Program 2

Muaz Alhaider and Zakariya Ahmed

November 11, 2020

Contents

1 Modules

1.1 Transitional Probabilty

$$Transitional\ Probabilty(Pos_i, Dir) = (Pos_{from\ left} : Drift(Left), Pos_{from\ straight} : Drift(Straight),$$

$$Transitional\ Probabilty(Pos_i, Dir) = (Pos_{from\ left} : Drift(Left), Pos_{from\ straight} : Drift(Straight),$$

Left, *Right*, and *Straight* are all positions defiend in relation to what *Dir* is:
if *Dir* is EAST, then *Left* is SOUTH, *Right* is NORTH, and Straight is EAST.

Here, transitional probabilty has two forms: one which is all paths that converge to a point, and another where they diverge from a point. Prediction (as the name implies) wants all the possiple paths to a point, which is why we include $Pos_{from\ behind}$. Smoothing however, does not require that, which is why it's not include.

```
  @enum SquareType OPEN CLOSED
@enum Direction WEST NORTH EAST SOUTH
@enum DriftType STRAIGHT LEFT RIGHT
const Drift=Dict(STRAIGHT=>.7, LEFT=>.15, RIGHT=>.15)
const AllDirects =(WEST, NORTH, EAST, SOUTH)

   # Gets the transitional probabilty of posistions going into a point,
# or transitional probablites from the point
# the first is used for prediction, the other for smoothing
function transprob( grid::Array{Array{Float64,1},1}, pos::Tuple{Int64, Int64}, dir::Di
arr = []

function _gen_parts(straightDir::Direction , leftDir::Direction, rightDir::Direction, b
parent_pos = []
behind = move(pos, behindDir)
straight = move(pos, straightDir)
left = move(pos, leftDir)
right = move(pos, rightDir)


if( !notblocked(grid, straight)) # Bounce
push!(parent_pos, (pos,Drift[STRAIGHT]*grid[pos[1]][pos[2]], Drift[STRAIGHT] ))
elseif (getforward) # If this is smoothing, want the probabilty in front
push!(parent_pos, (straight,Drift[STRAIGHT]*grid[straight[1]][straight[2]], Drift[STRA
end
if( notblocked(grid, behind) && !getforward) # Prob of square behind current ot move to
push!(parent_pos, (behind,Drift[STRAIGHT]*grid[behind[1]][behind[2]], Drift[STRAIGHT]))
end
if(notblocked(grid, left)) # Get probabitly of the left pos coming to curretn
push!(parent_pos, (left,Drift[LEFT]*grid[left[1]][left[2]], Drift[LEFT] ))
else #Boucne from left
push!(parent_pos, (pos,Drift[LEFT]*grid[pos[1]][pos[2]], Drift[LEFT] ))
end
if(notblocked(grid, right)) #Get probabilty of right pos coming to current
```

```
push!(parent_pos, (right,Drift[RIGHT]*grid[right[1]][right[2]],Drift[RIGHT]))
else #Bounce
push!(parent_pos, (pos,Drift[RIGHT]*grid[pos[1]][pos[2]], Drift[RIGHT]))
end
end

if(dir==WEST)
arr=_gen_parts(WEST, SOUTH, NORTH, EAST, grid, pos)
end
if(dir==NORTH)
arr=_gen_parts(NORTH, EAST, WEST, SOUTH, grid, pos)
end
if(dir==SOUTH)
arr=_gen_parts(SOUTH, EAST, WEST, NORTH, grid, pos)
end
if(dir==EAST)
arr=_gen_parts(EAST, SOUTH, NORTH, WEST, grid, pos)
end
arr
end
```

1.2 Prediction

$$Prediction(Grid, Direction) = \left\{ pos_i \in Grid | \sum_{(Pos_j, DriftProb)}^{Transiton\ \ Probability(pos_i, direction)} DriftProb \cdot P(Pos_j) \right\}$$

$Prediction(Grid, Direction)$ (as the name implies) attempts to predict where the agent will be given previous infroamtion. It does this by transforming the grid by the equation $\sum_{(Pos_j, DriftProb)}^{Transiton\ \ Probability(pos_i, direction)} DriftProb \cdot P(Pos_j)$.This gets the probabilty of an agent drifting (or if direction is straight, accurantly going to) a point, and what is the probabtily the agent would be at the point Pos_j.

```
    # Prediction is the sum of possiple transitional probabilties
# that can reach pos_i, * P(pos).j
function predict(grid::Array{Array{Float64,1}}, dir::Direction)
tmp_grid = deepcopy(grid)
for row in 1:6
for col in 1:5
```

```
if(notblocked(grid, (row,col)))
val=sum([x[2] for x in transprob(grid, (row, col), dir)])
tmp_grid[row][col]=val
end
end
end
tmp_grid

end
```

1.3 Evidence Contional Probabilty

$$Evidence\ Contional\ Probability(Pos_i, Evidence) = \prod_{dir=W}^{\text{Directions}} Sense(evidence[pos_i dir], actual[pos_i + dir])$$

This is the equation we use to get the evidence contional probability: it's the product of each the evidecne's value at a direction times what's actaully in the value of the direction. So if *Left* has open, but evidecne says it's closed, it's 0.2. Taking the prodcut of all direction's sensed value and actual value, it will result in the Evidecne Contional Probabilty at Pos_i given $Evidence$

```
  # Given posistion and evidecne return probabilty of being in that posistion
const Sense=Dict(OPEN=>Dict(OPEN=>.8, CLOSED=>.2), CLOSED=>Dict(OPEN=>.25, CLOSED=>.75)

function evidence_Probabilty(grid::Array{Array{Float64,1},1}, pos::Tuple{Int64, Int64}
prod = 1
for i in 1:4
tmp_pos = move(pos,AllDirects[i])
block = notblocked(grid, tmp_pos)
if (block)
prod*= Sense[OPEN][evidence[i]]
else
prod*= Sense[CLOSED][evidence[i]]
end
end
prod
end
```

