Agent Based Modeling of Crowd Behavior in an Evacuation Scenario

Josh Zukoff (Colgate University), Dr. Cliff Behrens (Telcordia)

July 18, 2011

Abstract

My research involved the utilization of Agent Based Modeling (ABM) tools to simulate a quasi-realistic evacuation scenario in a built space. The space took the form of a stadium and the model attempted to take into account psychological aspects like group formation and social comparison to accurately represent crowd behavior. Crowd behavior in built spaces is of interest to a number of types of individuals ie. urban planners, emergency personell. Similarly, agent based modeling can adapt itself to the simulation of a vast array of scenarios. My model was written to take advantage of flocking behavior to simulate grouping behavior. The model illustrates the ability for ABM to approximate crowd behavior using low level decision making processes. Additionally various aspects of communication among people can be shown to have an affect on evacuation time.

1 Introduction

1.1 Intro to ABM

Agent based modeling is a tool/system for modeling and simulating anything which can be represented by the interactions of agents in an environment. The building blocks of ABM are agents and patches. Agents are endowed with characterstics and decision making processes. Each agent can be entirely unique and follow a unique set of rules. Patches form a grid that make up the environment or "map". Patches also can have their own characteristics but unlike agents, patches cannot move. Various "breeds" can be created to have different default characteristics but for my model this was not necessary. The program I used was Netlogo which is written and maintained by Uri Willensky. Netlogo is an all in one development environment for ABM that uses the logo programming language for coding and has an attractive GUI for running simulations.

1.2 Overview of Modeling Crowd Behavior

Crowd behavior in built spaces must take into account a few aspects. First is that the built space affects the actions taken by the individuals and crowd. Second, crowds are made up of individuals with their own goals and characteristics.

People do not behave like particles, rather they are governed by psychological aspects and behave intelligently. For this reason, abm seems ideal. Individuals can be endowed with their own decision making processes and localized interactions among them can bring out a global emergent behavior. Crowds form as a result of these localized behaviors.

Flocking aims to represent the flocking of birds or fish and is achieved using three methods: Separation tells an agent to avoid neighboars and “separate from them”, Alignment sets the heading of an agent to the average heading of its flockmates or groupmates, and Cohesion brings agents in a flock closer together. These simple processes can bring about interesting behavior. By building in obstacle avoidance and grouping based on certain factors, flocking can be made to be a reasonable representation of group formation and travel.

