N-Body Simulation

- Applications to astrophysics.
  - Orbits of solar system bodies.
  - Stellar dynamics at the galactic center.
  - Stellar dynamics in a globular cluster.
  - Stellar dynamics during the collision of two galaxies.
  - Formation of structure in the universe.
  - Dynamics of galaxies during cluster formation.

N-Body Simulation

1. Setup initial distribution of particles.
   - Need accurate data and model of mass distribution.

2. Compute forces between particles.
   - Direct sum: $N^2$.
   - 
   \[ F_i = \sum_{j \neq i} \frac{Gm_i m_j}{|r_i - r_j|^2 + \epsilon^2} \]
   - $\epsilon$ - softening parameter.
   - Eliminates binary stars with $|r_i - r_j| < \epsilon$.
   - Hard binaries can be important.

3. Evolve particles using ODE solver.
   - Leapfrog method balances efficiency and accuracy.

4. Display and analyze results.

\[
\begin{align*}
\frac{dx_i}{dt} &= v_i \\
\frac{dv_i}{dt} &= F_i 
\end{align*}
\]

If Statement Examples

| absolute value | if (x < 0) x = -x; |
| put x and y into sorted order | if (x > y) |
| maximum of x and y | if (x > y) max = y; |
| error check for division operation | System.out.println("Division by zero"); |
| error check for quadratic formula | double discriminant = b*b - 4.0*c; |

While Loop

- The while loop. A common repetition structure.
  - Check a boolean expression.
  - Execute a sequence of statements.
  - Repeat.
For Loops

- The for loop. Another common repetition structure.
  - Execute initialization statement.
  - Check boolean expression.
  - Execute sequence of statements.
  - Execute increment statement.
  - Repeat.

```
for (init; boolean expression; increment) {
  statement 1;
  statement 2;
  ...
}
```

Loop Examples

```
int v = 3;
for (int i = 0; i <= N; i++)
  System.out.println(1 + " + v);

int v = 2 * v;
```

```
int sum = 0;
for (int i = 1; i <= N; i++)
  sum += i;
System.out.println(sum);
```

```
int product = 1;
for (int i = 1; i <= N; i++)
  product *= i;
System.out.println(product);
```

While Loops: Powers of Two

- Ex. Print first n powers of 2.
  - Increment i from 1 to n.
  - Double v each time.

```
int i = 0;
int v = 1;
while (i <= N) {
  System.out.println(v);
  i = i + 1;
  v = 2 * v;
}
```

```
Console:
1
2
4
8
16
32
64
```
Problem 2

- **Square Root:**
  - Given a real number \( c \) and some error tolerance \( \epsilon \)
  - Estimate \( t \), the square root of \( c \)

While Loops: Square Root

- **Q.** How might we implement `Math.sqrt()`?
- **A.** To compute the square root of \( c \):
  - Initialize \( t_0 = c \).
  - Repeat until \( t_i = c / t_i \) up to desired precision:
    - set \( t_{i+1} \) to be the average of \( t_i \) and \( c / t_i \).

    \[
    \begin{align*}
    t_0 &= 2.0 \\
    t_1 &= \frac{1}{2}(t_0 + \frac{c}{t_0}) = 1.5 \\
    t_2 &= \frac{1}{2}(t_1 + \frac{c}{t_1}) = 1.416666666666665 \\
    t_3 &= \frac{1}{2}(t_2 + \frac{c}{t_2}) = 1.4142156862745097 \\
    t_4 &= \frac{1}{2}(t_3 + \frac{c}{t_3}) = 1.4142135623746899 \\
    t_5 &= \frac{1}{2}(t_4 + \frac{c}{t_4}) = 1.414213562373095
    \end{align*}
    \]

    computing the square root of 2

Problem 3

- **Gambler's ruin.** Gambler starts with $\text{stake}$ and places $1$ fair bets until going broke or reaching $\text{goal}$.
  - What are the chances of winning?
  - How many bets will it take?

Gambler's Ruin

- **One approach.** Monte Carlo simulation.
  - Flip digital coins and see what happens.
    - Pseudorandom number generation
    - `java.util.Random`
  - Repeat and compute statistics.

![Gambler's Ruin Diagram](image-url)