2-Dimensional Random Walk Simulation with Characters

Zhouyang Wen*
Graduate School of Frountier Science,
The University of Tokyo
(Dated: January 28, 2024)

This report studies the Random Walk process of one or more objects with certain rules in a 2-dimensional map. Initially, the behavior of a single object, which can move randomly in four directions (along the X-axis or Y-axis), is investigated. Then the simple 2-D Random Walk in a bounded map of three different objects/characters, each has certain rules of motion once encounters each other, is simulated. Finally, the objects are given the ability to move in all vectorized directions and their Random Walk processes are studied and discussed.

I. INTRODUCTION

In natural science research, theoretical studies and computer simulations of Random Walk processes have been successfully applied in multiple disciplines such as mathematics, physics, chemistry and economics. Random walk is the basis of the diffusion process and is widely used in the simulation of the motion of particles as they propagate through liquids or gases, the search paths of foraging animals, fluctuation of stock prices, etc.

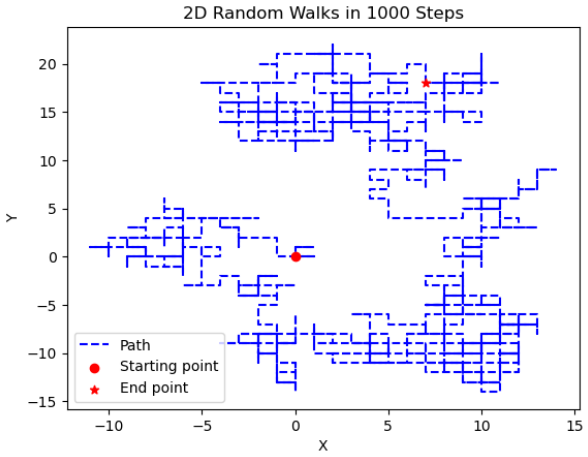
In this report, I use Python's random number generator function in combination with Arrays to simulate the Random Walks of one or multiple objects on a 2-dimensional map.

II. SIMULATION RESULTS

A. Simple 2-D Random Walk

In the first part, I tried to simulate the Random Walk of one object moving freely in a 2-D map. This is achieved by designing a function plot_random_walks(num_steps), which takes the number of steps as the input and the object will then choose randomly (with equal probabilities) from four directions: either along X-axis or Y-axis (up, down, left, right with respect to eye observation) and moves in 1 unit-length per step. By giving the object 1000 steps, its path is shown as figure 1a.

As the number of steps increased to 100000 as shown in figure 1b, it can be observed that the object's path is more spread out and covers more area.



(a) 1000 steps

(b) 100000 steps

FIG. 1: Simple 2-D Random Walk simulation of one object starting from origin

B. Simple 2-D Random Walk with Characters

In this part, three characters, Villager, Hunter and Monster, were introduced for simulation. Each of the characters have certain initial and boundary conditions, basic stats, and rules of motion, such as:

- Initial and boundary conditions
  1. Map is in 50 by 50 size and rectangular shape.
  2. Villager and Hunter start from origin. Monster will appear in random position.
- Basic stats

* 4783396298@edu.k.u-tokyo.ac.jp

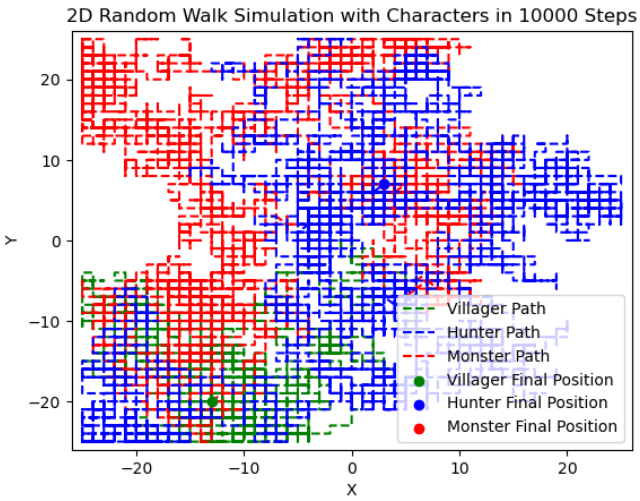


FIG. 2: Simple 2-D Random Walk simulation of 3 characters in 10000 steps in a bounded map.