2 Results

3 Algorithm/Code

```
globals [middle rightExit]
turtles-own [

  knowledge
  team
  leaderTurtle
  closestAisle
  reachAisle
  reachConcourse
  leadership
  reachExit
  closestExit
  height
  weight
  age
  sex
  child
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
  groupLeader
  expandGroup
  parkingLot
  family
  ]


patches-own [ptype pheight exit exitNumber]
to setup
```

```
clear-all
ask patches [set pcolor violet]
import-pcolors nameMap
set rightExit 0
set middle patch (max-pxcor / 2) (max-pycor / 2)
ifelse teams = false [ask patches with [
    shade-of? pcolor turquoise or shade-of? pcolor cyan
        or shade-of? pcolor sky] [
    set ptype ”seat” if (round (distance middle) mod (
       rowDensity) = 0   ) [
      sprout 1 [
        set parkingLot one-of [0 1 2 3] set family
            turtles-here set expandGroup true
        set shape ”person” set groupLeader false set
            knowledge 0 getAttributes
        set reachExit false set closestExit Nobody set
            leaderTurtle Nobody
        set reachConcourse false set closestAisle
            Nobody set reachAisle false
        set flockmates turtles-here set size 3 set team
             color] ]]] [
  ask patches with [
    shade-of? pcolor turquoise or shade-of? pcolor cyan
        or shade-of? pcolor sky][
     set ptype ”seat” if (round (distance middle) mod (
        rowDensity) = 0
       and pycor > (max-pycor / 2 )) [
       sprout 1 [
         set parkingLot one-of [0 1 2 3] set family
             turtles-here set expandGroup true
         set shape ”person” set groupLeader false set
             knowledge 0 getAttributes
         set reachExit false set closestExit Nobody set
              leaderTurtle Nobody
         set reachConcourse false set closestAisle
             Nobody set flockmates turtles-here
         set reachAisle false set size 3 set color
             orange set team color ] ]]
  ask patches with [
    shade-of? pcolor turquoise or shade-of? pcolor cyan
        or shade-of? pcolor sky][
     set ptype ”seat” if (round (distance middle) mod (
        rowDensity) = 0
       and pycor < (max-pycor / 2) ) [
       sprout 1 [
         set parkingLot one-of [0 1 2 3] set family
             turtles-here set expandGroup true
         set shape ”person” set groupLeader false set
             knowledge 0 getAttributes
         set reachExit false set closestExit Nobody set
              leaderTurtle Nobody
         set reachConcourse false set closestAisle
             Nobody set flockmates turtles-here
         set reachAisle false set size 3 set color
             black set team color] ]]]
ask patches with [
  shade-of? pcolor yellow] [
  set ptype ”aisle” set pheight distance middle]
ask patches with [
  shade-of? pcolor green or shade-of? pcolor lime] [
  set ptype ”field” set pheight 0]
ask patches with [
  shade-of? pcolor gray] [
  set ptype ”concourse” set pheight 300]
ask patches with [
  shade-of? pcolor red or shade-of? pcolor magenta or
     shade-of? pcolor pink] [
  set ptype ”wall”]
ask patches with [
  shade-of? pcolor orange] [
  set ptype ”upperLevel”]
ask patches with [
  pcolor = white] [
  set ptype ”exit”]
ask patches with [
  pcolor = 0] [
  set ptype ”nothing”]
ask patches with [
  pcolor = violet] [
  set ptype ”outside”]
while [count patches with [ptype = 0] > 0] [
  ask patches with [ptype = 0] [
    ask one-of patches in-radius 1 [
      let tempType ptype let tempColor pcolor ask
          myself[
        set ptype tempType set pcolor tempColor]]]]

if seatDensity != 50 [ask n-of ((count Turtles) /
   seatDensity) turtles [die]]

ask patches with [ptype = ”exit”] [ifelse (pxcor < 36
    and pycor > 36) [set exitNumber 0] [
     ifelse (pxcor > 36 and pycor < 36) [set exitNumber
        1] [
       ifelse (pxcor > 36 and pycor > 280) [
         set exitNumber 2] [set exitNumber 3]]]]
;   if globalNotification = true [
;     ask patches with [ptype = ”exit”] [
;       ask turtles in-radius 150 [
;         set closestExit myself]]]

if globalNotification = true [
  ask turtles [set closestExit ([one-of patches
      in-radius 5] of
       min-one-of patches with [ptype = ”exit”] [
           distance myself]) ]]

if police = true [
  ask n-of policeCount (patches with [ptype = ”
      concourse”
       and not any? patches in-radius 2 with [ptype != ”
           concourse”] ]) [
       ask patches in-radius 1 with [ptype = ”concourse”
           ] [set ptype ”police”
         set pcolor blue set exit (min-one-of patches
             with [ptype = ”exit”] [
              distance myself])]]]

ask turtles [let famSize random 5 if count turtles
    in-radius 5 >= famsize [
     set family n-of  famSize turtles in-radius 6 ask
         family [
       set family [family] of myself set parkingLot [
           parkingLot] of myself ]]]
ask turtles [set family (turtle-set turtles-here family
    ) set flockmates family]
end

to getAttributes
;0 is male, 1 is female
set sex random 2
