```
;Evacuation Efficency (v.1.6.3 LAMBDA)
;Team SFHS-2 by Micah Sulich, Daniel Onstott, Luke Shankin, Lileigh Thomas, and Marisa Tedori
;Made in netlogo v6.0

globals [escaped alex_escaped audrey_escaped reg_escaped injured]
;the more agressive agent set
breed [alex alexes]
;the more submissive agent set
breed [audrey audreys]
;the middle of the two
breed [andy andys]
to clear
  clear-all
end
to setup
  ifelse sprout? = true
  [
      ;reset graph
  set escaped 0
  set alex_escaped 0
  set audrey_escaped 0
  set reg_escaped 0
  set injured 0
  clear-turtles
  ;finds patches that are within the building to spawn from
  ask n-of numb (patches with ([pcolor != red and pcolor != green and pxcor > -13 and pxcor < 15]))
  [
    ;coinflip for regular andy
    ifelse (random(2)) = 0
    [
    sprout 1
    [set color yellow
     set breed andy
    set shape "square"]
    ][
     ;further coinflip o divide remaining andy
     ifelse (random(2)) = 0 [
     sprout 1
    [set color blue
     if color_toggle = false
      [
        set color yellow
      ]
     set breed alex
    set shape "square"]
    ][
      sprout 1
    [set color orange
      if color_toggle = false
      [
        set color yellow
      ]

     set breed audrey
    set shape "square"]
    ]
    ]
  ]

  ]
  ;---sprout off------------
  [
  set escaped 0
  set alex_escaped 0
  set audrey_escaped 0
  clear-turtles
  if andy_on = true
  [
    create-andy numb
    [

      setxy (((-12) + 2) + random (26)) (((-12) + (-2)) + random
(29))
      set color yellow

  ]
  ]

  if alex_on = true
  [
    create-alex numb_alex
    [

    setxy (((-12) + 2) + random (26)) (((-12) + (-2)) + random (29))
    set color blue

    ]
  ]

if audrey_on = true
  [
  create-audrey numb_audrey
  [
    setxy (((-12) + 2) + random (26)) (((-12) + (-2)) + random (29))
    set color orange

  ]
  ]
  ]
  reset-ticks
  setup-plots
end


to bounce_alex
   ;The bounce function is the dictates how the agents interact with
eachother.
; determines if the agent goes up or down
  if [pcolor] of patch-at dx 1 = red
  [
    if ycor > (-1 * box)[set heading (180)]
    if ycor < (-1 * box)[set heading (90)]
  ]
if not any? other audrey-on patch-ahead 1 = false
  [

    set heading (heading - 180)
    fd 1.2
    set heading(heading + 79)
    set heading(heading - 259)

  ]



end

to bounce_audrey
  ;The bounce function is the dictates how the agents interact with
each other.
 ; In the case of this breed, if a red wall blocks its path, it will
turn towards the door and try to move closer to it until it can move
around the obstacle.
 ;When coming into contact with another agent, It will turn around to
find a clear path before trying to go towards the door again.
 ;In this sense the agent “waits” and “pushes” its way around other
agents.