1. All Villager, Monster and Hunter will perform random walk inside the map at first at a speed of 1 unit-length per move.
2. They all have 2 unit-lengths of vision distance.
3. Monster and Hunter have 1 unit-length of attack range and Hunter will always win against Monster.
4. They have different agilities (move orders): Villager moves the first, Monster the second, and Hunter moves last.

- Rules of motion upon encounter

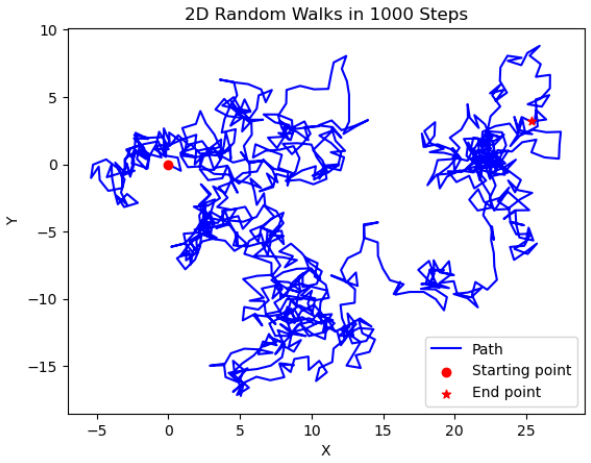
1. Villager will escape from Monster.
2. Monster will chase the closer one of both Villager and Hunter.
3. Once Monster hunts down the target it will continue random walk.
4. Hunter will chase and hunt Monster.
5. The motion will end when Monster is eliminated or the movement limit is reached.

Based on the set of rules, the simulation is shown as figure 2. In this simulation, Villager was caught by Monster at step 2249 and Monster was eliminated by Hunter at step 4501 and ended the simulation.

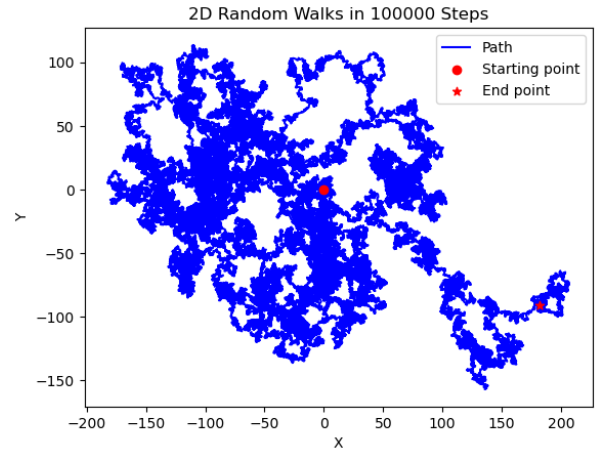
However, at given 10000 steps in a 50 by 50 map, there are still high chances where Villager and Monster are both alive at the end.

C. 2-D Vectorized Random Walk

In this part, the Random Walk simulation is not limited to only four directions, but all possible vectorized directions as shown in figure 3.



(a) 1000 steps



(b) 100000 steps

FIG. 3: 2-D vectorized Random Walk simulation of one object starting from origin.

With the freedom to move in any direction, the variability in the paths increases significantly. Each step can lead to a unique angle, resulting in a more complex path and the path appears smoother.

The 2-D Random Walk with characters is also vectorized as shown in figure 4. In this simulation, Villager was caught by Monster at step 7858 and Monster was eliminated by Hunter at step 9031 and ended the simulation. Still, there are high probabilities where Villager and Monster are both alive at the end of each simulation.

III. DISCUSSION

In the two-dimensional case, the object can only move in four directions (up, down, left, and right) at each moment. Denoting $P(i, j, n)$ as the probability that the object appears at the point (i, j) at the nth step. The cor-

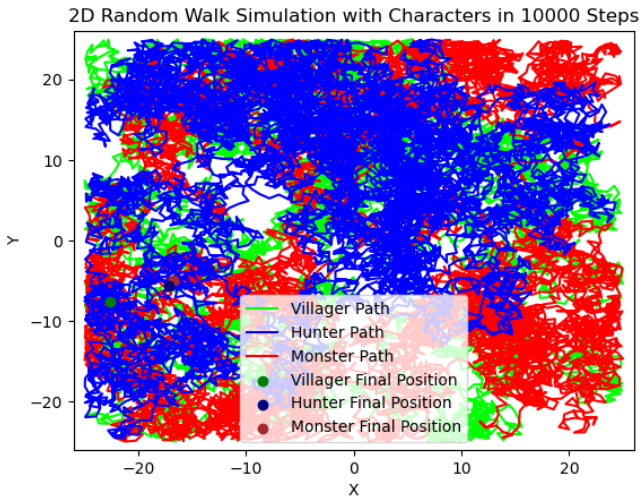


FIG. 4: 2-D vectorized Random Walk simulation of 3 characters in 10000 steps in a bounded map.

