```
_includes ["communication.nls" "bdi.nls"]

breed [ civilians civilian]
breed [ obstacles obstacle]
breed [ rescue-units rescue-unit]
breed [ rescued rescue-1]
breed [ bases base]
breed [ ambulances ambulance]

globals [rescued-cvls picked-up distance-traveled]
ambulances-own [alive beliefs intentions incoming-queue load]
bases-own [incoming-queue]

;;; Setting up the environment
to setup
   ;; (for this model to work with NetLogo's new plotting features,
   ;; __clear-all-and-reset-ticks should be replaced with clear-all at
   ;; the beginning of your setup procedure and reset-ticks at the end
   ;; of the procedure.)
   ;;__clear-all-and-reset-ticks
   clear-all
   reset-ticks
   random-seed seed
   setup-civilians
   setup-obstacles
   setup-ambulances
   setup-rescue-units
   setup-base
   set rescued-cvls 0
   set distance-traveled 0
   set picked-up 0
end

;;; creating disaster victims (civilians)
to setup-civilians
   create-civilians num-victims [
     rand-xy-co
     set shape "person"
     set color red
   ]
end

;;; creating obstacles
to setup-obstacles
   create-obstacles num-obstacles [
     rand-xy-co
     set shape "box"
     set color yellow
   ]
end

;;; creating ambulances
to setup-ambulances
```

```
   create-ambulances num-ambulances [
     rand-xy-co
     set alive true
     set shape "abulance"
     set color red
     set beliefs []
     set intentions []
     set incoming-queue []
     set load 0
   ]
end
;;; creating Rescue units
to setup-rescue-units
   create-rescue-units num-rescue-units [
     rand-xy-co
     set shape "rescue-unit"
     set color blue
   ]
end

to setup-base
   create-bases 1 [
     set shape "triangle 2"
     set color red
     setxy 0 0
     set incoming-queue []
   ]
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;; END of SETUP

;;;; Experiment
to run-rescue
   if count civilians = 0 and count rescued = 0 [stop]
   ask ambulances [ambulance-behaviour]
   ask rescue-units [rescue-unit-behaviour]
   tick
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;
;;; Ambulance Agent
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Hybrid Layer.
to ambulance-behaviour

   ;;; In a real world situation, problems are likely to occur. To simulate an
   ;;; ambulance breaking down (making them lose all communication with the other vehicles)
   ;;; I have added this condition (control the behaviour using the sliders)
   if possibility-of-ambulance-breakdown [
     if random ambulance-breakdown-chance = 0 [
       set alive false
       set color blue
     ]
   ]

   ;;; If the ambulance has not broken down (if it is alive), go and do your ambulance duties
   if alive [
     if reactive-ambulance-unit [stop]
     collect-msg-update-intentions-unit
     execute-intentions
   ]}

end

;;; Reactive layer of ambulance Agent.
to-report reactive-ambulance-unit
   if detect-ambulance [avoid-obstacle report true]
   if load >= maximum_load and at-dest base-id [set load 0] ;; unload patients
   if load >= maximum_load [move-towards-dest base-id report true]
   if detect-civilian [rescue-civilian pick-up-victim report true]
   report false
end

;;; Ambulance unit proactive behaviour
to collect-msg-update-intentions-unit
   let msg 0
   let performative 0

   let attendedTasks [] ;; a list of all tasks that are being tended to
   let firstAttendedTasks [] ;; a subset of attendedTasks consisting only of tasks that have only
been viewed once

   let bids [] ;; a list of all bids
   let bid-items [] ;; the coord on which bidding occured
   let highest-bids [] ;; the values of the highest (winning) bids
   let highest-bidders [] ;; A list of the highest bidders for particular auctions

   let bidItem false ;; The Item on which the current ambulance has bid on
   let bidVal false ;; The Value of the current ambulances bid

   ;; This variable will only get set if there is a new civilian that has been found that is
closer than the civilian that
   ;; is currently being looked for. If this flag is set (to the coords of the closer civilian)
then the agent will bid
   ;; will keeping on going to its currently selected civilian. Only if it wins the bid does it
actually change its intentions
   let closerItem false

   ;; loop through the incoming queue
   while [not empty? incoming-queue]
   [
     set msg get-message ;; pop message from the queue
     set performative get-performative msg ;; get the performative from the message
     ;; A request is a message that is broadcasted by a rescue agent when that agent finds a
     ;; civilian. If a civilian is found and that civilian is the closest civilian to the agent
     ;; he should make a bid.