;assumes roughly two adults to every child at the
    stadium
set child (random 3 < 2)
ifelse child = false [
  set age 2 + abs round random-normal 40 15
  ifelse sex = 0 [
      set height 2 + abs round random-normal 69.5 3] [
      set height 2 + abs round random-normal 64 3 ]][
      set age 2 + abs round random-normal 10 5
      ifelse sex = 0 [
        set height 2 + abs round random-normal 50 4][
        set height 2 + abs round random-normal 47 4]]
  millerFormula
end

to penOn
  ask turtles[pen-down]
end

to penOff
  ask turtles[pen-up]
end

;replace with classification tree: leader, peer, no
   relation
to-report socialComparison [agent1 agent2]
  let numericalComparison 0
  if [color] of agent1 != [color] of agent2 [set
     numericalComparison (numericalComparison - 30)]
  if [child] of agent1 = [child] of agent2 [set
     numericalComparison (numericalComparison + 30)]
  if [sex] of agent1 = [sex] of agent2 [set
     numericalComparison (numericalComparison + 30)]
  set numericalComparison (numericalComparison + (20 -
     abs (.25 * ([height] of agent1 - [height]
         of agent2))))
  set numericalComparison (numericalComparison + (20 -
     abs (.25 * ([weight] of agent1 - [weight]
          of agent2))))
  set numericalComparison (numericalComparison + (20 -
     abs (.5 * ([age] of agent1 - [age]
          of agent2))))
  report numericalComparison
end

to go
  ;if ticks = 250 [set preferExit false]
  if count turtles = 0 [stop]
  tick
  if (ticks mod upperLevelRate) = 0 [
    ask n-of upperLevelQuantity patches with [
      ptype = ”upperLevel” and count turtles-here = 0] [
      sprout 1 [
        set parkingLot one-of [0 1 2 3] set family
           turtles-here set expandGroup true
        set shape ”person” getAttributes set flockmates
           turtles-here set groupLeader false
        set reachExit false set closestExit Nobody set
           leaderTurtle Nobody
        set reachConcourse true set closestAisle Nobody
           set reachAisle true
        set size 3 set color (one-of list orange black)
           set team color]]]
  ask turtles [ ifelse reachAisle = false [
          if closestAisle = Nobody and leaderTurtle =
             Nobody[
            locateAisle] moveToAisle] [
            ifelse reachConcourse = false [
              moveToConcourse] [
              escapeBuilding ]]]


end

to millerFormula
  ;Miller formula for adult weight
  ;Luscombe weight formula for estimating children's
     weight
  ifelse child = false or age > 13[
    let tempHeight height ifelse sex = 0 [
      ifelse height >= 60 [
        set weight (weight + 124)
        set tempHeight (tempHeight - 60)
        set weight (tempHeight * 3)][set weight 124]][
      ifelse height >= 60 [
        set weight (weight + 117) set tempHeight (
           tempHeight - 60) set weight (tempHeight * 3)][
        set weight 117]]][let weightkg ((3 * age) + 7)
           set weight (weightkg * 2.2)]


end
; making it so that height must be positive allows me to
    ”disable” an aisle by setting height to 0 or negative
to locateAisle
  let aisle (min-one-of patches in-radius 10 with [ptype
     = ”aisle” and pheight > 0] [
      distance myself])
  ifelse aisle = Nobody [
    set leaderTurtle one-of turtles with [distance myself
        < 10 and closestAisle != Nobody]
    if leaderTurtle != Nobody [
      let a Nobody ask leaderTurtle [
        set a closestAisle]
      set closestAisle a
      set leaderTurtle Nobody]] [set closestAisle aisle
         set leaderTurtle Nobody ]

end

to moveToAisle
  ifelse closestAisle = Nobody [ set heading (random 360)
      locateAisle
      jump 1] [face closestAisle jump 1]
  if collisions = true [
    if count turtles-here > 1 [jump -1 locateAisle]]
  let dest one-of patches in-radius 2 with [ptype = "
     aisle" and count turtles-here = 0]
  if dest != nobody [move-to dest]
  if ptype = "aisle" [set reachAisle true]
end

to moveToConcourse
  ;field is included to take into account a fiew patches
     that wind up being shades of green unintentionally
  let nextStep max-one-of patches in-radius 3 with [ptype
      = "aisle"
    or ptype = "concourse" or ptype = "field"] [pheight]
  if nextStep != Nobody [
  face nextStep ]
  jump 1
  if collisions = true [
    if count turtles-here > 1 [jump -1 set heading (
       heading + random 45)] ]
  let dest one-of patches in-radius 2 with [ptype = "
     concourse" and count turtles-here = 0]
  if dest != nobody [move-to dest]
  if ptype = "concourse" [set reachConcourse true]
end

to locateExit
; 70 is typical human cone of vision

  set closestExit (one-of patches in-cone 50 70 with [
      ptype = ”exit” and (preferExit = false or
         exitNumber = [parkingLot] of myself)])
  if closestExit = nobody and any? patches in-cone 10 70
     with [ptype = ”police”] [
    ifelse preferExit = false [set closestExit [exit] of
       one-of patches in-cone 10 70 with [
        ptype = ”police”]][
      if any? patches in-cone 10 70 with [ptype = ”police
         ” and exitNumber = [
