Integrating educational dialogue into a classic random walk activity

Lewis A. Baker

Faculty of Engineering and Physical Sciences, University of Surrey, Guildford, GU2 7XH, United Kingdom.

E-mail: `lewis.baker@surrey.ac.uk`

Abstract. This work reframes the classic 1-D random walk problem to provide structured opportunities for developing students' educational dialogue skills. By focusing on the rare events that can happen across 20 rounds of European roulette, students can be tasked to construct, describe and search for sequences of events across a large dataset. This enhances, rather than detracts from the mathematical and programming skills traditionally developed by modelling such problems. This work therefore seeks to emphasise the opportunities that likely already exist in curricula which can embed opportunities for students to deliberately practice their educational dialogue skills, skills that are often overlooked in traditional Science, Technology, Engineering and Mathematics (STEM) courses.

Keywords: Educational dialogue, Random walks, European Roulette, Python, Activity ideas

Submitted to: *EdArXiv Preprints*

1. Introduction

'Educational dialogue' is the deliberate use of speaking and listening skills to build and communicate ideas, critique them, and co-construct knowledge [1]. These are readily understood to be essential skills to engage with the modern scientific process. However, whilst there are some educational contexts which have embraced the importance of explicitly teaching educational dialogue as part of a curriculum [2], notably early-years education, these skills are often underdeveloped in students at university, and their importance is overlooked in Science, Technology, Engineering and Mathematics courses [3, 4]. One reason for this is that teachers unfamiliar with educational dialogue may overlook opportunities to integrate it into a sequence of teaching [5]. This work, therefore, seeks to provide science teachers with a common activity that demonstrates how additional context-relevant opportunities for students to practice and develop their educational dialogue skills can be embedded. Importantly, this work does not claim any specific impact on student's educational dialogue skills; the author anecdotally trialled some of these ideas in extension activities with students as part of a computing module (see forthcoming [6]) which prompted some excellent discussions, but, has now since moved educational contexts rendering a rigorous study unfeasible at this time. Rather, then, the intention is to share ideas and practice for other practitioners who might want to integrate educational dialogue more explicitly into their curricula.

The activity is a computer simulation of the classic one-dimensional random walk (sometimes referred to as the drunkard's walk). Each step of the simulation performs a virtual 'coin flip' which decides one of two possible outcomes. As such, a random walk is an example of a stochastic process and provides the basis of several statistical physics phenomena, including Brownian motion and thermodynamics. Here we apply the random walk to simulate the outcome of several games of roulette. This is a simple application of the random walk, which is particularly well-suited to students who are new to programming, statistical mechanics and probability theory, and stochastic processes, because the context is easily imagined. As a mathematical and computational exercise, this is valuable, but there is an explicit opportunity for students to construct and discuss the narrative of a gambler's experience, i.e., how the specific sequences of the wins and losses have led to the gambler's overall winnings (or lack thereof), effectively mapping the gambler's 'journey'. Instead of risking student engagement and "buy-in" by "bolting on" educational dialogue opportunities in these disciplines, this work presents a pedagogical example focused on the process of dialogue itself.

2. Simulation details

The how? A random walk simulation was written to model a simplified game of European roulette (see Appendix). In this model, a player will bet 'red' or 'black', which has a probability $p = \frac{18}{37} = 0.4865$ of winning. When the player wins, they increment their displacement by $+1$, when they lose, they increment by -1. The displacement, which starts from 0, tracks the player's score over a total of 20 rounds of roulette. This model is described by the Binomial theorem, where the probability of observing k successes (wins) across n trials (rounds of roulette) is given by:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{1}$$

which, for $n = 20$ trials and $p = 0.4865$, we would expect the distribution of outcomes given in Figure 1(a).

