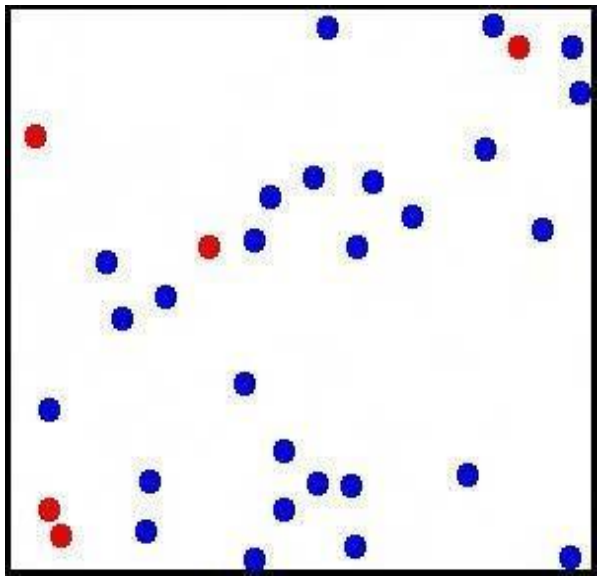
Random Walks

1

Random Walks

A **random walk** is a mathematical object, known as a stochastic or random process, that describes a path that consists of a succession of random steps.

In other words it describes the occurrence of an event (determined by a series of random movements) that cannot be predicted (follows no pattern or trend).



Why Random Walks?

- Random walks are important in many domains
  - Understanding the stock market (maybe)
  - Modeling diffusion processes
  - Etc.
- Good illustration of how to use simulations to understand things.
- Excuse to cover some important programming topics:
  - Practice with classes
  - Practice with plotting

Random Walks – Drunkards Walk

A random walk that simulates actual walking.

In this problem we imagine a drunken farmer standing in the middle of a field.

Every second the farmer takes one step in a random direction.

What is his/her expected distance from the origin in 1000 seconds?

If (s)he takes many steps, is (s)he likely to move ever farther from the origin, ***or*** is (s)he likely to wander back to the origin over and over, and end up not far from where (s)he started?

Using a Python simulation, we can answer these questions.

Understanding the Problem

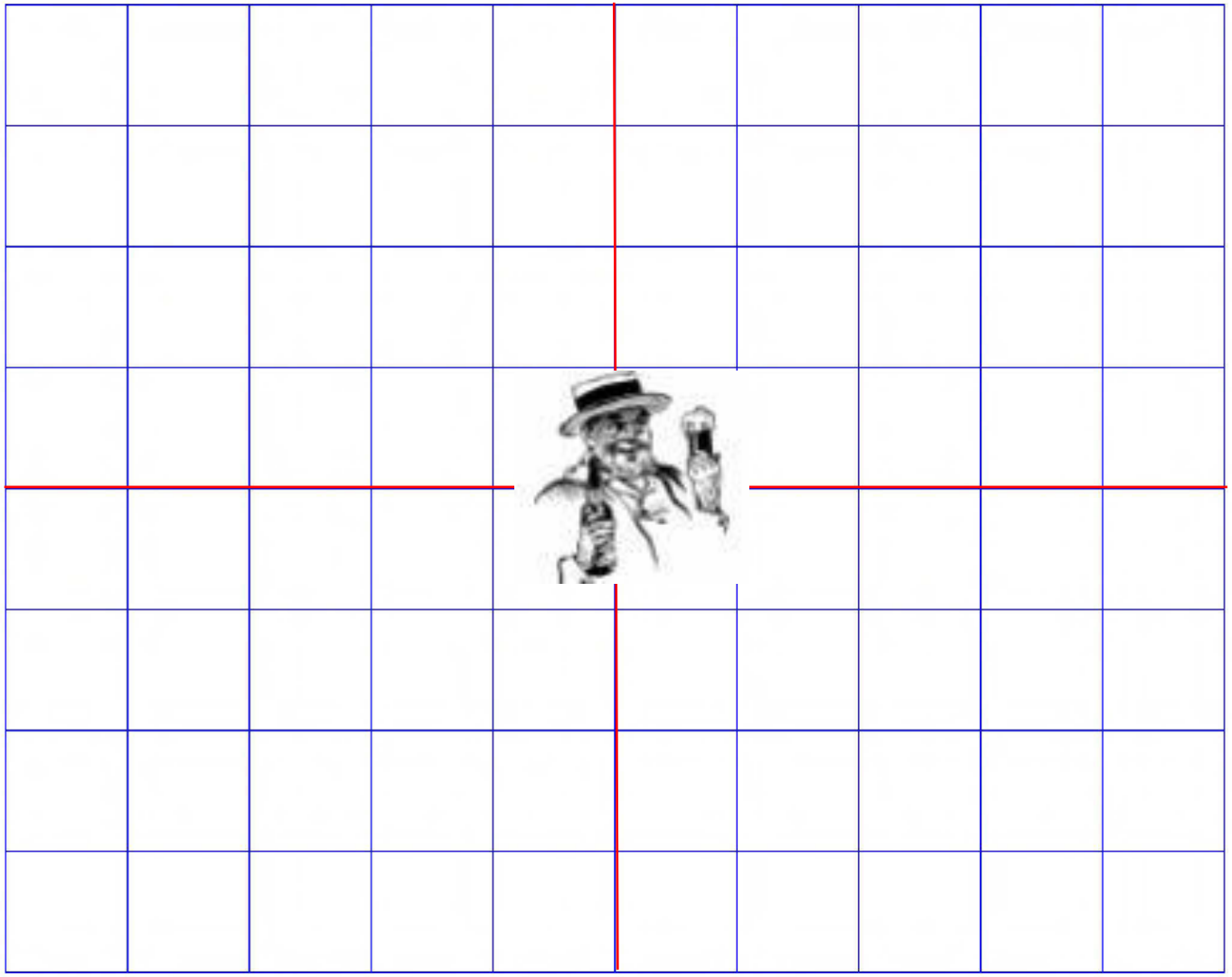
The following slides allow us to visualize the problem to help understand what is required.

