Random walk simulation with various cases

Taehoon Kwon

April 22, 2020

Abstract. In this paper we will analyze *random walk* that has been done in the field of discrete mathematics, specifically examining the numerical estimation of *general random walk*, and *self-avoiding random walk*. We present the framework used, results and observations from the simulation controlling various conditions, and difficulties throughout the project.

1 Introduction

A *random walk* is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps on some mathematical space such as the integers. We simulate the random walk in different several conditions and provide statistics of the following measurement results :

- Numerical estimation of the distance from the origin to endpoint of random walk.
- Numerical estimation of the number of steps until a random walk returns to the origin.
- Numerical estimation of the number of steps until two random walks intersect.

An interesting question is – Will the *random walk* move in a biased direction, or will it swing the space evenly?

2 Framework

The framework of a *random walk* simulation is divided into two main panels. A simulation panel displays random walkers with spherical shape following by path lines in the *Cartesian coordinates*. Each different walkers has different color encoded. The simulation can be played, paused, stopped, rewind, and proceed with top button controls. A configuration panel has multiple options to set the initial state of the simulation. **Figure 1** shows the several controls used in the framework. **Figure 2** shows the framework simulated with 10 number of walkers.

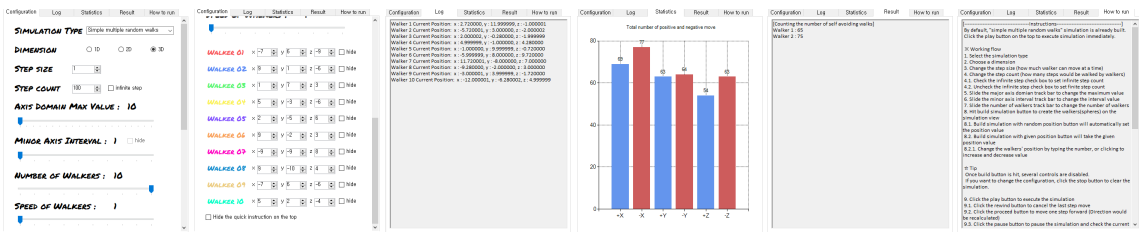


Figure 1: Various controls in configuration panel

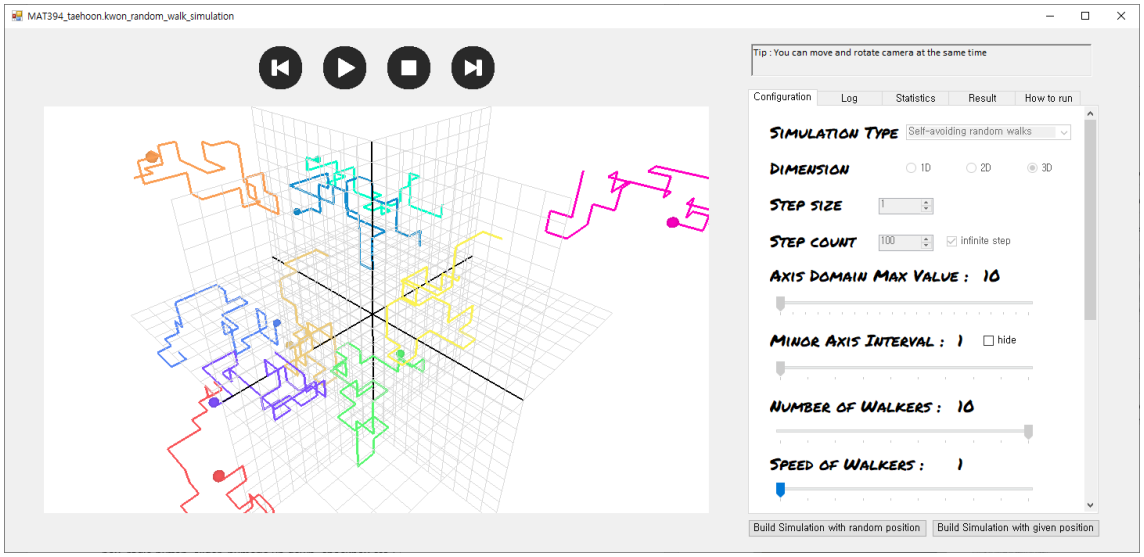
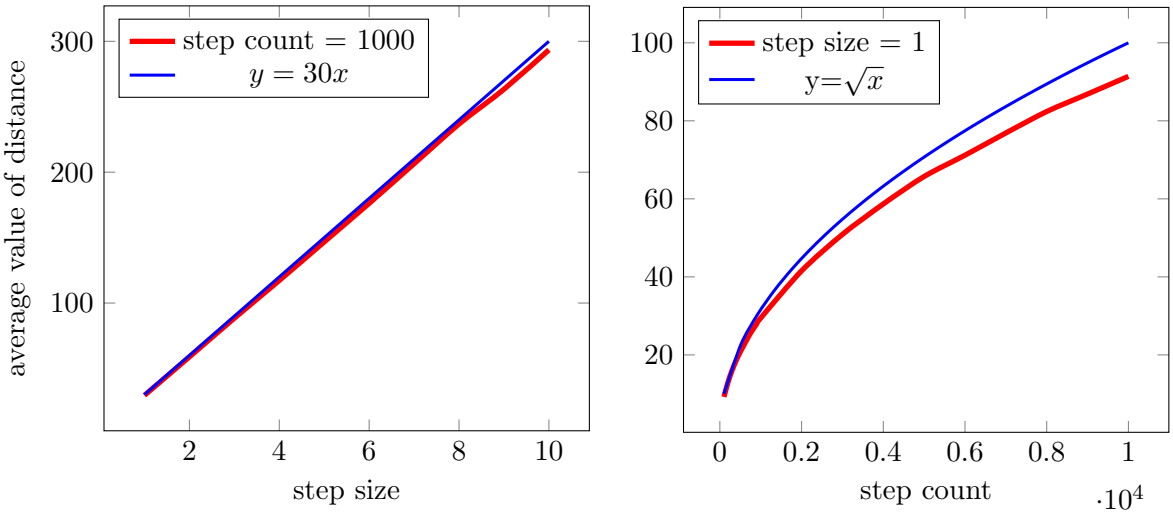


Figure 2: A random walk simulation framework

3 Results and observations

3.1 Distance from the origin to endpoint of random walk

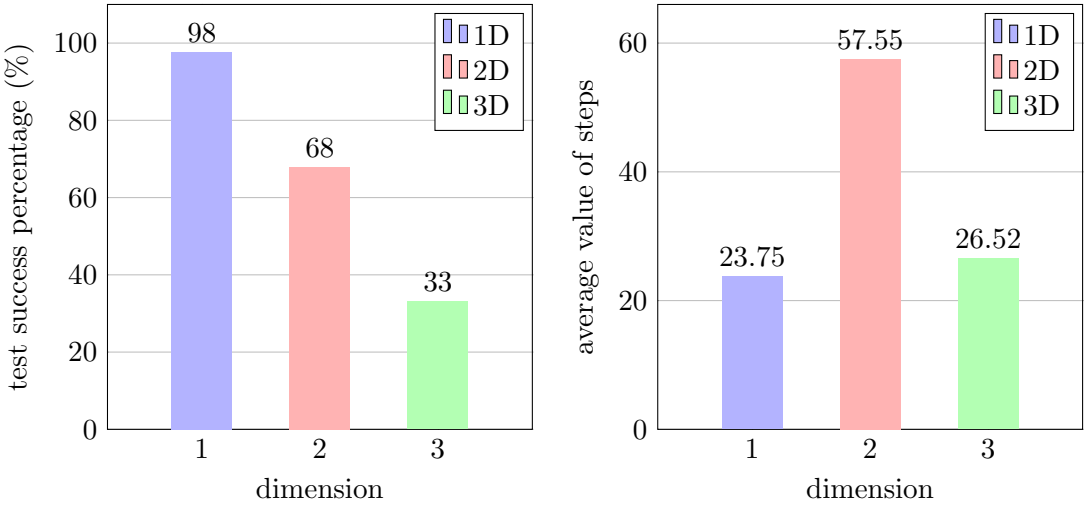
We conducted two experiments with a simple random walk algorithm in which walkers determine the next direction with $\frac{1}{6}$ probability at each stage in three dimension space. The averages of distance were calculated with 10,000 tests of simulation. In the first experiment, the manipulated variable was the step size which indicates how many walkers can go in one step within a single test, and the control variable was the step count that indicates how many times walker can decide the next direction within a single test. The step count was fixed with 1,000 times, and the step size varies from 1 to 10. In the second experiment, manipulated and control variables are swapped, and step size was fixed with 1, and the step count varies from 100 to 10,000 times.



