Markov Chain Simulation Library

Specifications

Background:

This library is intended to simulate and visualize a Markov decision process represented as a Markov chain. To develop, test, and understand this library you do not need in-depth knowledge of what a Markov chain is. Although, a primer on Markov chains is also provided alongside this specification.

The key idea is that a Markov chain has one "active state" among a group of "possible states". At each time step of the simulation (discrete steps, the actual time units does not matter), the chain randomly transitions from its active state into a new state, which becomes the new active state. While the transition is random, the probability of transitioning to each state is encoded as the Markov chain's transition matrix.

Goal:

The ultimate goal is to simulate *walks* generated by a Markov chain. A `RandomWalker` class uses a Markov chain to encode randomness, and then walks around a two-dimensional grid for a certain number of steps. The *path* this walk produces is saved to a file and can be visualized using the `WalkFrame` class provided.

Input Files:

The simulation is completely controlled by the Markov chain and, thus, its transition matrix. The input files thus encode the matrix to use for the Markov chain and the random walker. It is in the format expected by `FloatMatrix`, see below. The Markov chain must have 4 states, each corresponding to a cardinal direction: North, East, South, West, in that order.

Output Files:

The simulation outputs a text file representing the path produced by the walker. Each line of the output file represents a single Coordinate along the path. It starts at the starting point and each successive line is one successive step along the path.

Class Summary:

- `FloatMatrix`**:** a class for matrices of floating point numbers
- `MarkovChain`**:** a class for encoding a Markov chain
- `Coordinate`**:** a class for encoding two-dimensional Cartesian coordinates

- `RandomWalker`: a class using a Markov chain to randomly walk around a lattice
- `WalkSim`: the class containing the main method to read input and produce output
- `WalkFrame,WalkCanvas`: GUI classes for visualizing a walk

FloatMatrix

The `FloatMatrix` class represents a matrix of floating point numbers. Each entry of the matrix is encoded as a float. Matrices must have at least one row and at least one column. A `FloatMatrix` can be created from an array of floating point numbers. This class allows for getting and setting individual entries of the matrix. It also provides functionality for multiplying matrices together, where the dimensions of the input matrices are compatible for multiplication.

A class method reads in a properly formatted text file to create a FloatMatrix object. In this text file, each row of the matrix is one comma-separated line of text. Each line of the file should contain the same number of values and the same number of commas. No other information should be in the file.

MarkovChain

The `MarkovChain` class represents a Markov chain. It is encoded as a transition matrix, implemented as a `FloatMatrix`, and a "current state" instance variable. The transition matrix must be square and must have the entries of each row sum to 1. The number of states in the chain is equal to the number of rows in its transition matrix. A class method is provided to validate if a `FloatMatrix` is a valid transition matrix. The class provides the ability to label each state of the chain with a string label.

The chain transitions to its next state only when a client prompts it to do so by calling its `nextState()` instance method. Getter methods are provided for its current state and number of possible states. Clients can manually set the active state by calling `setState().`

Coordinate

The Coordinate class encodes a pair of integers representing the possible integer coordinates on a two-dimensional Cartesian plane. Because the coordinates are integers, this is really coordinates of a two-dimensional *integer lattice* (fancy words for 2D points with only integers as values). This class provides functionality for adding and subtracting coordinates. It also provides the ability to scale (multiply by a constant) a coordinate by a scalar number.

RandomWalker

The Random walker encodes a random "walker". A walker begins at a particular location, encoded by a Coordinate. Then, the walker randomly moves one unit to the North, South, East, or West. This "walk" continues for a certain number of "steps". A "path" traces out the walk as a list of Coordinates: the starting point plus one Coordinate for each step of the walk.

The randomness of the walker is encoded by a MarkovChain with four states, each corresponding to one of North, South, East, West. With each step of the walk, the walker gets a new state from the MarkovChain and moves one unit in the corresponding direction.

WalkSim

The class containing the main method. Command-line arguments can be passed to specify the input file name, the output file name, and the number of steps to simulate. Without any CLI argument, the program will use default files names and numbers of steps. This class assumes that the float matrix read from file encodes its states in the order: North, East, South, West.

Recall that you can set the command line arguments in IntelliJ via Run > Edit Configurations... > Build and Run > Program arguments.

WalkCanvas

A graphical component for drawing out the path of a walker. The canvas displays a Cartesian plane with the origin (0,0) at the center. It is a subclass of java.awt.Canvas.

WalkFrame

A graphical window in which to hold and display the WalkCanvas. It is a subclass of java.awt.Frame.