The why? Whilst each sequence has the same probability of occurring, some sequences will lead to the same final displacements and therefore, both the final displacement and the specific win-loss sequence are of interest. There are, of course, some very unlikely final displacement scenarios which can occur, such as observing 20 consecutive wins, $P(X = 20) = 6 \text{ x } 10^{-7}$, or 6 events out of 10 million. Whilst rare, given

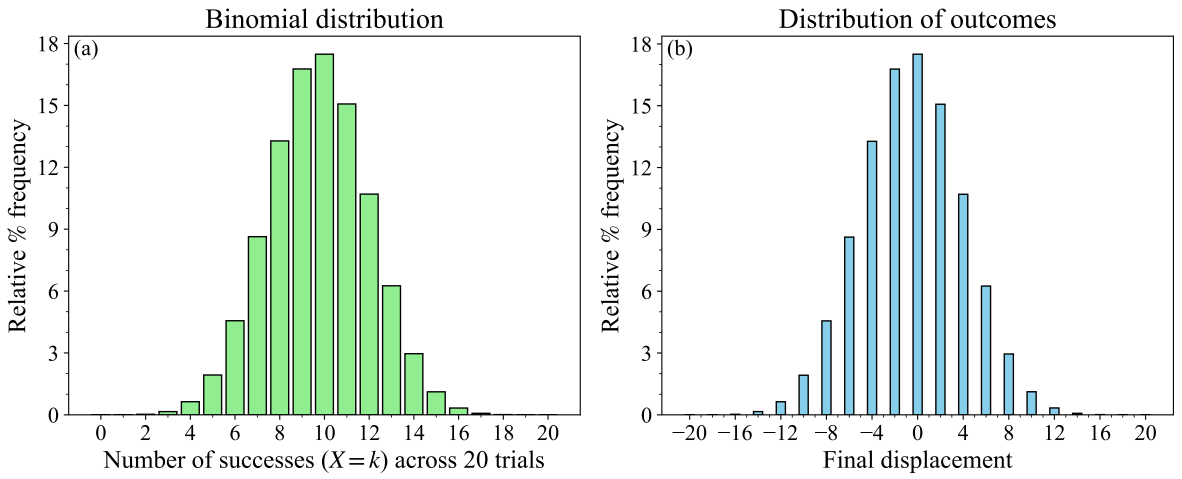

Figure 1: (a) The probability distribution of observing k successes (winning k rounds of roulette) based on the Binomial theorem. (b) The distribution of outcomes from ten million random walks of the gambler playing 20 rounds of European roulette. Each win increments the displacement by $+1$ and each loss by -1. The final displacement from zero give the total winnings. The seed for each random walk is stored and the sequence of win-losses are searched to identify random walk patterns of interest.

the simplicity of the random walk model described, 10^7 independent random walks, each of 20 steps, can be easily simulated in a few minutes on a standard computer. Here, Python (version 3.11.7) is used along with the numpy (version 1.26.4) library to initialise a random seed using *np.random.seed(i)*, where i is incremented from 0 to $10^7 - 1$. For convenience, the displacement array, which contains the result of each simulation, is stored (∼500 Megabytes), which can be searched through for displacement sequences of interest (for instance, using *np.all()* and *np.where()*). The row number of a particular sequence in the displacement array corresponds directly to the seed number used for that simulation, allowing any sequence to be easily found and reproduced. As an example, 20 consecutive wins would result in a displacement sequence of [0, 1, 2, 3, ... 20], and a final displacement of 20. This sequence can then be searched for within the displacement array, which contains all Seeds simulated. Any sequence of interest can be searched in this way, assuming it has occurred within the total number of simulations. The distribution of outcomes from 10^7 simulations is shown in Figure 1(b).

3. A walkthrough of the activities

By generating and storing the displacement array, it can be conveniently searched through for displacement sequences (random walk patterns) for which students can discuss and provide a narrative or context, see Table 1 and Figure 2(a)-(f). Conversely, and arguably more formatively, students can agree on a specific scenario that the gambler might experience, then work out the expected sequence that describes this scenario, before searching the displacement array and determining the Seed number.

Table 1: Examples of interesting win-loss sequences which students can be tasked to find. Importantly, students can be given (or co-construct) a description of an interesting sequence, then determine the displacement sequence of the random walk, before reporting back the Seed number for verification. Figure 2 gives the visual representation of possible sequences.