  let flip (random 2)
; determines if the agent goes up or down
  if [pcolor] of patch-at dx 1 = red
```

```
[
  if ycor > (-1 * box)[set heading (180)]
  if ycor < (-1 * box)[set heading (90)]
]

if not any? other alex-on patch-ahead 1 = false
[
  if flip = 1
  [

  set heading (heading - 180)
  fd .5
  set heading(heading + 85)
  set heading(heading - 259)

  ]

  if flip = 2
  [

  set heading (heading + 180)
  fd .5
  set heading(heading - 85)
  set heading(heading + 259)

  ]

]

if not any? other andy-on patch-ahead 1 = false
[
  if flip = 1
  [

  set heading (heading - 180)
  fd .75
  set heading(heading + 85)
  set heading(heading - 259)

  ]

  if flip = 2
  [

  set heading (heading + 180)
  fd .75
  set heading(heading - 85)
  set heading(heading + 259)
```

```
    ]

  ]

  if not any? other audrey-on patch-ahead 1 = false
  [
    if flip = 1
    [

    set heading (heading - 180)
    fd .9
    set heading(heading + 85)
    set heading(heading - 259)

    ]

    if flip = 2
    [

    set heading (heading + 180)
    fd .9
    set heading(heading - 85)
    set heading(heading + 259)

    ]

  ]

end

to bounce_andy
  ;The bounce function is the dictates how the agents interact with
eachother
  let flip (random 2)
; determines if the agent goes up or down
  if [pcolor] of patch-at dx 1 = red
  [
    if ycor > (-1 * box)[set heading (180)]
    if ycor < (-1 * box)[set heading (90)]
  ]

  if not any? other turtles-on patch-ahead 1 = false
  [
    if flip = 1
    [

    set heading (heading - 180)
    fd 1
    set heading(heading + 85)
    set heading(heading - 259)

    ]

    if flip = 2
    [

    set heading (heading + 180)
    fd 1
    set heading(heading - 85)
    set heading(heading + 259)

    ]

  ]

end
to makebox

  ;made by Micah
  ;SETTING UP THE SCHOOL and classroom enviroment
  clear-patches

  ;origin points
  let origin_x (-12)
  let origin_y (box)

  ;top room vars
  let top_room_1 (1 + box)
  let top_room_2 (1 + box)
  let top_room_3 (1 + box)

  ;hallway length vars
  let hall_leng_1 (-12)
  let hall_leng_2 (-12)
  let hall_leng_3 (-12)
  let hall_leng_4 (-12)

  ;bottom room vars
  let bt_room_1 (1 + box)
  let bt_room_2 (1 + box)
  let bt_room_3 (1 + box)

  ;back wall
  let bk_wall_1 (16)
  let bk_wall_2 (-16)
  let bk_wall_3 (-16)


  ;making the front door/building edge
  ask (patch (origin_x) (-1) )
    [
    set pcolor green
    ]
  ask (patch (origin_x) (0) )
    [
    set pcolor green
    ]ask (patch (origin_x) (1) )
    [
    set pcolor green
    ]

    ask (patch (origin_x) (-2) )
    [
    set pcolor red
    ]
    ask (patch (origin_x) (-3) )
    [
    set pcolor red
    ]
    ask (patch (origin_x) (-4) )
    [
    set pcolor red
    ]

    ask (patch (origin_x) (2) )
    [
    set pcolor red
    ]
    ask (patch (origin_x) (3) )
    [
    set pcolor red
    ]
    ask (patch (origin_x) (4) )
    [
    set pcolor red
    ]

  ;TOP HALLWAY WALLs and doors
  while [hall_leng_1 < (17)]
  [
    ask (patch (hall_leng_1) (origin_y) )
    [
    set pcolor red
```

```
    ]

    set hall_leng_1 (hall_leng_1 + 1)
  ]
  while [hall_leng_3 < (17)]
  [
    ask (patch (hall_leng_3) (16) )
    [
    set pcolor red
    ]
    set hall_leng_3 (hall_leng_3 + 1)
  ]

     ;Making the upper hallway door gaps
  ask (patch (0) (origin_y) )
    [
    set pcolor black
    ]
   ask (patch (-1) (origin_y) )
    [
    set pcolor black
    ]

   ask (patch (-11) (origin_y) )
    [
    set pcolor black
    ]
   ask (patch (-10) (origin_y) )
    [
    set pcolor black
    ]
   ask (patch (9) (origin_y) )
    [
    set pcolor black
    ]
   ask (patch (10) (origin_y) )
    [
    set pcolor black
    ]

  ;LOWER HALLWAY WALL
  while [hall_leng_2 < (17)]
  [
