Drunken ant

Ants mainly communicate with each other using odorants called **pheromones**. Like other insects, ants perceive smells with their long, thin, and mobile antennae. The paired antennae provide information about the direction and intensity of scents. Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed by other ants. A forager that finds food, marks a trail on the way back to the colony. If this trail is followed by other ants, these ants then reinforce the trail when they head back with food to the colony. When the food source is exhausted, no new trails are marked by returning ants and the scent slowly dissipates.

Foraging ants travel distances of up to 200 meters from their nest and scent trails allow them to find their way back even in the dark. Distances traveled are measured using an internal pedometer that keeps count of the steps taken and also by evaluating the movement of objects in their visual field. Directions are measured using the position of the sun. They integrate this information to find the shortest route back to their nest. Ants that become separated from pheromone trails (for example because they have been moved to another location by a human being) may run around continuously until they die of exhaustion.

Assignment

In this assignment we simulate the behaviour of an ant tracking its way back to the nest by following a disturbed pheromone trail. The environment in which the ant forages is represented by a square $n \times n$ grid having n row and n columns. The ant initially is located in the bottom left corner of the grid and its nest is located in the top right corner of the grid. Each cell in the grid contains a pheromone trail that points up, down, left or right. On each turn in the simulation, the ant moves to the neighbouring cell that is pointed to by the pheromone trail, and then the direction of the trail on the cell that it has left turns 90° clockwise. If the ant is not able to move in the indicated direction because the edge of the grid blocks its path, it remains on its current cell and the direction of the trail on the cell turns 90° clockwise.

```
step:     0
arrow: > ,  position: (3, 0)

 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
[>] >  v  >
```

The original configuration of a square $n \times n$ grid with pheromone trails is stored in a text file. The file contains n lines, with each line having n direction symbols that are separated from each other by a single space. The following four characters are used as direction symbols:

- `>`: the pheromone trail points to the right
- `<`: the pheromone trail points to the left
- `^`: the pheromone trail points up
- `v`: the pheromone trail points down

You are asked to define a class `DrunkenAnt` that can be used to simulate the behaviour of an ant. The objects of this class must support at least the following methods:

- An initialization method that takes the location of a text file. This file contains the original configuration of a square $n \times n$ grid with pheromone trails. Initially the ant is located at the bottom left corner of the grid.
- A method `position` that returns the current position of the ant in the grid. The rows of the grid are indexed top to bottom, and the columns from left to right, with indexing starting from zero. Each position in the grid is represented as a tuple whose first element indicates the row index and whose second element indicates the column index.
- A method `__repr__` that returns a string representation of the current status of the environment. This string representation represents a square $n \times n$ grid in the same format as used to store the original configuration of the grid in a text file. Note that by simulating the steps of the ant, the direction symbols in the grid may have changed with respect to the initialization of the object.
- A method `step` that can be used to perform a single simulation step. The method must return the new position of the ant after a single step of the ant has been simulated.
- A method `steps` that can be used to perform an entire simulation of the steps taken by the ant from its current position until it reaches its nest at the top right corner of the grid. The method must return a list of positions that starts with the original position of the ant at the start of the simulation, followed by the positions of the ant after each step in the simulation.
- A method `__str__` that returns a string representation of the current status of the environment, which also indicates the current position of the ant in the grid. In contrast to the string representation returned by the method `__repr__`, the direction symbols on each line are no longer separated by spaces, but each direction symbol is preceded and followed by a single space. Instead of using spaces, the direction symbol at the current location of the ant is preceded by an opening square bracket (`[`) and followed by a closing square bracket (`]`).

Example

In the following interactive session we assume that the text file square.txt is located in the current directory.

```
>>> ant = DrunkenAnt('square.txt')
>>> ant.position()
(3, 0)
>>> ant
> > > >
^ < ^ v
^ v ^ ^
> > v >
>>> print(ant)
 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
[>] >  v  >

>>> ant.step()
(3, 1)
>>> ant.position()
(3, 1)
>>> ant
> > > >
^ < ^ v
^ v ^ ^
v > v >
>>> print(ant)
 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
 v [>] v  >

>>> ant.step()
(3, 2)
>>> ant.position()
(3, 2)
>>> ant
> > > >
^ < ^ v
^ v ^ ^
v v v >
>>> print(ant)
 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
 v  v [v] >

>>> ant.steps()
[(3, 2), (3, 2), (3, 1), (3, 1), (3, 0), (3, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 3)]
>>> ant.position()
(0, 3)
>>> print(ant)
 v  v  v [>]
 >  <  ^  v
 >  v  ^  ^
 >  ^  ^  >
```

