffusion theory. The implementation balances physical realism with computational efficiency, making it suitable for both educational purposes and quantitative analysis of diffusion phenomena.