The model of the drunkards walk is shown using Cartesian coordinates.

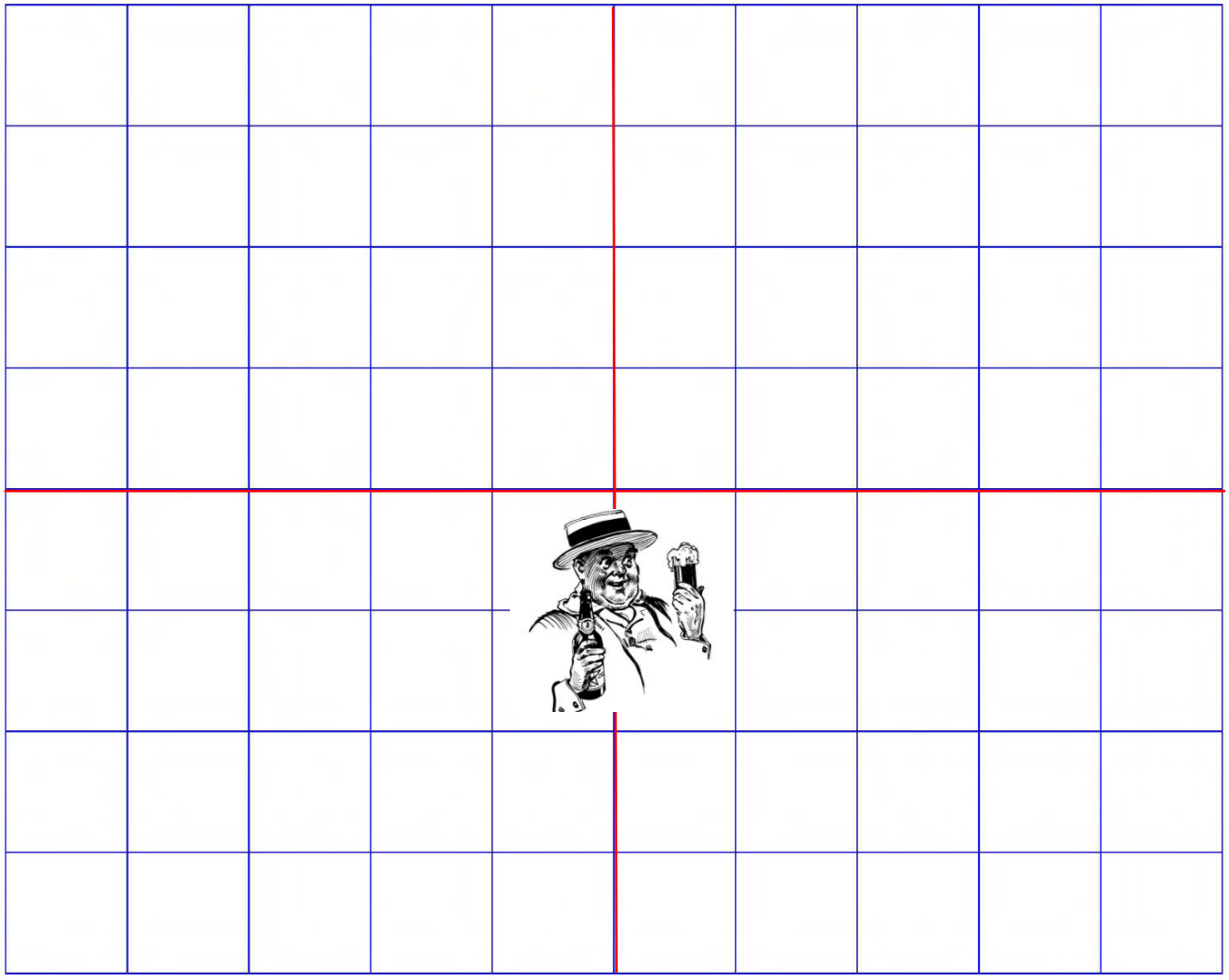
We will make the following assumptions:

- Each step the farmer takes is of length 1
- Each step is parallel to either the x or y axis.

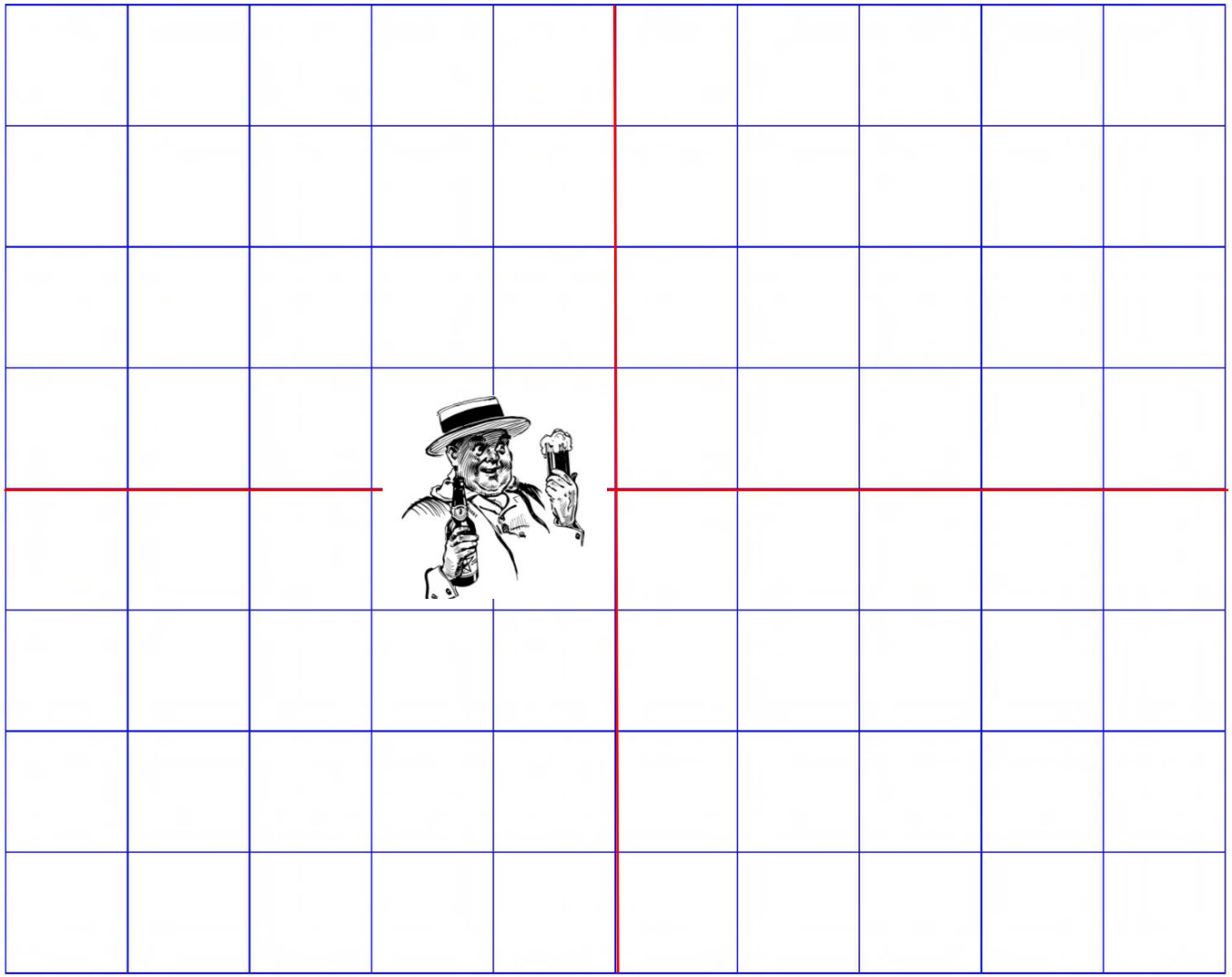
Drunkard’s Walk



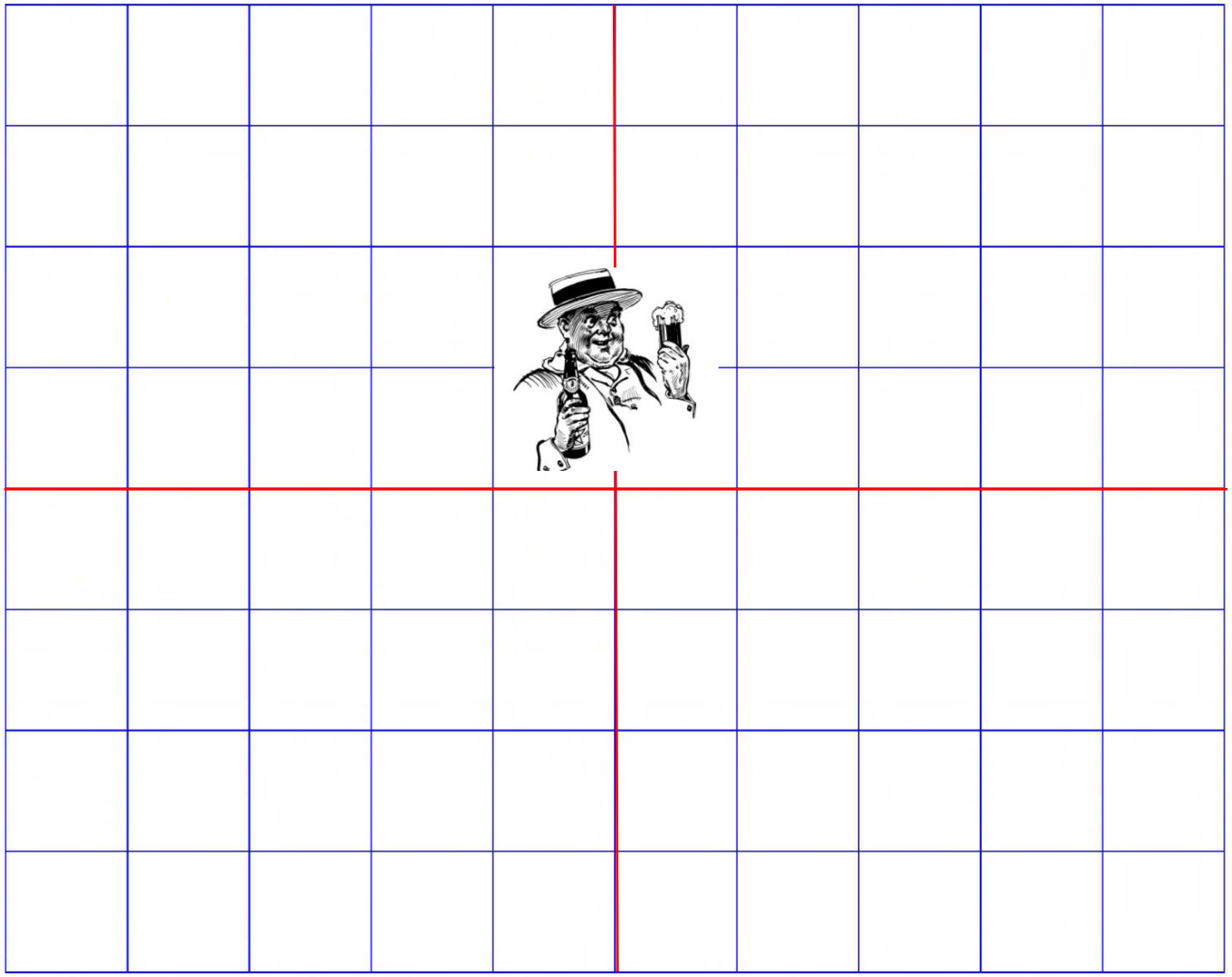
One Possible First Step



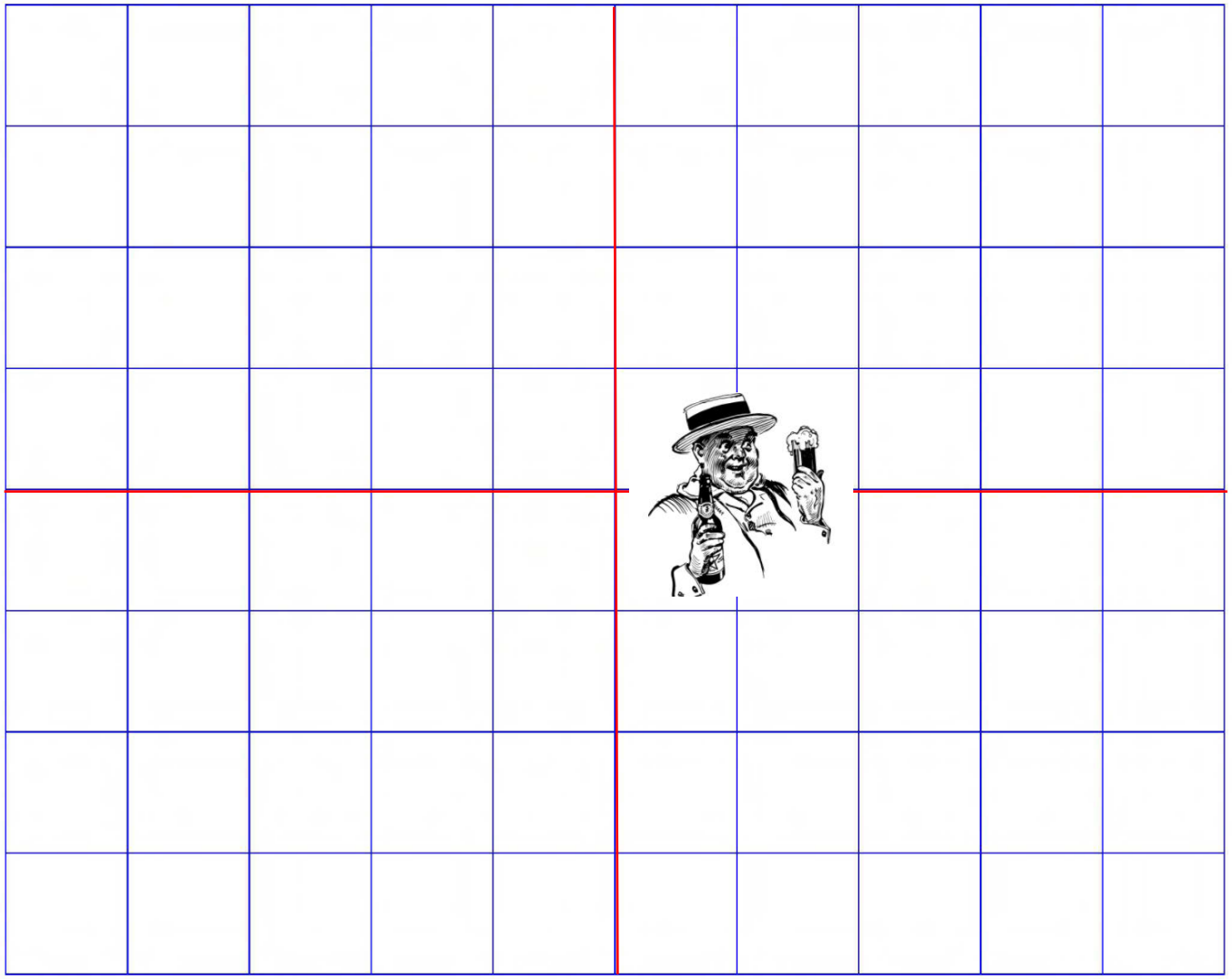
Another Possible First Step



Yet Another Possible First Step



Last Possible First Step



Expected Distance After 100,000 Steps?

- To determine the probable distance after a large number of steps we need a different approach.
- Using Python, we can implement this with a simulation.
- The problem as you will see lends itself to an object-oriented approach.

Structure of Simulation

Our simulation will do the following:

- Simulate one walk of k steps.
- Simulate n walks of k steps.
- Simulate m drunks walking n walks of k steps.
- Report distance from origin.

First, Some Useful Abstractions

- Location—a place
- Field—a collection of places and drunks
- Drunk—somebody who wanders from place to place in a field

Class Location, part 1

```
class Location(object):
    def __init__(self, x, y):