responding recurrence relation is as follows:

$$P(i,j,n) = \frac{1}{4}\Big[P(i+1,j,n-1) + P(i-1,j,n-1) + P(i,j+1,n-1) + P(i,j-1,n-1)\Big].$$

It can be written in difference equation to express the change in particle density over time:

$$\begin{aligned} &P(i,j,n) - P(i,j,n-1) \\ &= \frac{1}{4}\Big[P(i+1,j,n-1) + P(i-1,j,n-1) \\ &\quad - 2P(i,j,n-1) + P(i,j+1,n-1) \\ &\quad + P(i,j-1,n-1) - 2P(i,j,n-1)\Big]. \end{aligned}$$

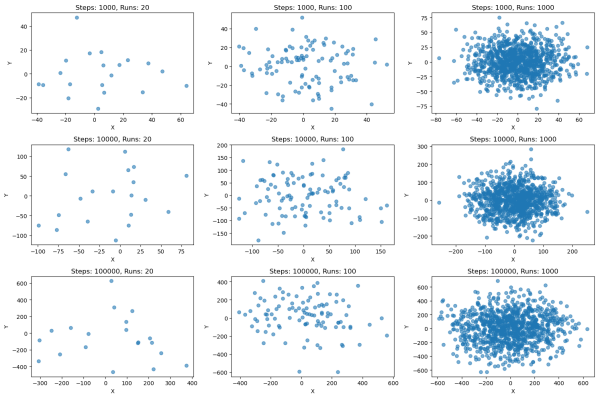
When the time and displacement is very small $\Delta t \to 0$ and $\Delta x \to 0$, the above difference equations can be transformed into partial differential equations, i.e., two-dimensional diffusion equations:

$$\frac{\partial P(x,y,t)}{\partial t} = D\nabla^2 P(x,y,t)$$

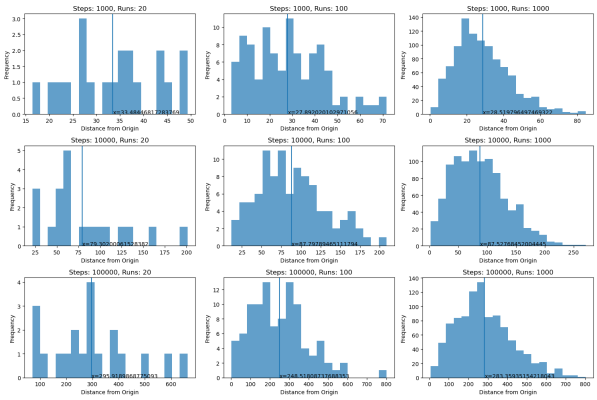
in which ∇^2 is the 2-D Laplacian and D is the diffusion coefficient. This equation describes the diffusion process of an object over time in 2-D spaces.

According to the law of large numbers and Central Limit Theorem (CLT), when a large number of independent random variables are added together, their normalized sum tends to be a Gaussian distribution, regardless of the distribution of the original variables. In the case of Random Walk, the position of an object is the sum of many independent random steps, so that over time, its position distribution tends to be a Gaussian distribution.

Let $\mathbf{R}$ be the distance vector from start to end of a Random Walk process of fixed step length $\mathbf{r}_i = \mathrm{l}$. $\mathbf{R}$ can



(a) final positions



(b) distributions

FIG. 5: The final positions and their distributions for 20, 100 and 1000 runs with 1000, 10000 and 100000 steps.

then be expressed as $\mathbf{R} = \sum_{i=1}^{N} \mathbf{r}_i$, where $\mathbf{r}_i$ is the vector of the i-th step. The Root-Mean-Square of the distance vector is given by

$$\begin{aligned} \mathrm{RMS} &= \sqrt{\langle R^2 \rangle} = \sqrt{\langle \mathbf{R} \cdot \mathbf{R} \rangle} \\ &= \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} \langle \mathbf{r}_i \cdot \mathbf{r}_j \rangle} = \sqrt{Nl^2} = l\sqrt{N}. \end{aligned}$$

Let Z_i denote the i-th coordinate of $\mathbf{R}$ after N steps, the expected value of

$$\langle R \rangle = \sqrt{\langle R^2 \rangle} = \sqrt{\sum_{i=1}^{d} Z_i^2}.$$

The square root of a sum of k independent random variables is distributed according to the chi distribution, χ_k. Therefore $\sqrt{\sum_{i=1}^{d} dZ_i^2/l^2 N}$ is approximately χ_d-distributed for large values of N, and its expected value is $\sqrt{2}\Gamma(\frac{d+1}{2})/\Gamma(\frac{d}{2})$.