Pattern	**Description**	**Sequence**	**Example Seed**
Monotonic rise	The gambler wins all the games of roulette.	[0, 1, 2, 3, ... 20]	1966236
Monotonic decrease	The gambler loses all the games of roulette.	[0, -1, -2, -3, ... -20]	466500
Peak	The gambler wins the first half of the roulette games but then loses all remaining rounds.	[0, 1, 2, 3, ... 10, 9, 8 . . . 0]	2493195
Trough	The gambler loses the first half of the roulette games but then wins all remaining rounds.	[0, -1, -2, -3, ... -10, -9, -8 . . . 0]	1697496
Zig-Zag	The gambler experiences persistent win-loss cycles across all the rounds of roulette.	[0, 1, 0, 1, 0, 1 ...]	662501
Delayed rise	The gambler experiences a win-loss cycle for their first 8 rounds of roulette, after which they win all remaining rounds.	[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 2, 3 ... 12]	661202
Increasing oscillation	The gambler experiences larger win-loss cycles where they win once, lose once, then win twice but lose twice, and this continues.	[0, 1, 0, -1, 0, 1, 2, 1, 0, -1, -2, -1, 0, 1, 2, 3, 2, 1, 0, -1, -2]	3216780
Ladder	The gambler wins two rounds in a row before losing a round. This cycle repeats.	[0, 1, 2, 1, 2, 3, 2, 3, 4 . . .]	541788

There are, of course, many potential activities that can be given to students in order to help students develop programmatic (P), mathematical (M) or dialogic (D) skills, with appropriate differentiation to suit the student's educational context.

Writing a 1-D random walk model for European roulette (P). Perhaps the most obvious activity is choosing whether students will write their own 1-D random walk model, augment an existing model, or simply run a piece of code given to them. Similarly, students can be tasked to determine the probability value of the success (or failure) of each round of roulette. Often, this is the first time students realise European roulette has a single '0' (green number) and American roulette (often seen in movies) has '0' and '00' (two green numbers) as well as how many red and black numbers appear

