Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра інформатики

Звіт з лабораторної роботи №4

По предмету: «Штучний інтелект для інформаційних технологій»

Виконав:

студент групи ІТІНФ-20-1

Самченко С. О.

Харків 2023

**Мета:**

Вивчити та опанувати алгоритми створення лабіринтів.

**Завдання:**

Реалізувати The Recursive Backtracker Algorithm.

**Хід роботи:**

;; Maze Escape a simulation to escape from a maze

;; Version 1.0

;; Written by Riccardo Rotondo (riccardo.rotondo@phd.unict.it)

;; NetLogo version 6.1.1

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

;; load network extensions

extensions [Nw]

;; defining global variables

globals

[

  tiles

  new-links-red

]

;; defining breeds

breed [nodes node]

breed [builders builder]

breed [maze-runners mr]

;; defining breeds variable

nodes-own [node-id maze-entrance maze-exit exit? corner?]

builders-own [stack]

maze-runners-own [prev-node current-node next-node next-path visited-nodes

                  visited-hubs I-found-exit?]

;; all functions defined here

;; setup button

to setup

  clear-all

  build-tiles

  init-nodes

  build-maze

  set-entrance-exit

  setup-maze-runners

  reset-ticks

end

;; use in order to run simulation on the same maze several times

to reset-maze-runners

  reset-ticks

  ask links [set color black set thickness 0]

  ask maze-runners [die]

  setup-maze-runners

  clear-all-plots

end

;; build orderd white tiles in the world

;; according to the spacing (their distance)

to build-tiles

  ask patches [set pcolor 4 ]

  set tiles patches with

  [ pxcor mod spacing = 0

    and pycor mod spacing = 0

    and abs pxcor +  spacing < max-pxcor

    and abs pycor +  spacing < max-pycor

    and abs pxcor -  spacing > min-pxcor

    and abs pycor - spacing > min-pycor

  ]

  ask tiles [ set pcolor white ]

  set new-links-red 0

end

;; Init nodes of given color,size and shape on each nodes

;; All boolear variables are set false

to init-nodes

  let index 1

  ask tiles

  [

    sprout-nodes 1

    [

       set color black

       set size 1

       set shape "circle"

       set node-id index

       set exit? false

       set maze-entrance false

       set maze-exit false

       set corner? false

     ]

     set index index + 1

  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

;; Build maze

to build-maze

  create-builders  1

  [ ;; choose a random starting point

    let start one-of tiles

    set xcor [pxcor] of start

    set ycor [pycor] of start

    ;; set heading and color

    set heading 0

    set color 9.91

    ask patches in-radius 1 [ set pcolor [color] of myself ]

    set stack []

  ]

  ask builders

  [ ;; store starting point

    set stack fput ( list xcor ycor ) stack

    while [ length stack > 0 ]

    [ ;; in this while the maze building process

      let target 0

      let left-right 0

      let straight 0

      let running 0

      let paths find-open-paths

      ifelse any? paths

      [ ;; ifelse any? paths --> paths is not-empty

        set straight patch-ahead spacing

        set left-right paths with [ self != straight ]

        let node1 0

        if (any? nodes-on patch-here)

        [ ask one-of nodes-on patch-here [set node1 self] ]

        ifelse (any? left-right ) or not is-open straight

        [

          set target one-of left-right

          ;; record stack

          set stack fput ( list xcor ycor ) stack

          set heading towards target

          draw-move

        ]

        [

          set running true

          while [ running ]

          [

            set heading towards straight

            draw-move

            set straight patch-at ( dx \* spacing) ( dy \* spacing )

            set running ( random-float 1.0 >= 1 and is-open straight )

          ]

        ]

        if (any? nodes-on patch-here)

        [ask one-of nodes-on patch-here

          [create-link-with node1 [set color black]]]

       ]

      [ ;; ifelse any? paths --> path is empty

        ifelse length stack > 0

        [ ;; start the building process

          setxy (item 0 (item 0 stack)) (item 1 (item 0 stack))