     if performative = "request" [
       add-belief get-content msg
       show item 1 get-content msg
       if item 1 get-content msg = item 1 closer beliefs [
         set closerItem item 1 get-content msg
         ;; set intentions [] ;this should be where the bid is made since the bid will be made,
change to just be a flag
       ]
     ]

     ;; If a victim has been saved by a different ambulance, then this ambulance should remove
the beliefs
     ;; and possibly intentions if the intentions are for the same coord
     if performative = "saved" [
       remove-belief get-content msg
       let coords item 1 get-content msg
       if not empty? intentions [
         if get-intention = (list (word "move-towards-dest " coords) (word "at-dest " coords)) [
           set intentions []
         ]
       ]
     ]

     ;; if a bid has arrived, add it to the list of arrived bids, if the bid is from yourself,
     ;; save it into a variable (later we must compare our own bid to all of the other bids)
     if performative = "bid" [
       set bids lput msg bids

       if get-sender msg = (word who) [
         set bidItem first get-content msg
         set bidVal item 1 get-content msg
       ]
     ]

     ;; This performative lets the ambulance know that someone else is already saving a specific
person.
     ;; Also saves the message and later will repost it with the second-saving performative. This
is because
     ;; the order in which ambulances do their task in a tick may change so you need atleast 2
ticks to be sure
     ;; that you have received all messages.
     if performative = "saving" [
       set attendedTasks lput get-content msg attendedTasks
       set firstAttendedTasks lput get-content msg firstAttendedTasks
     ]

     ;; This performative lets the ambulance know that someone else is already saving a specific
person.
     ;; It does not add a new message.
     if performative = "second-saving" [
       set attendedTasks lput get-content msg attendedTasks
     ]
   ]

   ;; for each 'saving' message received send a message to yourself saying that someone is
saveing a certain co-ord.
   ;; This is so that the 'saving' message stays around for the next tick, even if the order of
which ambulances
   ;; execute tasks when can change
   foreach firstAttendedTasks [
     send add-receiver who add-content ? create-message "second-saving"
   ]

   ;; if a bidItem was found, it means that you have cast a bid and everyone else must have
bidded
   ;; already too. In this case you should find the winning bids and see if you have won.
   ;; Checking if the bidItem is not already taken by another ambulance prevents unnecessary
computation and
   ;; stops ambulances which are the same distance away from a civilian going together to get the
same civilian
   ifelse (bidItem != false) and not (member? bidItem attendedTasks) [
     foreach bids [
       let bid ?

       let content get-content bid ;; the full content of the bid [coords, distance]
       let bid-item first content ;; the coords that are being bid on

       let bid-item-pos position bid-item bid-items ;; the position inside bid-items of the
current auction for coords.
       let sender get-sender bid ;; the sender of the bid

       ;; if bid-item-pos = false, no auction for the specific coords has been started.
       ifelse bid-item-pos = false [
         ;; start the auction by adding the bid-item to the end of the list of bid-items. This
will make the position
         ;; of the bid-item be the last index of bid-items
         set bid-items lput bid-item bid-items
         set bid-item-pos (length bid-items) - 1

         ;; since this is the first bid in the auction, we set it to be the highest bid. Assign
the value of the
         ;; highest bidder for the auction to be the sender of the bid
         set highest-bids lput item 1 content highest-bids
         set highest-bidders lput sender highest-bidders

       ] [
         ;; we want the closest position to a set of coords so the highest bid is actually the
bid with the lowest value
         ;; (the smallest distance). Check if the current bid-item's distance is less then the
current highest bid.