          parkingLot] of myself] [set closestExit [exit]
             of one-of patches in-cone 10 70 with [
          ptype = ”police” and exitNumber = [parkingLot]
             of myself]]]]

  if closestExit != Nobody [let temp nobody
      ask closestExit  [set temp one-of patches in-radius
         8 with [ptype = ”exit”]]
      set closestExit temp
      face closestExit]

end

to learnExit
  if learningType = ”local” [
  if any? turtles in-radius leaderVolume with [
     closestExit != nobody and (preferExit = false
     or parkingLot = [parkingLot] of myself)][
      ifelse preferExit = false [ set closestExit [
         closestExit]
        of one-of turtles in-radius leaderVolume with [
           closestExit != nobody]] [
        set closestExit [closestExit] of (one-of turtles
           in-radius leaderVolume
          with [closestExit != nobody and parkingLot = [
             parkingLot] of myself])]
  let temp nobody
  ask closestExit  [set temp one-of patches in-radius 5
     with [ptype = ”exit”]]
  set closestExit temp
  ]]

  if learningType = ”flockmates” [
  if any? flockmates with [groupLeader = true] [if
     distance one-of flockmates with [
     groupLeader = true] < leaderVolume [
  set closestExit [closestExit] of one-of flockmates with
```

```
    [groupLeader = true]]  ]
  ; nobody in group knows, find out from people in small
     area
  if groupLeader = true [if closestExit = nobody [
    ifelse preferExit = false [if any? turtles in-radius
       leaderVolume with [closestExit != nobody][
      set closestExit [closestExit] of (one-of turtles
          in-radius leaderVolume with [
           closestExit != nobody])]] [if
      any? turtles in-radius leaderVolume with [
          closestExit != nobody and parkingLot = [
           parkingLot] of myself][
      set closestExit [closestExit] of (one-of turtles
          in-radius leaderVolume with [
        closestExit != nobody and parkingLot = [
            parkingLot] of myself])]]
       ]]
  ]
  if closestExit = nobody and any? patches in-cone 10 70
     with [ptype = ”police”] [
    ifelse preferExit = false [set closestExit [exit] of
       one-of patches in-cone 10 70 with [
        ptype = ”police”]][
      if any? patches in-cone 10 70 with [ptype = ”police
          ” and exitNumber = [
           parkingLot] of myself] [set closestExit [exit]
               of one-of patches in-cone 10 70 with [
           ptype = ”police” and exitNumber = [parkingLot]
               of myself]]]]

  if closestExit != Nobody [let temp nobody
      ask closestExit  [set temp one-of patches in-radius
          8 with [ptype = ”exit”]]
      set closestExit temp
      face closestExit]

end



to escapeBuilding

;if count flockmates = 0 [set flockmates turtles-here]

;if not any? flockmates [set heading (heading + random
    (60) - random 120)]
```

```
if closestExit = nobody and flockCommunication = true and
    any? flockmates with [
  groupLeader = true and closestExit != nobody] [set
     closestExit [closestExit]
  of one-of flockmates with [groupLeader = true and
     closestExit != nobody]]
if closestExit = nobody and learningType != ”none” [
   learnExit]
if closestExit = Nobody and groupLeader = true [
   locateExit]
;if closestExit != Nobody [ face (closestExit)
 ; if distance closestExit <= 15 [set heading ( heading +
     random 50 - random 100)]
  ;ask other flockmates [set heading [heading] of myself
     + random 50 - random 100]]

flock
let safe avoid_obstacles
ifelse safe = true [fd 1][ifelse closestExit != nobody [
   face closestExit
     set heading (heading + random 30 - random 40)] [
     set heading (heading + random 30 - random 40)]
  if any? patches in-cone 2 150 with [count turtles-here
     = 0
    and (ptype = ”concourse” or (preferExit = false and
       ptype = ”exit”
         or (ptype = ”exit” and exitNumber = [parkingLot]
             of myself))) ] [
    move-to one-of patches in-cone 2 150 with [count
       turtles-here = 0
      and (ptype = ”concourse” or (preferExit = false and
          ptype = ”exit”
         or (ptype = ”exit” and exitNumber = [parkingLot
            ] of myself))) ]]]
;fd 1
if any? patches in-radius 3 with [ptype = ”exit”] [if [
   exitNumber]
  of one-of patches in-radius 3 with [ptype = ”exit”] =
     parkingLot
  and preferExit = true [set rightExit (rightExit + 1)]
     set reachExit true die]

end

to flock  ;; turtle procedure
```

```
  ;flockmates too far away, find new flockmates

  ;if expandGroup = true [if random 10 < 2 [ask
     flockmates [set expandGroup false]]]
  if count flockmates > maxGroupSize [ask flockmates [set
      expandGroup false]]
  if count flockmates < maxGroupSize and expandGroup =
     true [find-flockmates]


  if not any? flockmates with [groupLeader = true] [
     locateExit]

  if any? other flockmates
    [ find-nearest-neighbor
      ifelse distance nearest-neighbor <= 1
        [ separate ]
        [ align
          cohere ] ]