1.4 Filtering

$$pos_{s+1,i} = \frac{Evidecne\ Conditional\ Probabtily(pos_{s,i},evidecne)\cdot P(pos_{s,i})}{\sum_{pos}^{all\ posistions} Evidecne\ Conditional\ Probabtily(pos_{s,i},evidecne)\cdot P(pos_{s,i})}$$

$$Filtering(Grid, Evidence) =$$

$$\left\{pos_i \in Grid | \frac{P(pos_i)\cdot Evidence\ Conditional\ Probability(pos_i,Evidecne)}{\sum_{pos}^{all\ posistions} P(pos_i)\cdot Evidecne\ Conditional\ Probabtily(pos_i,evidecne)}\right\}$$

Filtering is a transformation upon the grid: each value gets transformed by the expression $\frac{P(pos_i)\cdot Evidence\ Conditional\ Probability(pos_i,Evidecne)}{\sum_{pos}^{all\ posistions} P(pos_i)\cdot Evidecne\ Conditional\ Probabtily(pos_i,evidecne)}$, whcih for purposes of making it easier to talk about, will be expressed as $Filter\ Step(pos_i, Evidence)$. $Filter\ Step$ is condtional probabilty of each point times what the point was previously, and then dividng it by the sum of all points on the grid. This operatoin is $O(n)$, although more accuratly it's $O(2n)$ because there's a minimal of iterating through each value twice.

```
   # Get the evidecne contional probabilty of each posistoin*Pos(s_i)
# Then divide each posistion with the evidnece conditonal probabilty
function filter(grid::Array{Array{Float64,1},1}, evidence::Tuple{SquareType, SquareType
tmp_grid = deepcopy(grid)
for row in 1:6
for col in 1:5
if(notblocked(grid, (row,col)))
tmp_grid[row][col]*=evidence_Probabilty(grid, (row, col), evidence)
end
end
end
total_sum = sum(sum(tmp_grid))
# println("SUM: ", total_sum)
tmp_grid / total_sum
end
```

1.5 Smoothing

$SmoothingPart(Grid, Previous, Direction, Evidence) =$

$\{pos_i \in Grid | P(Pos_i) \cdot \sum_{(Pos_j, DriftProb)}^{Transiton\ Probability(pos_i,direction)}$

$P(Pos_j)\cdot DriftProb\cdot Evidence\, Conditional\ Probabtily(Grid, Pos_j, Evidence)\}$

$$Smoothing(Grid, Previous, Direction, Evidence) = \left\{ pos_i \in Grid | \frac{SmoothPart(Grid,Previous,Direction,Evidence)}{\sum_{pos}^{\text{All Positons}} Smoothpart(Grid,Prevoius,Direction,Evidence)} \right\}$$

Smoothing invovles

```
   # Get the transitional probabilty of a point going OUT, not in
# an it's conditional probabilty, with it's inital probabilty
# returns 2 things: B at pos, and B*p(s)
function smoothpart( last_grid::Array{Array{Float64,1}}, Bgrid::Array{Array{Float64,1}}
parent_pos=transprob(grid, pos, dir, true )
x=0
# for i in parent_pos
for (tmp_pos, prob, drift) in parent_pos
# tmp_pos = i[1]
# prob = i[2]
# drift = i[3]
y=evidence_Probabilty(grid, tmp_pos, evidence)* Bgrid[tmp_pos[1]][tmp_pos[2]]* drift
x+=y
end

(x,   x *last_grid[pos[1]][pos[2]])
end

# Get the smoothing part for each posistion in grid
# Then divide the whoel grid by the sum of it's parts
function smooth( grid::Array{Array{Float64,1}}, last_grid::Array{Array{Float64,1}}, Bgr
SP = deepcopy(grid)
B = deepcopy(Bgrid)
for row in 1:6
for col in 1:5
if(notblocked(grid, (row,col)))
val=smoothpart(last_grid, Bgrid, evidence, dir, (row,col))
B[row][col] = val[1]
SP[row][col] = val[2]
else
B[row][col]  = 0
SP[row][col] = 0
end
end
end
```

```
# println("SUM: ", sum(sum(SP)))
SP/=sum(sum(SP))
# print_grid(SP); println(); print_grid(B); println()
(SP, B)
end
```

2 Results

The code outputs the following:

```
julia SUBMIT.jl
```

You see second smoothing, it's different from the report, which is

```
Smoothing: last 2nd location
  3.08    3.03    2.43    1.97    0.07
  2.21    ####    ####    0.74    0.16
  0.14    ####    9.83   58.17    0.10
  0.02    ####    ####    0.41    0.04
  0.02    ####    2.27    4.63    0.05
  0.59    2.20    7.30    0.53    0.02
```

Trying to piece how this happened, I found that second smoothing step, if I keep the direction as north it will result

```
Second Last posistion smoothing with Evidence [1, 1, 0, 1] And north (INCORRECT)
3.08 3.03 2.43 1.97 0.07
2.21 #### #### 0.74 0.16
0.14 #### 9.83 58.17 0.1
0.02 #### #### 0.41 0.04
0.02 #### 2.27 4.63 0.05
0.59 2.2 7.3 0.53 0.02
```

Which leads to the bug in the anncoement: the last action never changes: it stays to the north. My code on the other hand, changes the action, in accordance to how smoothign is suposed to be done.

3 Screenshots