    ask (patch (hall_leng_2) (-1 * origin_y) )
    [
    set pcolor red
    ]
    set hall_leng_2 (hall_leng_2 + 1)
  ]
  while [hall_leng_4 < (17)]
  [
    ask (patch (hall_leng_4) (-16) )
    [
    set pcolor red
    ]
    set hall_leng_4 (hall_leng_4 + 1)
  ]
     ;Making the lower hallway door gaps
  ask (patch (-3) (-1 * origin_y) )
    [
    set pcolor black
    ]
   ask (patch (-4) (-1 * origin_y) )
    [
    set pcolor black
    ]
   ask (patch (6) (-1 * origin_y) )
    [
    set pcolor black
    ]
   ask (patch (7) (-1 * origin_y) )
    [
    set pcolor black
    ]
   ask (patch (15) (-1 * origin_y) )
    [
    set pcolor black
    ]
   ask (patch (14) (-1 * origin_y) )
    [
    set pcolor black
    ]

;THIS CODE MAKES the top rooms
 while [top_room_1 < (17)]
  [
    ask (patch  (origin_x) (top_room_1))
    [
    set pcolor red
    ]
    set top_room_1 (top_room_1 + 1)
  ]

 while [top_room_3 < (17)]
  [
    ask (patch  (origin_x + 10) (top_room_3))
    [
    set pcolor red
    ]
    set top_room_3 (top_room_3 + 1)
  ]

 while [top_room_2 < (17)]
  [
    ask (patch  (origin_x + 20) (top_room_2))
    [
    set pcolor red
    ]
    set top_room_2 (top_room_2 + 1)
  ]

 ;this code makes the bottom rooms
 while [bt_room_1 < (17)]
  [
    ask (patch  (origin_x) (-1 * bt_room_1))
    [
    set pcolor red
    ]
    set bt_room_1 (bt_room_1 + 1)
    ]
 while [bt_room_3 < (17)]
  [
    ask (patch  (origin_x + 10) (-1 * bt_room_3))
    [
    set pcolor red
    ]
    set bt_room_3 (bt_room_3 + 1)
  ]
 while [bt_room_2 < (17)]
  [
    ask (patch  (origin_x + 20) (-1 * bt_room_2))
    [
    set pcolor red
    ]
    set bt_room_2 (bt_room_2 + 1)
  ]

 while [bk_wall_1 > (0)]
 [
   ask patch (16) (bk_wall_1)
   [
     set pcolor red
   ]
   set bk_wall_1 (bk_wall_1 - 1)
 ]
 while [bk_wall_2 < (0)]
 [
   ask patch (16) (bk_wall_2)
   [
     set pcolor red
   ]
   set bk_wall_2 (bk_wall_2 + 1)
 ]
 ask (patch (16) (0) )
    [
    set pcolor red
    ]
  while [bt_room_1 < (17)]
  [
    ask (patch  (origin_x) (-1 * bt_room_1))
    [
    set pcolor red
    ]
    set bt_room_1 (bt_room_1 + 1)
    ]

;this code inpart prevents the backwall bug
let bugfix1 (-16)
while [bugfix1 < (17)]
[
ask (patch  (17) (bugfix1))
    [
    set pcolor red
    ]
set bugfix1 (bugfix1 + 1)
]

end
to fix_error_alex
  ;These following "fix error" procedures fix an error that resulted in the turtles going too far into the wall and the program crashing.
  ask alex
  [
  if xcor > 17
  [
    set heading (180)
    fd 1
  ]
  ]

end
```

```
to fix_error_audrey
  ask audrey
  [
  if xcor > 17
  [
    set heading (180)
    fd 1
  ]
  ]

end
to fix_error_andy
  ask andy
  [
  if xcor > 17
  [
    set heading (180)
    fd 1
  ]
  ]

end

to go
  reset-timer

  ;origin points
  let origin_x (-12)
  let origin_y (box)
  ;the likeleyhood of an alex injuring the orther agents
  let injury_chance_alex1 ( numb * .06)
  let injury_chance_alex2 ( numb * .03)
  let injury_chance_alex3 ( numb * .01)




ask alex
  [
    if injurys = true
    [
      ;audrey 6%
      if injury_chance_alex1 > random(numb)
      [
        ask audrey-here [set color 27]
      ]
      ;andy 3%
      if injury_chance_alex2 > random(numb)
      [
        ask andy-here [set color 47]
      ]
      ;alex 1%
      if injury_chance_alex3 > random(numb)
      [
        ask alex-here [set color 95]
      ]

    ]
      set heading towards patch (origin_x) ((0) + 1)
      bounce_alex
      fix_error_alex
      ifelse color != 27
      ;This set of commands dictates the movement of the healthy
Alexes while the second one dictates how they move when injured
      [
       fd 1.5
       ; this algorithm makes the Alex disaper as they go through the
end door, simulating their escape.
