This task focuses on understanding how distributions, the law of large numbers, computational complexity, and accuracy interplay in simulating random processes, specifically cell movements on a grid. It aims to demonstrate how bias emerges and diminishes in random processes and how uniformity develops with increasing iterations. Depending on how the randomness is implemented, it could lead to a variety of distributions, such as the normal distribution, which describes data using a probability density function with mean μ and standard deviation σ and is often used in random number generation (Shiflet & Shiflet, 2014). The uniform distribution ensures that all outcomes have an equal probability in discrete cases, or that equal-length intervals have an equal chance of occurring in continuous cases (Shiflet & Shiflet, 2014). The Bernoulli distribution arises when modelling binary outcomes, and its conditional version, the conditional Bernoulli distribution, is useful when several Bernoulli random variables are conditioned on their sum equalling a specific value, with applications in sampling and hypothesis testing (Chen and Liu, 1997). The beta-binomial distribution applies when the probability of success in a binomial process itself follows a beta distribution, offering a model for over-dispersed data, common in biological and reliability studies (Skellam, 1948; Altham, 1978). Lastly, the Poisson distribution generalizes the binomial distribution by allowing different probabilities of success in each trial, which is useful in survey sampling and logistic regression (Chen and Liu, 1997). These varying distributions help explain how randomness in cell movement can lead to different patterns, and how bias and uniformity evolve as the process progresses. In practice however never can a distribution truly be split uniformly however due to the law of large numbers the outcomes will be slowly converging towards an even probability; so, when trying to find if the distribution is uniform a chi-square test can be implemented the purpose of the chi-square test is to compare the expected and observed outcomes. The chi-square test does not confirm whether the distribution is random, but it does give an indication as the chi-square test only looks at frequency a test that looks at the shape of the frequency which allows for detection of small deviation even when looking at large sample sizes, that test is the Kolmogorov-Smirnov test. When testing how the distribution was done using the numpy random function “numpy.random.uniform” which in theory should give each of the directions an equal chance of occurring. When it comes to the chi-squared test across three runs it becomes clear that the distribution is uniform with each checkpoint however when you look at the graphs (See Figure 1,2 and 3) this is clearly not the case with a bias in all three of the runs in different directions however most commonly in the “up” direction this is seen in the KS results with each of them failing the test this is due to the fact that KS is a test which is more sensitive to deviation than the chi-squared test. This bias may be due to the small amount of steps causing statistical noise to get rid of this noise large step sizes may be needed to reduce the noise as according to the law of large numbers which says as more steps are taken in the random walk the closer to converging to the mean which in this case would be 25% distribution between all four directions as at a small step count like 100 the randomness isn’t fully expressed meaning that slight deviations are more common and more detrimental. When increasing the stepsize however this increases the time complexity simply because there are more steps that it must do. The larger sample size would also give the KS and Ch2 test more chance of being accurate reading.

## Task 1\_2

When it comes to the additional directions a new way of getting directions is needed to be taken because the generation of a two binary numbers is no longer appropriate instead it is implemented as seen below.

Pseudocode