```
turtles-own[
  insured?
  checked?
  payoff
  m
  explored?
  ;numberofinsured
  ;numberofnotinsured
  max?
]
globals[
  numberofcliques
  component-size        ;; current running size of
component being explored
  giant-component-size  ;; size of largest connected
component
  components
  donewithinsured?
  donecounting?
  done?
  ]
to setup
  clear-all
  setup-turtles
  reset-ticks
  set-max-degree
  set donewithinsured? false

  setup-patches
end
to set-max-degree
  ask turtles[
   ifelse random-max-degree?
   [
     set m ((random 5) + 1)
     ]
   [
     set m max-degree
     ]
  ]
end
to setup-turtles
  set-default-shape turtles "circle"
  set numberofcliques 0
  set done? false
  set donecounting? false
  crt num-nodes
  layout-circle turtles max-pxcor - 20
  ask turtles [
    set payoff 0

    set max? false
    ;set numberofinsured 0
    ;set numberofnotinsured 0
    set insured? false
    set checked? false
    set color red

     if (random-float 100.0 <(prob-insured))[
      set color green
      set insured? true
      ]
  ]
  ;ask turtles [ set label who set label-color black]

end
to show-label
  ask turtles[

  ifelse show-payoff?
    [ set label payoff]
    [ set label ""]
  ]
end

to setup-patches
ask patches [
  set pcolor white
  ]
end

to go
  show-label
  if done? and not donecounting?
  [
    find-all-components
  ]
  if not donewithinsured? [
    add-edge
  ]
  ;if not donewithnotinsured?[
  ;  add-edge-not-insured
  ;  ]
  tick
end

to add-edge
 let node1 one-of turtles with[not checked? and
not max?]
 if node1 = nobody
 [
   set done? true
   display
   user-message "insured clique finished"
   stop
   ]
  ask node1[
    if (m - (count(link-neighbors))) <= 0
    [
     set max? true
     add-edge
    ]
    let node2 one-of turtles with [not link-neighbor?
node1 and (self != node1) and not checked? and
not max?]

    ifelse node2 = nobody
   [
     set checked? true
     add-edge
     ]
   [
     let nolinkpayoff payoff
     let n1m m
     let n1numberofinsured (count(link-neighbors
with[insured?]))
     let n1numberofnotinsured (count(link-neighbors
with[not insured?]))
     ifelse insured?
     [
      ;node1 is insured
      ask node2
      [
        if (m - (count(link-neighbors))) <= 0
        [
         set max? true
         add-edge
        ]
        let nolinkpayoff2 payoff
        let n2m m
        let n2numberofinsured (count(link-neighbors
with[insured?]))
        let n2numberofnotinsured (count(link-
neighbors with[not insured?]))
        ifelse insured?
        [
         ;node2 and node1 insured
         let g1 ((gamma / 100 ) / (n1m -
n1numberofinsured - n1numberofnotinsured ) )
         let g2 ((gamma / 100 ) / (n2m -
n2numberofinsured  - n2numberofnotinsured ))
         let newpayoff1 (nolinkpayoff + (beta / 100 ) -
(insurancelink / 100 ) + g1)
         let newpayoff2 (nolinkpayoff2 + (beta / 100 )
- (insurancelink / 100 ) + g2)
         if newpayoff1 > nolinkpayoff and newpayoff2
> nolinkpayoff2
          [
           ;add link
             create-link-with node1
             set payoff (newpayoff2 - g2)
             ;set numberofinsured (numberofinsured +
1)
             if (m - (count(link-neighbors))) <= 0
             [;set max true
              set payoff (payoff + (gamma / 100 ))
              set max? true
              ]
             ask node1[
              set payoff (newpayoff1 - g1)
              ;set numberofinsured (numberofinsured +
1)
              if (m - (count(link-neighbors))) <= 0
             [;set max true
              set payoff (payoff + (gamma / 100 ))
              set max? true
              ]

             ]
            ]
            ;done with adding link
            ]
          [;begin else
            ;node2 not insured
            let g1 ((gamma / 100 ) / (n1m -
n1numberofinsured - n1numberofnotinsured ) )
            let g2 ((gamma / 100 ) / (n2m -
n2numberofinsured  - n2numberofnotinsured ))
            let newpayoff1 (nolinkpayoff + (beta / 100 ) -
(risk / 100 ) - (insurancelink / 100 ) + g1)
            let newpayoff2 (nolinkpayoff2 + (beta / 100 )
+ g2)
            if newpayoff1 > nolinkpayoff and newpayoff2
> nolinkpayoff2
            [
             ;add link
             create-link-with node1
             set payoff (newpayoff2 - g2)
             ;set numberofinsured (numberofinsured +
1)
             if (m - (count(link-neighbors))) <= 0
             [;set max true
              set payoff (payoff + (gamma / 100 ))
              set max? true
              ]
             ask node1[
              set payoff (newpayoff1 - g1)
              ;set numberofnotinsured
(numberofnotinsured + 1)
              if (m - (count(link-neighbors))) <= 0
             [;set max true
              set payoff (payoff + (gamma / 100 ))
            set max? true
            ]
           ]
          ]
          ;done with adding link

          ];end else
        ];done with node2
        ]
      [
        ;node1 not insured
        ask node2
        [
          if (m - (count(link-neighbors))) <= 0
          [
            set max? true
            add-edge
          ]
          let nolinkpayoff2 payoff
          let n2m m
          let n2numberofinsured (count(link-neighbors
with[insured?]))
          let n2numberofnotinsured (count(link-
neighbors with[not insured?]))
          ifelse insured?