       if xcor <= (origin_x) + .5
       [
         if xcor >= (origin_x) - .5
         [
           if ycor <= ((0) + 1.5)
           [
             if ycor >= ((0) + .5)
             [
               set escaped (escaped + 1)
               set alex_escaped (alex_escaped + 1)
               die
             ]
           ]
         ]
       ]
      ]
      ;if the Alexes are injured, they go 50% of their movement
speed.
      [
       fd .7
      ; this algorithm makes the andy disaper as they go through the
end door, simulating their escape.
       if xcor <= (origin_x) + .5
       [
         if xcor >= (origin_x) - .5
         [
           if ycor <= ((0) + 1.5)
           [
             if ycor >= ((0) + .5)
             [
               set escaped (escaped + 1)
               set alex_escaped (alex_escaped + 1)
               set injured (injured + 1)
               die
             ]
           ]
         ]
       ]
      ]
  ]

ask audrey
  [

      set heading towards patch (origin_x) ((0) + 1)
      bounce_audrey
      fix_error_audrey
      ifelse color != 27
      ;This set of commands dictates the movement of the healthy
Audreyes while the second one dictates how they move when injured
      [
       fd .9
       ; this algorithm makes the audrey disaper as they go through
the end door, simulating their escape.
       if xcor <= (origin_x) + .5
       [
         if xcor >= (origin_x) - .5
         [
           if ycor <= ((0) + 1.5)
           [
             if ycor >= ((0) + .5)
             [
               set escaped (escaped + 1)
               set audrey_escaped (audrey_escaped + 1)
               die
             ]
           ]
         ]
       ]
      ]
      ;if the Audreys are injured, they go 50% of their movement
speed.
      [
       fd .4
      ; this algorithm makes the andy disaper as they go through the
end door, simulating their escape.
```

```
       if xcor <= (origin_x) + .5
       [
         if xcor >= (origin_x) - .5
         [
           if ycor <= ((0) + 1.5)
           [
             if ycor >= ((0) + .5)
             [
               set escaped (escaped + 1)
               set audrey_escaped (audrey_escaped + 1)
               set injured (injured + 1)
               die
             ]
           ]
         ]
       ]
      ]
  ]


  ask andy
  [

      set heading towards patch (origin_x) ((0) + 1)
      bounce_andy
      fix_error_andy
      ifelse color != 47
      ;This set of commands dictates the movement of the healthy
andys, while the second one dictates how they move when injured
      [
       fd 1
       ; this algorithm makes the Andy disaper as they go through the
end door, simulating their escape.
       if xcor <= (origin_x) + .5
       [
         if xcor >= (origin_x) - .5
         [
           if ycor <= ((0) + 1.5)
           [
             if ycor >= ((0) + .5)
             [
               set escaped (escaped + 1)
               set reg_escaped (reg_escaped + 1)
               die
             ]
           ]
         ]
       ]
      ]
      ;if the andys are injured, they go 50% of their movement speed.
      [
       fd .5
      ; this algorithm makes the andy disaper as they go through the
end door, simulating their escape.
       if xcor <= (origin_x) + .5
       [
         if xcor >= (origin_x) - .5
         [
           if ycor <= ((0) + 1.5)
           [
             if ycor >= ((0) + .5)
             [
               set escaped (escaped + 1)
               set reg_escaped (reg_escaped + 1)
               set injured (injured + 1)
               die
             ]
           ]
         ]
       ]
      ]

]



  tick
end
```