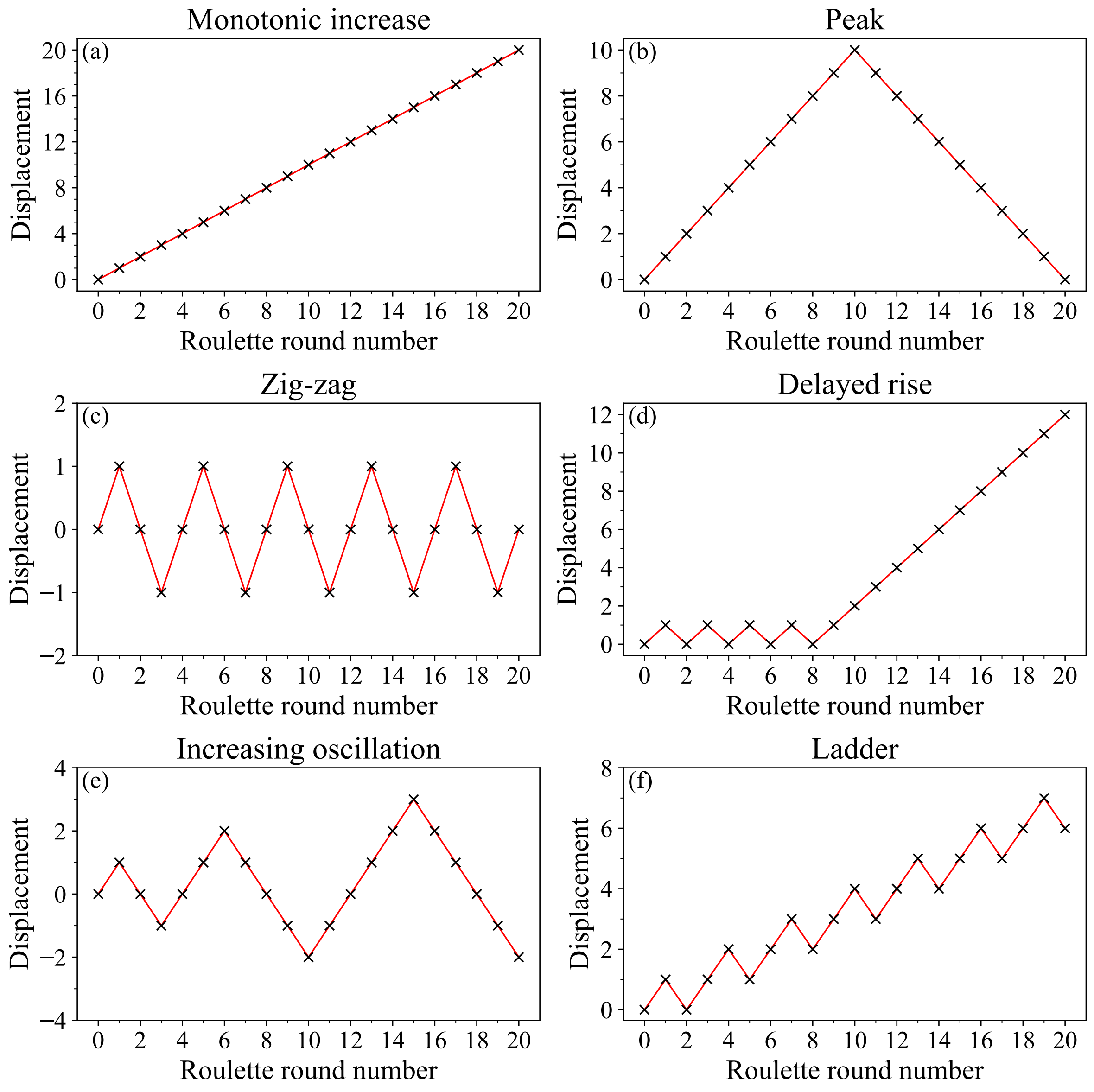

Figure 2: (a)-(f) A number of random walk patterns can be identified. Here, where the random walk models 20 rounds of European roulette, these patterns can be described by the events experienced by the gambler, for example, winning every round of roulette (a), reaching a peak before losing everything (b). The value is that a student can be given or tasked to find a pattern which describe a sequence of events. These are activities for students in pairs or groups to describe and agree on an event of interest, construct the corresponding sequence and searching for it, creating opportunities for developing educational dialogue in addition to the mathematical and programmatic learning objectives.

on the tables. This also provides a worthwhile discussion for students to consider the implications of the two formats (D). This activity would benefit first as a single example of a gambler playing 20 rounds of roulette and plotting the displacement. The sequence of how the displacement has changed over 20 rounds can be more easily deduced from this plot (M). The students should then be encouraged to formulate a sentence or two to explain what the gambler has experienced over the 20 rounds of roulette (D) (e.g., Table 1). Students who rerun this code several times *might* notice the displacement plot keeps changing (M), which provides the opportunity to discuss the nature of stochastic simulations. A basic script for this simulation is provided in the Appendix that can be adapted or extended for these activities.

Simulating many gamblers (P). At this point, students may not realise that a (pseudo)random number is being selected and that it can be controlled by stating the random seed. Fixing the random seed is therefore a good first task. A decision on how to store each simulation into an array which is saved is an additional activity which can be offered (P). Students may also decide how many simulations they want to run. There are two obvious strategies (i) based on how rare the rarest event to be observed is (20 consecutive wins), one can simulate a similar order of magnitude (M; see activity below), (ii) let students try increasing the number of total simulations and quickly learn that adding another magnitude onto the number of simulations can make the runtimes unfeasible!

Predicting probabilities (M). The suggested 1-D random walk model is used as it is fully described by the Binomial theorem. Given this, students can use the Binomial theorem with the number of trials ($n = 20$) and a probability of success (winning; $p = 0.4865$) to calculate the probability ($P(X = k)$) of a specific number of observed successes (e.g., Figure 1(a)). Multiplying the probabilities by the number of simulations gives the expectation value for frequencies of simulated outcomes which have k successes. Additional activities here would include comparing different roulette table probabilities (M), and comparing the distribution of simulated outcomes with the theoretical distribution (as is given in Figure 1(b)). There are several opportunities for students to discuss the implication of the probability distribution, noting for example, the slightly greater cumulative probability for negative displacement values, compared to those greater than 0, which demonstrates the 'house edge' in this game (D). Furthermore, once the probability of any one pattern is determined to be the same, students can discuss the following prompt: "If every specific path is equally rare, why is a final displacement of 0 so much more common than a final displacement of 20?". This is likely counter-intuitive to many students and can bridge concepts of combinatorics (i.e., many paths lead to 0, but only one path leads to +20) (M, D).