          ;; removing first element from stack

          set stack but-first stack

         ]

         [ stop ]

       ]

    ] ;;close while

    die

  ];; close ask builders

end

;; draw move

to draw-move

  let start-spot patch-here

  ask start-spot [ ask patches in-radius 1 [ set pcolor 9.91 ] ]

  repeat spacing [ ask patches in-radius 1 [ set pcolor 9.91 ] jump 1 ]

 end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

;; find maze entrance and exit

to set-entrance-exit

  let set-nodes-exit false

  let minx min [xcor] of nodes

  let miny min [ycor] of nodes

  let maxx max [xcor] of nodes

  let maxy max [ycor] of nodes

  let edge-nodes nodes with [

    pxcor = minx or pxcor = maxx or pycor = miny or pycor = maxy ]

  ask edge-nodes

  [

    ;set color black

    if (pxcor = minx and pycor = miny) [set corner? true]

    if (pxcor = minx and pycor = maxy) [set corner? true]

    if (pxcor = maxx and pycor = miny) [set corner? true]

    if (pxcor = maxx and pycor = maxy) [set corner? true]

  ]

  ask nodes

  [

    let exit-found? false

    ask patch-here

    [

       if (count neighbors with [pcolor = 4 ] = 5 ) [set exit-found? true]

       if (count neighbors with [pcolor = 4 ] = 2

        and count neighbors with [pcolor = 9.91] = 6 ) [

        set exit-found? true]

    ]

    if exit-found? = true  [set color black set size 2 set exit? true]

  ]

  let minx-exit min [xcor] of nodes with [exit? = true]

  let miny-exit min [ycor] of nodes with [exit? = true]

  let maxx-exit max [xcor] of nodes with [exit? = true]

  let maxy-exit max [ycor] of nodes with [exit? = true]

  ;; let's define two possible exit, one in the edge, the other even in the

  ;; middle

  let edge-inout-nodes edge-nodes with [exit? = true]

  let inout-nodes nodes with [exit? = true]

  let possible-entrance one-of edge-inout-nodes

  if possible-entrance = nobody

  [ while [possible-entrance = nobody]

    [ set possible-entrance one-of inout-nodes]

  ]

  ask possible-entrance

  [ set maze-entrance true

    set label-color black

    set label "entrance"

    set color green

    set size 3

    (ifelse

    pxcor = minx-exit

    [

      if debug >= 1 [print "pxcor = minx-exit"]

      let possible-exit one-of edge-inout-nodes with [pxcor = maxx-exit]

      ifelse possible-exit != nobody

        [

          ask possible-exit

          [

            set maze-exit true set color green set size 3

            set label-color black set label "exit"

          ]

        ]

        [

          set possible-exit one-of inout-nodes with [label != "entrance"]

          ifelse possible-exit != nobody

          [

            ask possible-exit

            [

              set maze-exit true set color green set size 3

              set label-color black set label "exit"

            ]

          ]

          [

            print "Unable to find and entrance"

            print "Check spacing or other parameters"

          ]

        ]

      ]

    pxcor = maxx-exit

    [

        if debug >= 1 [print "pxcor = maxx-exit"]

        let possible-exit one-of edge-inout-nodes with [pxcor = minx-exit]

      ifelse possible-exit != nobody

        [ ask possible-exit

          [ set maze-exit true set color green set size 3

              set label-color black set label "exit"

          ]

        ]

        [ set possible-exit one-of inout-nodes with [label != "entrance"]

          ifelse possible-exit != nobody

          [

            ask possible-exit

            [

              set maze-exit true set color green set size 3

              set label-color black set label "exit"

            ]

          ]

          [

            print "Unable to find and entrance"

            print "Check spacing or other parameters"

          ]

        ]

      ]

    pycor = miny-exit

    [

        if debug >= 1 [print "pycor = miny-exit"]

        let possible-exit one-of edge-inout-nodes with [pycor = maxy-exit]

      ifelse possible-exit != nobody

        [

          ask possible-exit

          [

            set maze-exit true set color green set size 3

            set label-color black set label "exit"

