Exercise 1: Building RNNs: From Rolling Balls to Hidden States

Step 1.1 Imagine a ball rolling on a surface with air resistance (drag). The ball's momentum (hidden state) changes based on:

1. Previous momentum (how fast it was already rolling)
2. New external force (input)

For force sequence [1, 1, 1, 1], shade boxes to show momentum (white = 0 m/s, light gray = 1 m/s, medium gray = 2 m/s, dark gray = 3 m/s, black = 4+ m/s):

Momentum				
	t_1	t_2	t_3	t_4

Step 1.2 In both physics and RNNs, we need a way to control how much past information is retained. In physics, this is air resistance (drag), which reduces velocity proportionally to speed. In RNNs, we use a weight (w) that serves the same purpose - it determines what fraction of the previous momentum is kept. A high weight ($w = 0.9$) means 90% of momentum is retained (low drag), while a low weight ($w = 0.1$) means only 10% is retained (high drag).

For force sequence [2, 2, 0, 0], shade three sets of boxes for:

(a) High retention ($w = 0.9$)

(b) Medium retention ($w = 0.5$)

(c) Low retention ($w = 0.1$)

$w = 0.9$				
$w = 0.5$				
$w = 0.1$				

> *Think:* How does w affect the network's memory of past inputs?

Step 1.3 Just as real objects can't have infinite speed due to air resistance, RNNs need a way to prevent unbounded growth of the hidden state. We introduce the tanh activation function as our "speed limit." Let's see how our physical model becomes an RNN:

Physical model: new momentum $= w \times$ old momentum $+ w_x \times$ force RNN model: $h_{\text{new}} = \tanh(w \times h_{\text{old}} + w_x \times \text{force})$

where:

1. h_{old} is previous momentum
2. force is input
3. w controls memory retention (like inverse drag coefficient)
4. w_x controls force sensitivity
5. tanh keeps values between -1 and 1 (our speed limit)

For these force sequences, shade the predicted momentum (white = -1, light gray = -0.5, medium gray = 0, dark gray = 0.5, black = 1), with $w = 1.0$ and $w_x = 1$:

[3,0,0,0]	[1,1,1,1]	[0,3,0,0]
☐ ☐ ☐ ☐	☐ ☐ ☐ ☐	☐ ☐ ☐ ☐

> *Think:* How does each force sequence affect momentum differently? Which sequence would be hardest for the network to "learn"?

Step 1.4 Design your own RNN weights! If you wanted to: (Design 1) Remember past inputs longer, (Design 2) Respond more quickly to new forces, (Design 3) Have a maximum speed limit. Shade these weight matrices (white = 0, black = 1) to achieve each goal:

Design 1		Design 2		Design 3	
w	w_x	w	w_x	w	w_x

Step 1.5 Predict what would happen with these "physically impossible" weights:

1. $w > 1$ (momentum grows from previous state)
2. No activation function
3. Negative weights

Shade the momentum evolution for input sequence [1, 1, 1, 1] (use white = minimum value, black = maximum value in each case):

$w = 1.2$				
No tanh				
$w = -0.5$				

Think: Why are these scenarios "impossible" physically but possible in an RNN? What problems might they cause or solve?