Random walk pattern searches (P, D). As the examples in the previous section show, a multitude of random walks can be simulated and specific patterns can be identified. One option is to give students a scenario, for which they must discuss what the sequence of the random walk displacements would look like, and then search the displacement array to find it and report back the Seed. Another option is to give a Seed to students (such as those provided as examples in this work), who then run the simulation and explain what the situation describes. Finally, students can agree on a specific, and possibly novel or imaginative scenario, map this into a displacement sequence, find it in the displacement array and report back the Seed, perhaps then giving this to another pair or group of students to investigate and validate the physical interpretation of the sequence.

Link to physical phenomena (D, P). Students should then be prompted to discuss what the final displacement means for physical systems, such as an ideal gas, a particle undergoing diffusion or Brownian motion. Coupled with simulating the random walk with different probabilities, it can be used to bridge the connection between concepts such as drift velocity (biased random walks).

4. Conclusions

The aim of this work is to provide an example of how educational dialogue can be embedded into classic problems that are traditionally selected to develop students' mathematical and programming skills. Here, that example is a simple random walk model of European roulette. Without loss of programming and mathematical skill development opportunities, framing and extending these exercises can provide several opportunities to develop their educational dialogue skills. Several differentiated activities are suggested to complement the mathematical, programming, and dialogue skill competencies of the students in their educational context.

5. Acknowledgements

The author thanks the teachers and students of the ENG0018 module who engaged in ad hoc discussions about these topics, motivating the author to write this article to share these activities with other teachers.

6. References

[1] Sara Hennessy, Sylvia Rojas-Drummond, Rupert Higham, Ana María Márquez, Fiona Maine, Rosa María Ríos, Rocío García-Carrión, Omar Torreblanca, and María José Barrera. Developing a coding scheme for analysing classroom dialogue across educational contexts. *Learning, culture and social interaction*, 9:16–44, 2016.

[2] Neil Mercer, Paul Warwick, and Ayesha Ahmed. An oracy assessment toolkit: Linking research and development in the assessment of students' spoken language skills at age 11-12. *Learning and Instruction*, 48:51–60, 2017.

[3] Lewis A. Baker and Marion Heron. Advocating for oracy: supporting student success in foundation year study. *Journal for Further and Higher Education*, 47(10):1289–1303, 2023.

[4] Richard M. Felder and Rebecca Brent. *Teaching and Learning STEM: A Practical Guide.* John Wiley and Sons, Inc., San Francisco, CA, 1st edition, 2016.

[5] Marion Heron, Sally Baker, Karen Gravett, and Evonne Irwin. Scoping academic oracy in higher education: knotting together forgotten connections to equity and academic literacies. *Higher Education Research & Development*, pages 1–16, 2022.

[6] Lewis A. Baker, Carol Spencely, Alifah Rahman, and Richard Harrison. Enhancing digital literacy skills as part of an integrated curriculum. *Journal of the Foundation Year Network*, Under review, 2025.

7. Appendix

A very basic Python script to simulate many simplified games of roulette. The author makes no claims of clean or efficient code. The output is a file containing the full history of each game. Each row in the output file represents a single game (a unique random seed; e.g., row 0 was generated using seed number = 0), and each column represents a step in that game, beginning at 0, followed the new displacement after each round of roulette. This allows any specific game's journey to be found and analysed.

```
#Python (version 3.11.7) and numpy (version 1.26.4) used
import numpy as np

#Initialise length of each game, total number of games
number_of_steps = 20
games = 10000000

#Allocate storage to a displacement array to save outputs
displacement = np.zeros((games, number_of_steps + 1))

#Initialise displacement magnitude and direction (vector)
dx = 0
step_direction = 0

#Loop through each game, using a new random seed for each game
for i in range(games):
        np.random.seed(i)

        for j in range(number_of_steps):
                step_direction = np.random.random()

                #Probability of loss = 1 − (18/37) = 0.5135
                if step_direction < 0.5135:
                dx = −1
                else:
                dx = +1

                #Store the cumulative displacement sequence
                displacement[i, j+1] = displacement[i, j] + dx

#Save displacement array as a file for later searching
filename = ‘simulated_games.txt’
with open(filename, ‘w’) as f:
        np.savetxt(f, displacement, fmt=‘%i’, delimiter=‘,’)
```