        """x and y are floats"""
        self.x = x
        self.y = y

    def move(self, deltaX, deltaY):
        """deltaX and deltaY are floats"""
        return Location(self.x + deltaX,
                        self.y + deltaY)

    def getX(self):
        return self.x

    def getY(self):
        return self.y
```

Class Location, continued

```
def    distFrom(self,     other):
          xDist   =            -  other.getX()
          yDist           self  -  other.getY()
          return  .x  (xDist**2  +
                  =                      yDist**2)**0.
def    __str__(self):     self     5
          return'<y       +       str(self.x)  +       ',
                          +       str(self.y)          '\
                                                +      '>'
```

Class Drunk

```
class Drunk(object):
    def __init__(self,  name =
            None): """Assumesname  is
            a             str"""
        self.name=      name

    def __str__(self):
        if  self  !=
            None:
             return
            self.name return
            'Anonymous'
```

Not intended to be useful on its own. This is a *base class to be inherited.*

Subclass of Drunk

```
import random

class  UsualDrunk(Drunk):
     def takeStep(self):
          stepChoices =[(0,1),(0,-1),(1,0),(-1,0)]
          return random.choice(stepChoices)
```

Field Class:

```
class Field(object):
    def __init__(self):
        self.drunks = {}

    def addDrunk(self, d, loc):
        self.drunks[d] = loc

    def getDrunks(self):
        return self.drunks

    def moveDrunk(self, drunk):
        if drunk not in self.drunks:
                print('Drunk not in field')
        else:
            xDist, yDist = drunk.takeStep()
            currentLocation = self.drunks[drunk]
            #use move method of Location to get new location
            self.drunks[drunk] = currentLocation.move(xDist, yDist)

    def getLoc(self, drunk):
        if drunk not in self.drunks:
            print('Drunk not in field')
        else:
            return self.drunks[drunk]
```

Simulating a Single Walk

```
def walk(f, d, numSteps):
    """Assumes: f a Field, d a Drunk in f, and
       numSteps an int >= 0. Moves d numSteps times, and
       returns the distance between the final location
       and the location at the start of the walk.
    """

    start = f.getLoc(d)
    for s in range(numSteps):
        f.moveDrunk(d)
    return start.distFrom(f.getLoc(d))
```

Let's Try It

The programs below use the classes Field, Location and UsualDrunk.

`Week12_BasicRandomWalkOne.py`: simulates 1 walk of 100 steps, and plot the final locations of the UsualDrunk, assuming (s)he starts from location 0,0.

`Week12_BasicRandomWalkTwo.py`: simulates 10 walks of 100 steps, and plot the final locations of the UsualDrunk, assuming (s)he starts from location 0,0.

`Week12_BasicRandomWalkThree.py`: simulates 5 UsualDrunks each walking 10 walks of 100 steps, and plot the final locations of each drunk, assuming they start from location 0,0.

Terms of Use

- This presentation was adapted from lecture materials provided in MIT Introduction to Computer Science and Programming in Python.
- Licenced under terms of Creative Commons License.