          [
            ;node2 insured and node1 not insured
            let g1 ((gamma / 100 ) / (n1m -
n1numberofinsured - n1numberofnotinsured ) )
            let g2 ((gamma / 100 ) / (n2m -
n2numberofinsured  - n2numberofnotinsured ))
            let newpayoff1 (nolinkpayoff + (beta / 100 ) +
g1)
            let newpayoff2 (nolinkpayoff2 + (beta / 100 )
- (insurancelink / 100 ) - (risk / 100 ) + g2)
            if newpayoff1 > nolinkpayoff and newpayoff2
> nolinkpayoff2
            [
              ;add link
              create-link-with node1
              set payoff (newpayoff2 - g2)
              ;set numberofnotinsured
(numberofnotinsured + 1)
              if (m - (count(link-neighbors))) <= 0
              [;set max true
                set payoff (payoff + (gamma / 100 ))
                set max? true
                ]
              ask node1[
                set payoff (newpayoff1 - g1 )
                ;set numberofinsured (numberofinsured +
1)
                if (m - (count(link-neighbors))) <= 0
                [;set max true
                set payoff (payoff + (gamma / 100 ))
                set max? true
                ]

              ]
            ]
            ;done with adding link
            ]
          [;begin else
            ;node2 and node1 not insured
            let g1 ((gamma / 100 ) / (n1m -
n1numberofinsured - n1numberofnotinsured ) )
            let g2 ((gamma / 100 ) / (n2m -
n2numberofinsured  - n2numberofnotinsured ))
            let newpayoff1 (nolinkpayoff + (beta / 100 ) -
(risk / 100 ) + g1)
            let newpayoff2 (nolinkpayoff2 + (beta / 100 )
- (risk / 100 ) + g2)
            if newpayoff1 > nolinkpayoff and newpayoff2
> nolinkpayoff2
            [
              ;add link
              create-link-with node1
              set payoff (newpayoff2 - g2 )
              ;set numberofnotinsured
(numberofnotinsured + 1)
              if (m - (count(link-neighbors))) <= 0
              [;set max true
                set payoff (payoff + (gamma / 100 ))
                  set max? true
                ]
              ask node1[
                set payoff (newpayoff1 - g1)
                if (m - (count(link-neighbors))) <= 0
                [;set max true
                  set payoff (payoff + (gamma / 100 ))
                  set max? true
                ]

                ;set numberofnotinsured
(numberofnotinsured + 1)
              ]
            ]
            ;done with adding link
            ];end else
        ];done with node2
        ]
      ;set color green
      ;add-edge
      ]
    ]
    layout
```

```
end

to add-edge-not-insured
 let node1 one-of turtles with[not insured? and not
checked?]
 if node1 = nobody
 [
   ;display
   ;user-message "non-insured clique finished"
   stop
   ]
  ask node1[
    let node2 one-of turtles with [not insured? and
not link-neighbor? node1 and (self != node1) and
not checked?]

    ifelse node2 = nobody
   [
     display
     set checked? true
     add-edge-not-insured
     ]
  [
   create-link-with node2
   add-edge-not-insured
    ]
   ]
   layout
end
to find-all-components
  set components []
  set giant-component-size 0

  ask turtles [ set explored? false ]
  ;; keep exploring till all turtles get explored
  loop
  [
    ;; pick a turtle that has not yet been explored
    let start one-of turtles with [ not explored? ]
    if start = nobody [
      set donecounting? true
      display
      user-message "Done counting cliques"
      stop ]
    ;; reset the number of turtles found to 0
    ;; this variable is updated each time we explore
an
    ;; unexplored turtle.
    set component-size 0
    ask start [ explore ]
    set numberofcliques numberofcliques + 1
    ;; the explore procedure updates the
component-size variable.
    ;; so check, have we found a new giant
component?
    if component-size > giant-component-size
    [
      set giant-component-size component-size
    ]
    set components lput component-size
components
  ]
end

;; finds all turtles reachable from this turtle
to explore ;; turtle procedure
  if explored? [ stop ]
  set explored? true
  set component-size component-size + 1
  ask link-neighbors [ explore ]
end

to layout
  repeat 10 [
    layout-spring (turtles with [any? link-neighbors])
links 0.4 6 1
    display  ;; so we get smooth animation
  ]
end
```