          ]

        ]

        [ set possible-exit one-of inout-nodes with [label != "entrance"]

          ifelse possible-exit != nobody

          [

            ask possible-exit

            [

              set maze-exit true set color green set size 3

              set label-color black set label "exit"

            ]

          ]

          [

            print "Unable to find and entrance"

            print "Check spacing or other parameters"

          ]

        ]

      ]

    pycor = maxy-exit

    [

      if debug >= 1 [print "pycor = maxy-exit"]

      let possible-exit one-of edge-inout-nodes with [pycor = miny-exit]

      ifelse possible-exit != nobody

        [

          ask possible-exit

          [

            set maze-exit true set color green set size 3

            set label-color black set label "exit"

          ]

        ]

        [

          set possible-exit one-of inout-nodes with [label != "entrance"]

          ifelse possible-exit != nobody

          [

            ask possible-exit

            [

              set maze-exit true set color green set size 3

              set label-color black set label "exit"

            ]

          ]

          [ print "Unable to find and entrance"

            print "Check spacing or other parameters"

          ]

        ]

      ]

    )

  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

;; setup maze runners

to setup-maze-runners

  ask one-of nodes with [label = "entrance"]

  [ let present-node self

    ask patch-here

    [ sprout-maze-runners 1

      [ set size 5

        set color sky

        set current-node present-node

        set visited-nodes []

        set visited-hubs []

        set I-found-exit? false

      ]

    ]

  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

to find-exit

  ask maze-runners

  [

    set visited-nodes lput current-node visited-nodes

    ifelse [label] of current-node = "entrance"

    [ ;; current-node is entrance

      if debug >= 1 [print "current node is entrance"]

      set next-path one-of [my-links] of current-node

      ifelse [color] of next-path = black

      [ ;;next path black

        if debug >= 1 [print "next path is black"]

        ifelse current-node = [end1] of next-path

        [set next-node [end2] of next-path] [set next-node [end1] of next-path]

        color-link-green

        forward-maze-runner

      ]

      [ ;;next path NOT black

        print "next black not black at entrance not defined"

      ]

    ]

    [ ;; current-node is NOT entrance

;      if debug >= 1 [print "current node is NOT entrance"]

      ifelse [exit?] of current-node = true

      [ ;; current node is a blind spot, could be an exit

        if debug >= 1 [print "current node is a blind spot"]

        ifelse [maze-exit] of current-node = true

        [ ;; exit found

          set I-found-exit? true

          if [color] of next-path != green

          [color-best-path]

          if debug >= 1 [print "exit found"]

        ]

        [ ;;exit NOT found

          color-link-red

          go-back

        ]

      ]

      [ ;; current node is NOT a blind spot

;        if debug >= 1 [print "current node NOT is entrance"]

        ifelse [color] of link [who] of prev-node

                               [who] of current-node = green

        [ ;; previous path is green

          if debug >= 1 [print "previous path is green"]

          ifelse count [my-links] of current-node > 2

          [ ;; node is a hub

            if debug >= 1 [print "node is a hub"]

            found-new-hub

            set next-path search-link green

            ifelse next-path != nobody

            [ ;; next path is green

              if debug >= 1 [print "next path is green"]

              ifelse current-node = [end1] of next-path

              [set next-node [end2] of next-path]

              [set next-node [end1] of next-path]

              forward-maze-runner

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

            ]

            [discover-unknown-hub]

          ]

          [ ;; node is NOT a hub

            set next-path search-link black

              ifelse next-path != nobody

              [ ;; next path is black

              if debug >= 1 [print "next-path is black"]

                ifelse current-node = [end1] of next-path

                 [ set next-node [end2] of next-path ]

                 [ set next-node [end1] of next-path ]

                color-link-green

                forward-maze-runner

             ]

            [print "next-path not black after green not defined"]

          ]

        ]