Hence, the overall expected value for the distance vec-

tor from start to end can be expressed as

$$\langle R\rangle = \left\langle l\sqrt{\frac{N}{d}\sum_{i=1}^{d}\frac{dZ_i^2}{l^2N}}\right\rangle = l\sqrt{\frac{2N}{d}}\frac{\Gamma(\frac{d+1}{2})}{\Gamma(\frac{d}{2})}.$$

For 2-D Random Walk processes with steps $N = 1000, 10000, 100000$, their expected values will be $\sim 28.02, 88.62, 280.25$ respectively.

The Monte Carlo method is very useful to find the distribution of final positions when simulating 2-D Random Walks. It can be seen from figure 5a that, as the number of runs and the number of steps in each run increase, the final positions become more spread out and their distribution approaches a Gaussian distribution, and from 5b the mean values for the distance between the final position and the starting point approaches their expected values.

IV. CONCLUSION

In conclusion, this report has shown the study of simulating the Random Walk process in 2-D spaces, has used the Monte Carlo method to explore the distribution of final positions of an object in the Random Walk process, and has found that it tends to be Gaussian distributed as the number of runs increases.

During the programming, I made a great effort to avoid hard-coding and to design the structure of the function that allows for the easy adjustment of characters with different stats (see Appendix A). For instance, I can easily change the order of movement, attack range or movement speed without the need to modify the original function, and I can change the position of map vertices to manipulate the map size and shape (figure 6). In this way a lot of variations for Random Walk can be easily observed.

There are still some ideas that I could not realize due to a very limited amount of time to work on this subject, such as exploring the Random Walk with more number of characters and different types of characters on a continuous map (as the surface of a sphere where there are no boundaries and an object moving in one direction will return to its original position). The map can be more complex by adding shelters and obstacles and having limited amount of resources or even renewable resources.

Appendix A: Example of the input for simulation and its output

```
villager = {
    'agility' : 0,     # order of movement
    'attack'  : 1,     # character with higher
        attack value will win
    'movement': 1,     # the length of every
        movement step
    'range'   : None, # attack range
    'vision'  : 2      # vision distance
}

monster = {
    'agility' : 1,
    'attack'  : 2,
    'movement': 1,
    'range'   : 1,
    'vision'  : 2
}

hunter = {
    'agility' : 2,
    'attack'  : 3,
    'movement': 1,
    'range'   : 1,
    'vision'  : 2
}

characters = {
    'villager': villager,
    'monster' : monster,
    'hunter'  : hunter
}

current_map = np.array([
    (-10, -20) , (-20, 0), (0, 20), (20, 0),
        (10, -20)
]) # Polygon map with ordered vertices

steps = 10000

plot_motion_simulation(steps, characters,
    current_map)
```

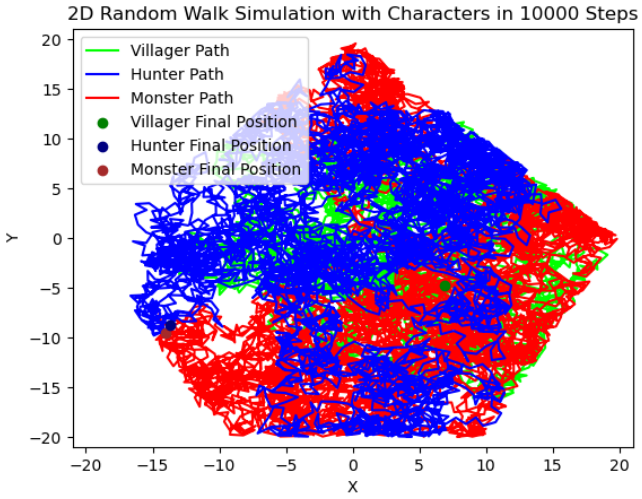


FIG. 6: An output of adjusted map shape (a pentagon).