The result of the first experiment shows that graph of the distance variation as the step size increase is close to the linear function. Second experiment result shows that the distance variation as the step count increase is close to the square root function.

3.2 Number of steps until a random walk returns to the origin

This experiment is also executed with a simple random walk algorithm. Since the simulation ended only if the walker returns to the origin, the step count is added to the termination conditions, which results in excluding the data of too many steps to return. The step count, and the number of tests were fixed with 1,000 times. The first graph below shows that the percentage of tests that the walker returned to the origin. The second graph below shows how many steps walker needed to return the origin in successful tests.



According to the first graph, in almost every test in one-dimensional space, walker returned to the origin, but only two third of tests are successful in two dimension, and one third of tests in three dimension. We could find interesting thing in the second graph. It shows that the largest number of steps are needed in two dimensions. In this experiment, there are two possible problems in this experiment. First, analyzing a set of data on the number of steps needed to return to the origin, in most cases few steps were needed, but in some cases a fairly large number of steps were needed. There is a large deviation between the tests. Second, since there is a large deviation, the real data might be omitted due to the not enough step count. In other words, the most of cases needed large number of steps for walker to return in three dimensions, but those cases are discarded. The fact that only 33% of the tests are successful tests means that it is difficult to trust the results of those tests.

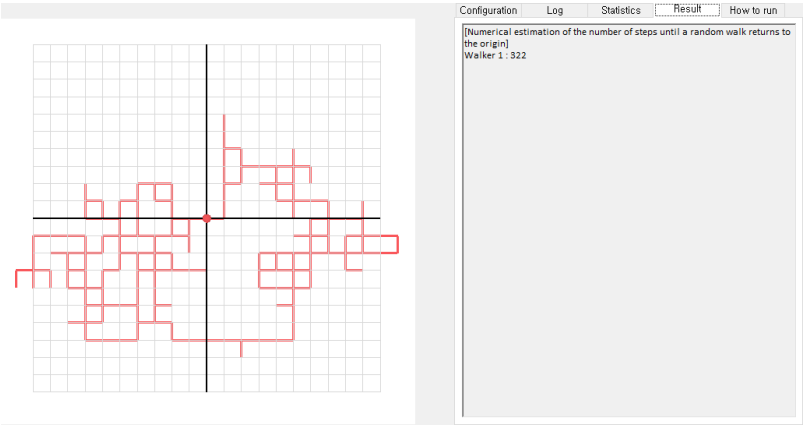


Figure 3: Numerical estimation of the number of steps until a random walk returns to the origin in two dimensional space.

3.3 Number of steps until two random walks intersect

This experiment is performed with self-avoiding random walk. Since the self-avoiding random walk in one dimensional shows the walker moves in one direction only, which cause the infinite loop. So, this case was excluded from the experiment. The step count was fixed with 1,000 times, and 1,000 tests are done for other dimensions. However, every series of tests returns a large deviation number of steps. For two dimensions, the results were from 2 to 25, and for three dimensions, the outcomes were from 3 to 6. Here are the same problems we discussed on the previous section. It might cause the meaningless results. However, increase of the step count and the number of test takes a lot of time to get the final result, which could not be done in limited time.

