Models of opinion formation have become popular in recent years. The basic idea is that the opinions of others will influence the opinion of individuals. The first two projects in the following explore some of the popular models.

Project 14.23 Models of opinion formation

(a) *The voter model.* On a regular lattice, assign each site the value ± 1. Choose a site (the voter) at random. The voter then adopts the same value as a randomly chosen neighbor. These two steps continue until all sites have the same value, that is, when they have reached consensus. Compute the probability of achieving a consensus of $+1$ given that the initial density of $+1$ sites is ρ_0. Use a 10×10 square lattice and make at least 20 runs at each density. Also compute the time to reach consensus as a function of the lattice size. In two dimensions this time scales as $N \ln N$, where N is the number of sites. How does the consensus time scale with N in $d = 1$ and $d = 3$ dimensions? How does it scale on a preferential attachment network (see the article by Sood and Redner)?

(b) *The relative agreement interaction model.* N individuals are initially assigned an opinion that takes on a value between 0 and 1. Choose two individuals, i and j, at random. Assume that the ith opinion O_i is greater than the jth opinion O_j. If their opinions differ by less than the parameter ϵ, then increase O_j by $(m/2)(O_i - O_j)$ and decrease O_i by the same amount, where m is another parameter. This model implements the idea that two people will influence each other only if their opinions are sufficiently close. Write a program to simulate this model. Use a `LatticeFrame` for which each cell can take on one of 256 values. The approximation of the continuum by 256 values is for visualization purposes only, and the 256 values should be sufficiently large to approximate a continuum of values. Choose $\epsilon = 10, 50$, and 100 (out of 256), and $m = 0.3$ and 0.6. To speed up the simulation include in your program the option to plot configurations only after a certain number of iterations (use `enableStepsPerDisplay(true)`). Choose $N \geq 2500$ and begin with a random set of opinions. Discuss whether a single opinion emerges and explain the magnitude of the fluctuations.

(c) *The Sznajd model.* Place individuals on a square lattice with linear dimension L and using periodic boundary conditions. Each individual has one of two opinions. At each iteration, an individual and one of the person's neighbors is chosen at random. If the two individuals have the same opinion, the opinion of the six neighbors of the pair is changed to that of the pair. The idea is that people are more likely to change their opinion to those physically near them if more than one person shares the same opinion (peer pressure). Write a program to simulate this model and show that consensus is always reached for all sites if the simulation is run for a sufficiently long time. Discuss the visual appearance of the groups of like-minded individuals. Consider initial configurations where the individuals are randomly assigned the two opinions and initial configurations where one opinion has a majority of 1%, 5%, and 10%. Choose $L \geq 50$.

(d) Generalize the Sznajd model so that an individual may be assigned one of more than two opinions. Is consensus still always reached? What happens if the individuals are not on the sites of a square lattice, but rather are the nodes of a preferential attachment network of at least 5000 nodes? ■

Project 14.24 The minority game

In certain situations we wish to be in the minority. For example, we might wish to go to a popular restaurant on an off-night so that we do not have to wait in line. A business might want to sell goods and services that are not being sold by other businesses. The following algorithm, known as the *minority game*, is a model of adaptive competition where each player tries to maximize his gain. We will find that there is a phase transition between states where the players mainly act on their own and states for which cooperative behavior emerges.

There are N players, where N is odd. At each iteration, each player can choose one of two actions which we call 1 or 0 but which we encode as the boolean `true` or `false`. A player's choice is determined by a strategy based on the previous m (memory) iterations. Each strategy is represented by a table of all the possible outcomes of the previous m iterations and a decision on what to do for each outcome. Each player has his own table of strategies. An outcome is defined as the action that was chosen *least* by all the players. For example, suppose $m = 2$. There are four possible pasts: (1,1), (1,0), (0,1), and (0,0). The past (1,1) means that in each of the last two iterations, action 1 was chosen by a minority of the players. A strategy would be encoded by a table such as the following: (1,1,1), (1,0,0), (0,1,0), and (0,0,1). The first two entries in each triple are the possible outcomes of the last two iterations, and the third entry in the triple gives the action that the strategy suggests taking. Thus the triple (1,1,1) means that if (1,1) occurred in the past, the strategy is to choose action 1; (1,0,0) means that if (1,0) occurred in the past, the strategy is to use action 0. At the beginning of the game, each player is assigned at least two strategies that are chosen at random from all possible strategies. As the game is played, the performance of each strategy (whether or not it is used) is updated, such that if a strategy leads to the same action that was in the minority, then this strategy is successful and its performance is incremented by unity; otherwise, it stays the same. At each iteration each player chooses the strategy with the best performance and takes the action determined by his best performing strategy. Then the outcome (which action was in the minority) for that iteration is determined, and the past m outcomes and the performance for all the strategies are updated. Note that the strategies available to each player does not change, but which of each player's strategy is best changes as the game is iterated.

To simplify the code, represent the past outcomes by an integer where each bit represents an outcome. For example, the bit 110 means that the outcome was 0 in the last iteration and was 1 for each of the earlier two iterations. You will need the following arrays: `strategies[i][j][k]`, which gives the action for the ith player using its jth strategy when the kth past occurred; `performance[i][j]`, which gives the performance for the ith player's jth strategy, and `chosenStrategy[i]`, which gives the strategy chosen by the ith player in the current iteration.

Let N_1 equal the number of players who chose action 1 in one iteration. The outcome is best if at each iteration the value of N_1 is close to $N/2$, because in this way there would be as many players as possible in the minority. The quantity of interest is σ, where σ is defined as

$$\sigma^2 = \sum_k (N_1(k) - \langle N_1 \rangle)^2, \tag{14.9}$$

and the sum is over all the iterations of the game, and $N_1(k)$ is the number of players choosing action 1 in the kth iteration. The quantity σ decreases as the efficiency increases.