         ;; If so, update the highest bid to be the current bid value and the highest bidder to
the sender of the bid
         if item bid-item-pos highest-bids > item 1 content [
           set highest-bids replace-item bid-item-pos highest-bids (item 1 content)
           set highest-bidders replace-item bid-item-pos highest-bidders sender
          ]
        ]
      ]

      ;; get the position of the auction that we actually care about (the auction that we bidded
on)
      let bidPos position bidItem bid-items

      ;; if the value that we bid for this auction is the same as the highest bid, it means that
we have won the bid.
      ;; The ambulance should set our intentions to be for the current coordinates and let the
other ambulances know that
      ;; it is going after the specific person. If two ambulances had the same bid, the one that
sends the 'saving' message first
      ;; will pick up the victim.
      if item bidPos highest-bids = bidVal [
        set intentions []
        add-intention "pick-up-victim" "true"
        if not at-dest bidItem [
          add-intention (word "move-towards-dest " bidItem) (word "at-dest " bidItem)
          broadcast-to ambulances add-content get-intention create-message "saving"
        ]
      ]
    ]
    [
      ;; Even though you haven't bidded yet, others may have already cast their bid. we have to
make sure
      ;; that when we compare bids (when the agent's own bid comes back) that we also include bids
that
      ;; happened before the agent cast its bid
      foreach bids [
        set incoming-queue lput ? incoming-queue
      ]
    ]

    ; all items that were bidded on which weren't won by the current agent can be assumed to be
attendedTasks
    ; since bidding is a one round process with a definative winner for each auction
    foreach bid-items  [
      set attendedTasks lput (list (word "move-towards-dest " ?) (word "at-dest " ?))
attendedTasks
    ]

    ifelse closerItem != false [
      ;; if closerItem is set, it means that although the ambulance has intentions, there is a
civilian which is closer
      ;; to the ambulance than the civilian we are currently chasing. If this is the case then we
should cast a bid on that item.
      broadcast-to ambulances add-content (list closerItem distance-coords closerItem) create-
message "bid"
    ] [
      ;; check if there are any civilians whos locations we know of. If we don't have intentions
we should bid on one of these
      ;; civilians so long as nobody else is already going to collect the civilian.
      if exist-beliefs-of-type "collect" and empty? intentions [

        ;; ------------------------------------------------------------------------------------
----
        ;; Create whitelist of all coords which are available for the ambulance to go to
        let whitelist []
        foreach beliefs [
          let b ?

          let match false
          foreach attendedTasks [
            if (list (word "move-towards-dest " item 1 b) (word "at-dest " item 1 b)) = ? [
              set match true
            ]
          ]

          if match = false [
            set whitelist lput b whitelist
          ]
        ]
        ;; ------------------------------------------------------------------------------------
----

        ;; if the whitelist is not empty it means that there are un-assigned tasks. We should bid
on one of these
        if length whitelist > 0 [

          ;; find the closest belief in the whitelist
          let bel closer filter [first ? = "collect"] whitelist
          let coords item 1 bel

          ;; cast a bid for the belief
          broadcast-to ambulances add-content (list coords distance-coords coords) create-message
"bid"
        ]
      ]
    ]

    ifelse not empty? intentions [
      ;; if we have intentions, broadcast these using performative saving. saving only cares about
      ;; intentions that are about saving a person so even if other intentions are sent it doesn't
      ;; matter (these other intentions will be ignored)
      broadcast-to ambulances add-content get-intention create-message "saving"
    ] [
      ;; If an ambulance has nothing to do, it is very time inefficiant to just sit and do
nothing.
      ;; so in order to not waste time, when there are no tasks for the ambulances to do, they
will
      ;; move-randomly with the rest of the rescue agents helping them search for victims. This
also
```

```
     ;; prevents the problem where sometimes and ambulance will have not intentions after
returning
     ;; to the base, so it will just sit and do nothing, blocking other ambulances from using the
base.
     move-randomly
   ]
end

;;; Reports the closest item in list.