        [ ;; previous path is NOT green

          ifelse [color] of link [who] of prev-node [who] of current-node = yellow

          [ ;; previous path is yellow

            if debug >= 1 [print "prev-path is yellow"]

            ifelse count [my-links] of current-node > 2

            [ ;; node is a hub

              if debug >= 1 [print "node is hub"]

              found-new-hub

              discover-unknown-hub

            ]

            [ ;; node is NOT a hub

              set next-path search-link black

              ifelse next-path != nobody

              [ ;; next path is black

              ifelse current-node = [end1] of next-path

                [set next-node [end2] of next-path]

                [set next-node [end1] of next-path]

              color-link-yellow

              forward-maze-runner

              ]

              [ ;;next path is NOT black

                print "previous yellow next not black not defined"

              ]

            ]

          ]

          [print "previous path is not green and yellow not defines"]

        ]

    ]

   ]

  ]

;  tick

  ifelse new-links-red != 0

  [tick-advance new-links-red set new-links-red 0]

  [ if not mr-found-exit? [tick] ]

  if debug >= 1 [print ticks]

  if mr-found-exit? [stop]

end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

to forward-maze-runner

  if debug >= 1 [print "forward"]

  set heading report-mr-direction

  fd [link-length] of link [who] of current-node [who] of next-node

  set prev-node current-node

  set current-node next-node

end

to go-back

  if debug >= 1 [print "go-back"]

  set current-node last visited-nodes

  set visited-nodes remove current-node visited-nodes

  set visited-hubs remove current-node visited-hubs

  set prev-node last visited-nodes

  set xcor [xcor] of current-node

  set ycor [ycor] of current-node

  set next-path link [who] of prev-node [who] of current-node

  set heading report-mr-direction + 180

end

to color-link-green

  ask link [who] of current-node [who] of next-node

    [set color green set thickness 1]

end

to color-link-yellow

  ask link [who] of current-node [who] of next-node [set color yellow]

end

to color-link-red

  let last-node last visited-nodes

  let before-last-node item (length visited-nodes - 2) visited-nodes

  if last-node = last visited-hubs

  [ ;;this happens when mr is in a hub and all branch are red

    ;;in order to go back we need to remove the last visited-hubs

    set visited-hubs remove last visited-hubs visited-hubs

  ]

  if debug >= 2

  [

    print "color-link-red"

    print "last visited hub"

    print last visited-hubs

    print "last-node in visited-nodes"

    print last-node

    print "before-last-node in visited-nodes"

    print before-last-node

    print "link color red:"

  ]

  while [last-node != last visited-hubs]

  [

    ask link [who] of last-node [who] of before-last-node [set color red]

    if debug >= 2 [print link [who] of last-node [who] of before-last-node]

    set visited-nodes remove last-node visited-nodes

    set last-node last visited-nodes

    set before-last-node

      item (position last-node visited-nodes - 1) visited-nodes

    set new-links-red new-links-red + 1

  ]

end

to color-best-path

  let last-node last visited-nodes

  let before-last-node item (length visited-nodes - 2) visited-nodes

  if debug >= 2

  [

    print "color-best-path"

    print "last visited hub"

    print last visited-hubs

    print "last-node in visited-nodes"

    print last-node

    print "before-last-node in visited-nodes"

    print before-last-node

    print "link color green:"

  ]

  while [last-node != first visited-hubs]

  [

    ask link [who] of last-node [who] of before-last-node

    [set color green set thickness 1]

    if debug >= 2 [print link [who] of last-node [who] of before-last-node]

    set visited-nodes remove last-node visited-nodes

    set last-node last visited-nodes

    set before-last-node

      item (position last-node visited-nodes - 1) visited-nodes

  ]

end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

to discover-unknown-hub

  if debug >= 1 [print "discovery new hub"]

  set next-path search-link green

  ifelse next-path != nobody

  [ ;;next path is green

    if debug >= 1 [print "next-path is green"]

    forward-maze-runner

  ]

  [ ;; next path is NOT green

    set next-path search-link black

    ifelse next-path != nobody

    [ ;; next path is black

      if debug >= 1 [print "next path is black"]

      ifelse found-best-path?