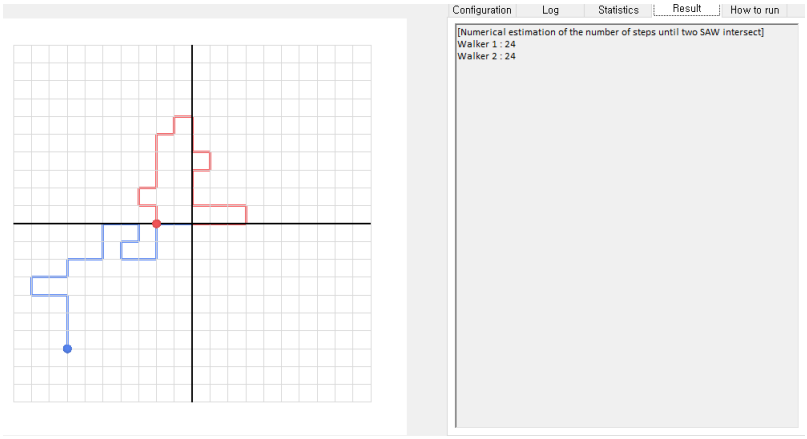


Figure 4: Numerical estimation of the number of steps until two random walks intersect in two dimensional space.

3.4 Discussable question

Back to the interesting question – Will the random walk move in a biased direction, or will it swing the space evenly? Since the pseudo random number actually is not a real random number, there may be a possibility of moving in a biased direction. However, the random walk simulation shows that walkers move evenly in different directions.

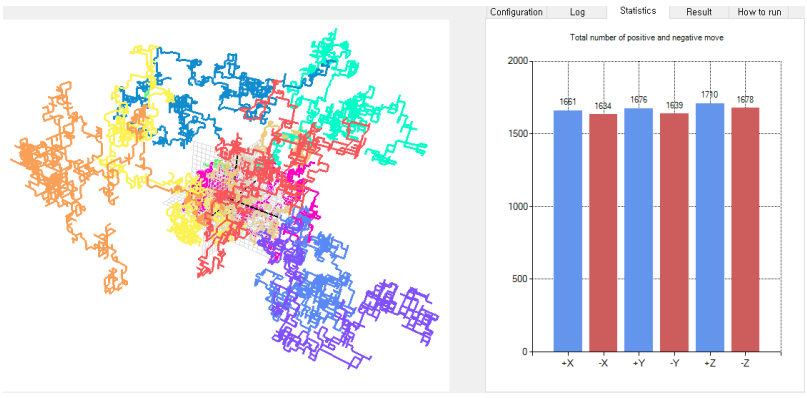


Figure 5: A simple random walk with the count of move

4 Difficulties

The difficulties throughout the project was setting up the graphic parts to make simulation work. Drawing single object on the screen requires the bunch of the code including object loader, shader manager, camera controller, model manager. Second, implementing the functionality of various controls. Since the base framework is the .NET framework, a managed type cannot be used as an arguments with the controls. Third, frequent crash in internal calls of .NET framework. Because of setting values of the controller with built-in property window, there potentially exists incompatible execution, which is difficult to catch exception.

5 References

https://en.wikipedia.org/wiki/Random_walk

https://en.wikipedia.org/wiki/Common_Language_Runtime

https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview?redirectedfrom=MSDN

https://docs.microsoft.com/ko-kr/cpp/dotnet/dotnet-programming-with-cpp-cli-visual-cpp?view=vs-2019

https://docs.microsoft.com/ko-kr/cpp/dotnet/managed-types-cpp-cli?view=vs-2019

https://docs.microsoft.com/ko-kr/cpp/dotnet/native-and-dotnet-interoperability?view=vs-2019

https://en.wikipedia.org/wiki/GLFW$