;;; This reports the closer to the agent item from a list of items. The coordinates of the
;;; different members in the list of items must be in a list as well. For example
;;; the list must be of the form [ ["collect" [12 3] ["collect" [14 7]]]
to-report closer [itemlist]
   let closest first itemlist
   foreach itemlist
   [
      if distance-coords (item 1 ?) < distance-coords (item 1 closest)
        [set closest ?]
   ]
   report closest
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Rescue Units
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to rescue-unit-behaviour
   if detect-civilian [rescue-civilian inform-base stop]
   if detect-obstacle [avoid-obstacle stop]
   if true [move-randomly]
end

;;; Informing base for victim
;;; creates a message for the location of the victim, where the content is
;;; "victim-at" [xcor ycor]
to inform-base
   broadcast-to ambulances add-content (list "collect" (list (round xcor) (round ycor))) create-
message "request"
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;; Sensors
;; Detecting obstacles
;; Obstacles are obstacles and other rescue agents.
to-report detect-obstacle
   foreach (list patches in-cone 2 30)
   [
     if any? obstacles-on ? [show "obstacle-on" report true]
     if any? other rescue-units-on ? [show "rescue-on" report true]
   ]
   report false
end

to-report detect-ambulance
```

```
   foreach (list patches in-cone 2 30)
   [
     if any? other ambulances-on ? [report true]
   ]
   report false
end

;;; detecting a civilian
to-report detect-civilian
   ifelse any? civilians-here
   [report true]
   [report false]
end

;;;; Returns true if an agent is at the specific destination.
to-report at-dest [dest]
   if is-number? dest [
      ifelse ([who] of one-of turtles-here = dest)
      [report true]
      [report false]
    ]

    if is-list? dest [
      ifelse (abs (xcor - first dest) < 0.4 ) and (abs (ycor - item 1 dest) < 0.4)
      [report true]
      [report false]
    ]
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;; Actions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; rescueing a civilian
to rescue-civilian
   set rescued-cvls rescued-cvls + 1
   ask one-of civilians-here [
      set breed rescued
      set shape "person"
      set color green
   ]
end

;;; Actions that move the agent around.
;;; Turning randomly to avod an obstacle
to avoid-obstacle
   set heading heading + random 360
end

;; moving randomly. First move then turn
to move-randomly
   fd 1
   set heading heading + random 30 - random 30
end
```

```
;;;;;;;;;;;;;;;;;
to pick-up-victim
   ask rescued-here [die]
   set picked-up picked-up + 1
   set load load + 1
   broadcast-to ambulances add-content (list "collect" (list (round xcor) (round ycor))) create-
message "saved"
end

;;; Top level Reactive-traveling.
to move-towards-dest [dest]
   if true [travel-towards dest stop]
end

;;; Traveling towars a destination.
to travel-towards [dest]
    fd 0.2
    set distance-traveled distance-traveled + 0.2
    if is-number? dest
    [
      if not ((xcor = [xcor] of turtle dest) and (ycor = [ycor] of turtle dest))
      [
        set heading towards-nowrap turtle dest
      ]
    ];; safe towards

    if is-list? dest
    [
      if not ((xcor = first dest) and (ycor = item 1 dest))
      [
        set heading towardsxy-nowrap (first dest) (item 1 dest)
      ]
    ];; safe towards
end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Utilities
to rand-xy-co
   let x 0
   let y 0

   loop [
      set x random-pxcor
      set y random-pycor
      if not any? turtles-on patch x y and not (abs x < 4 and abs y < 4) [setxy x y stop]
   ]
end

;;; Reports the distance from a set of coordinates [x y] that are given in  a list eg [3 4]
to-report distance-coords [crds]
   report distancexy-nowrap (first crds) (item 1 crds)
end
```

```
;;; base ID is required to broadcasy a message to the base.
;;; This is intended for use with the add-receiver reporter.
to-report base-id
   report first [who] of bases
end
```