      [ ;;one next-path is red and all others red

        if debug >= 1 [print "one next-path is black and all others red"]

        set visited-hubs remove last visited-hubs visited-hubs

        ifelse current-node = [end1] of next-path

        [set next-node [end2] of next-path][set next-node [end1] of next-path]

        color-link-green

        forward-maze-runner

      ]

      [ ;; more next-path black

        ifelse current-node = [end1] of next-path

        [set next-node [end2] of next-path][set next-node [end1] of next-path]

        color-link-yellow

        forward-maze-runner

      ]

    ]

    [ ;; there are not next path black

      set next-path search-link yellow

      ifelse next-path != nobody

      [ ;; next-path is yellow

        if debug >= 1 [print "next-path is yellow"]

        forward-maze-runner

      ]

      [ ;; there are not next path yellow

        set next-path search-link red

        ifelse next-path != nobody

        [ ;; next-path is red

          if debug >= 1 [print "there are only red path"]

          color-link-red

          go-back

        ]

        [ ;; next-path is NOT red]

          print "Error, this scenario should not happen"

        ]

      ]

    ]

  ]

end

to found-new-hub

  if debug >= 1 [print "found new hub"]

  set visited-hubs lput current-node visited-hubs

end

;; all to-report functions defined here

;; find open path

to-report find-open-paths

  let paths

  ( patches at-points

    (map [ [?1 ?2] ->

      (list (?1 \* spacing ) (?2 \* spacing) ) ] [ 0 0 1 -1 ] [1 -1 0 0 ])

   ) with [ pcolor = white ]

  report paths

end

;; check if path is open

to-report is-open

  [ a-patch ]

   report ([pcolor] of a-patch = white)

end

to-report report-mr-direction

  let lh 45

  ifelse current-node = [end1] of next-path

  [set lh [link-heading] of next-path][set lh

    [link-heading] of next-path + 180]

  report lh

end

to-report search-link [link-color]

  let new-link nobody

  let temp-prev-node prev-node

  ask current-node

  [ set new-link one-of

    (my-links with [color = link-color and other-end != temp-prev-node])

  ]

  report new-link

end

to-report found-best-path?

  if debug >= 1 [print "search for best bath"]

  let temp-prev-node prev-node

  let count-prev-path-green 0

  let count-next-path-black 0

  let count-next-path-red 0

  let total-path 0

  ask current-node

  [

    set count-prev-path-green count my-links with [color = green]

    set count-next-path-black count my-links with [other-end != temp-prev-node and color = black]

    set count-next-path-red count my-links with [other-end != temp-prev-node and color = red]

    set total-path count my-links

  ]

  if debug >= 2

  [

    print "count-prev-path-green"

    show count-prev-path-green

    print "count-next-path-black"

    show count-next-path-black

    print "count-next-path-red"

    show count-next-path-red

    print "total links"

    show total-path

  ]

  ifelse (count-next-path-black = 1) and

         (total-path = count-prev-path-green +

                           count-next-path-black + count-next-path-red )

  [ report true ][report false]

end

to-report mr-found-exit?

  let a-mr-found-exit? false

  let mr-on-exit one-of maze-runners with [I-found-exit? = true]

  if mr-on-exit != nobody [set a-mr-found-exit? true]

  report a-mr-found-exit?

end

;;;;;;;;;;;;;;;;;;;;;;;;line to keep code in 80 columns;;;;;;;;;;;;;;;;;;;;;;;;

**Результат роботи:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание  
Рисунок 4.1 — Результат виконання алгоритму.

**Висновок:**

При реалізації алгоритму застосував такий підхід: процес побудови лабіринту використовує стек для зберігання поточної позиції будівельника та використовує рекурсивні виклики для руху по лабіринту, а під час будівництва лабіринту використовуються деякі правила, такі як уникнення стикання зі стінами та створення "глухих кутів" для забезпечення проникнення вийти з лабіринту.