  if leadershipFunction > 70 [makeLeader]

end
to makeLeader
  set groupLeader false
  ask flockmates [set groupLeader false]
  ask max-one-of flockmates [leadershipFunction][set
     groupLeader true]
end

to-report leadershipFunction
  let leaderValue 0
  if child = false [set leaderValue (leaderValue + 50)]
  set leaderValue (leaderValue + .5 * age)
  set leaderValue (leaderValue + .5 * weight)
  if count flockmates = 1 [set leaderValue 0]
  report leaderValue
end

to find-flockmates  ;; turtle procedure

 set flockmates turtles in-radius (maxGroupSize)  with [
   ;socialComparison self myself > 20 and
   reachConcourse = true and expandGroup = true and (
      preferExit = false or parkingLot = [
       parkingLot] of myself)]
 ask flockmates [set flockmates [flockmates] of myself]
 if count flockmates with [groupLeader = true] > 1 [ask
    flockmates [set groupLeader false]]
end

to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance
      myself]
end

;;; SEPARATE

to separate  ;; turtle procedure
  turn-away ([heading] of nearest-neighbor) 4.5
end

;;; ALIGN

to align  ;; turtle procedure
  turn-towards average-flockmate-heading  1
end

to-report average-flockmate-heading  ;; turtle procedure
  ;; We can't just average the heading variables here.
  ;; For example, the average of 1 and 359 should be 0,
  ;; not 180.  So we have to use trigonometry.
  let x-component sum [sin heading] of flockmates
  let y-component sum [cos heading] of flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; COHERE

to cohere  ;; turtle procedure
  turn-towards average-heading-towards-flockmates 1
end

to-report average-heading-towards-flockmates  ;; turtle
    procedure
  ;; ”towards myself” gives us the heading from the other
       turtle
  ;; to me, but we want the heading from me to the other
      turtle,
  ;; so we add 180
  let x-component mean [sin (towards myself + 180)] of
      flockmates
  let y-component mean [cos (towards myself + 180)] of
      flockmates
  ifelse x-component = 0 and y-component = 0
    [ report heading ]
    [ report atan x-component y-component ]
end

;;; HELPER PROCEDURES

to turn-towards [new-heading max-turn]  ;; turtle
    procedure
  turn-at-most (subtract-headings new-heading heading)
      max-turn
end

to turn-away [new-heading max-turn]  ;; turtle procedure
  turn-at-most (subtract-headings heading new-heading)
      max-turn
end

;; turn right by "turn" degrees (or left if "turn" is
    negative),
;; but never turn more than "max-turn" degrees
to turn-at-most [turn max-turn]  ;; turtle procedure
  ifelse abs turn > max-turn
    [ ifelse turn > 0
        [ rt max-turn ]
        [ lt max-turn ] ]
    [ rt turn ]
end

to-report avoid_obstacles
    let i 0
    while [(([ptype] of patch-ahead i = "concourse" or ([
        ptype]
           of patch-ahead i = "exit" and (preferExit =
               false or [exitNumber]
             of patch-ahead i = parkingLot )) and i <= 10)
                ]
    [set i (i + 1) ]
    if ([ptype] of patch-ahead i != "concourse" and ([
        ptype] of patch-ahead i != "exit"
          or (preferExit = true and [ptype] of patch-ahead
              i = "exit" and [exitNumber]
```

```
            of patch-ahead i != parkingLot)))
     [
         if [ptype] of patch-at-heading-and-distance (
            heading - 5) (i + 1) != ”concourse”
         and ([ptype] of patch-at-heading-and-distance (
            heading - 5) (i + 1) != ”exit”
           or (preferExit = true and [ptype]
             of patch-at-heading-and-distance (heading -
                 5) (i + 1) = ”exit”
             and [exitNumber] of
                 patch-at-heading-and-distance (heading -
                 5) (i + 1) != parkingLot))
       [
             ifelse [ptype] of
                 patch-at-heading-and-distance (heading +
                 5) (i + 1) != ”concourse”
             and ([ptype] of patch-at-heading-and-distance
                  (heading + 5) (i + 1) != ”exit”
               or (preferExit = true and [ptype]
                  of patch-at-heading-and-distance (heading
                      + 5) (i + 1) = ”exit”
                 and [exitNumber] of
                     patch-at-heading-and-distance (heading
                      + 5) (i + 1)
                 != parkingLot))
             [
                  ifelse random 1 = 0
                  [ set i 0 rt 40 ]
                  [ set i 0 lt 40 ]
             ]
             [ifelse heading <= 180 and [ptype]
               of patch-at-heading-and-distance (heading -
                    5 ) ( i + 1 ) = ”wall” [
                  set i 0 rt 60 ][set i 0 lt 60]]
         ]
    ]
    ifelse count [other turtles-here] of patch-ahead 1 = 0
   and ([ptype] of patch-ahead 1 = ”concourse” or (
       preferExit = true and [ptype]
        of patch-ahead 1 = ”exit” and [exitNumber] of
           patch-ahead 1 = parkingLot)) [
        report true] [report false]
end
```

4 Conclusion

[1]

References

[1] Sam Hopkins, *Could minds be (anything like) machines?*