Epilogue

You may have wondered whether it is possible that during a simulation an ant keeps wandering without ever reaching its nest. We can easily prove that an ant will always reach the upper right

corner of the grid in a finite number of steps. Suppose by way of contradiction that an ant stays in the grid forever. Because the grid contains a finite number of cells, this means the ant will visit at least one cell an infinite number of times. And because this cell rotates after each visit, it follows that the ant visits each of its adjacent cells an infinite number of times. By extension this means that the ant will visit every cell in the grid an infinite number of times. This includes the cell in the upper right corner. Hence it's not possible for the ant to stay in the grid forever without ever reaching its nest.

Mierencommunicatie verloopt hoofdzakelijk door middel van geurstoffen die **feromonen** genoemd worden. Ze worden bijvoorbeeld gebruikt om aan te geven waar er voedsel te vinden is. Zo kan een voedselverzamelaar een spoor op de grond achterlaten om andere voedselverzamelaars te informeren waar er voedsel te vinden is. Mieren die het feromonenspoor volgen, zorgen voor een versterking van het signaal, waardoor nog meer mieren het spoor blijven volgen totdat het voedsel is uitgeput. Het feromonenspoor wordt dan niet langer versterkt en vervaagt langzaam.

Behalve feromonensporen gebruiken mieren verschillende soorten informatie om de weg naar hun nest terug te vinden. Zo tellen ze bijvoorbeeld het aantal stappen dat ze hebben genomen sinds hun vertrek, hebben ze een ingebouwd kompas en slaan ze herkenningspunten op in hun geheugen. Mieren die een feromonenspoor verliezen (bijvoorbeeld omdat ze door een mens op een andere plaats worden neergezet) hebben het vaak moeilijker om hun nest terug te vinden.

Opgave

In deze opgave simuleren we het gedrag van een mier die op basis van een verstoord feromonenspoor de weg naar zijn nest probeert terug te vinden. De omgeving waarin de mier zich beweegt, wordt voorgesteld als een vierkant $n \times n$ rooster met n rijen en n kolommen. De mier bevindt zich initieel in de linkeronderhoek van het rooster en zijn nest bevindt zich in de rechterbovenhoek van het rooster. Elke cel van het rooster bevat een feromonenspoor dat wijst naar boven, onder, links of rechts. Bij elke simulatiestap verplaatst de mier zich naar de naburige cel die wordt aangegeven door het feromonenspoor, waarna de richting van het spoor op de cel die wordt verlaten 90° in wijzerzin gedraaid wordt. Indien de mier niet in de aangegeven richting kan bewegen omdat ze tegen de rand van het rooster

aanloopt, blijft ze op haar huidige positie staan, maar wordt de richting van het spoor op de cel wel nog 90° in wijzerzin gedraaid.

```
stap:    0
pijl: > ,  positie: (3, 0)

 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
[>] >  v  >
```

De originele configuratie van een vierkant $n \times n$ rooster met feromonensporen werd opgeslaan in een tekstbestand. Het bestand bevat n regels en elke regel bestaat uit n richtingsaanwijzers die van elkaar worden gescheiden door een spatie. De volgende vier karakters worden gebruikt als richtingsaanwijzer:

- `>`: het feromonenspoor leidt naar rechts
- `<`: het feromonenspoor leidt naar links
- `^`: het feromonenspoor leidt naar boven
- `v`: het feromonenspoor leidt naar onder

Gevraagd wordt om een klasse `DronkenMier` te definiëren, waarmee de beweging van een mier kan gesimuleerd worden. De objecten van deze klasse moeten minstens de volgende methoden hebben:

- Een initialisatiemethode waaraan de locatie van een tekstbestand moet doorgegeven worden. Dit bestand bevat de originele configuratie van een vierkant $n \times n$ rooster met feromonensporen. Initieel bevindt de mier zich in de linkeronderhoek van het rooster.
- Een methode `positie` die de huidige positie van de mier in het rooster teruggeeft. De rijen van het rooster worden genummerd van boven naar onder, en de kolommen van links naar rechts, waarbij de nummering telkens start vanaf nul. Een positie in het rooster wordt voorgesteld als een tuple waarvan het eerste element de rij-index aangeeft en het tweede element de kolom-index.
- Een methode `__repr__` die een stringvoorstelling van de huidige toestand van de omgeving teruggeeft. Deze stringvoorstelling stelt een vierkant $n \times n$ rooster voor in hetzelfde formaat dat gebruikt wordt om de originele configuratie van het rooster voor te stellen in een tekstbestand. Merk op dat door het simuleren van de stappen die mier zet, de richtingsaanwijzer in het rooster reeds kunnen gewijzigd zijn ten opzichte van de initialisatie van het object.
- Een methode `stap` waarmee één enkele simulatiestap wordt uitgevoerd. De methode moet de nieuwe positie van de mier teruggeven nadat één enkele beweging van de mier werd uitgevoerd.
- Een methode `stappen` waarmee een volledige simulatie wordt uitgevoerd van de bewegingen die de mier maakt vanaf zijn huidige positie tot aan zijn nest in de rechterbovenhoek van het rooster. De methode moet een lijst van posities teruggeven die start met de positie waarop de mier zich bevond bij aanvang van de simulatie, gevolgd door de posities waarop de mier zich bevond na elke stap uit de simulatie.
- Een methode `__str__` die een stringvoorstelling van de huidige toestand van de omgeving teruggeeft, waarbij ook de huidige positie van de mier in het rooster wordt aangegeven. In tegenstelling tot de stringvoorstelling die door de methode `__repr__` wordt teruggegeven,

worden de richtingsaanwijzers op elke regel nu niet van elkaar gescheiden door een spatie, maar wordt elke richtingsaanwijzer voorafgegaan en gevolgd door één enkele spatie. In plaats van gebruik te maken van spaties, wordt de richtingsaanwijzer op de huidige positie van de mier voorafgegaan door een openend vierkant haakje ([) en gevolgd door een afsluitend vierkant haakje (]).

Voorbeeld

Bij onderstaande voorbeeldsessie gaan we ervan uit dat het tekstbestand vierkant.txt zich in de huidige directory bevindt.

```
>>> mier = DronkenMier('vierkant.txt')
>>> mier.positie()
(3, 0)
>>> mier
> > > >
^ < ^ v
^ v ^ ^
> > v >
>>> print(mier)
 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
[>] >  v  >

>>> mier.stap()
(3, 1)
>>> mier.positie()
(3, 1)
>>> mier
> > > >
^ < ^ v
^ v ^ ^
v > v >
>>> print(mier)
 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
 v [>] v  >

>>> mier.stap()
(3, 2)
>>> mier.positie()
(3, 2)
>>> mier
> > > >
^ < ^ v
^ v ^ ^
v v v >
>>> print(mier)
 >  >  >  >
 ^  <  ^  v
 ^  v  ^  ^
 v  v [v] >

>>> mier.stappen()
[(3, 2), (3, 2), (3, 1), (3, 1), (3, 0), (3, 0), (3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 3)]
```

```
>>> mier.positie()
(0, 3)
>>> print(mier)
 v  v  v [>]
 >  <  ^  v
 >  v  ^  ^
 >  ^  ^  >
```

Epiloog

Je hebt je misschien wel afgevraagd of het niet mogelijk is dat een mier tijdens de simulatie eeuwig rondjes blijft draaien zonder ooit zijn nest te bereiken. We kunnen echter eenvoudig aantonen dat een mier steeds na een eindig aantal stappen de rechterbovenhoek van het rooster zal bereiken. Veronderstel bij wijze van contradictie dat de mier voor eeuwig en altijd in het rooster blijft rondlopen. Omdat het rooster bestaat uit een eindig aantal cellen, betekent dit dat de mier minstens één cel een oneindig aantal keer bezoekt. Aangezien de richtingsaanwijzers bij elke stap draaien, volgt hieruit dat ook elke naburige cel een oneindig aantal keer bezocht wordt. Bij uitbreiding betekent dit ook dat elke cel in het rooster een oneindig aantal keer bezocht wordt, inclusief de cel in de rechterbovenhoek. Bijgevolg is het dus niet mogelijk dat de mier eeuwig in het rooster blijft ronddwalen, zonder de rechterbovenhoek van het rooster te bereiken.