Appendix A - PSP Time Recording Log

Student Name: Poorav Sharma (10636908)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start  Date  and  Time | Stop  Date  and  Time | Time  Taken | LOC | Paste actual code here |
| 08/09/2023  7:00am | 08/09/2023  13:30 | 6.5 hours | 198 | -----------------------------------------------------------------------------------------  --  -- main.lua  -- Week 7 deliverable  -----------------------------------------------------------------------------------------  -- 0 and O are the cell  -- 1 and # is the space  -- i tried using the cell rules but coudn't make detect the neighbours properly  local colums = 5  local rows = 5    function createMatrixArray(rows, colums)      local matrixArray = {}      for i = 1, rows do          matrixArray[i] = {}          for j = 1, colums do              matrixArray[i][j] = ""          end      end      return matrixArray    end  function fillMatrix(matrix)      for i = 1, rows do          for j = 1, colums do              matrix[i][j] =  1            end      end      --for the week 7 deliverable to get 2(a) figure state for the cell      matrix [2][3] = 0      matrix [3][3] = 0      matrix [4][3] = 0  end  function displayMatrix(matrix)      for i = 1, rows do          for j = 1, colums do              if matrix[i][j] ==  0 then                  io.write("O")              else                  io.write("#")              end          end          print("")      end  end  function changePattern(matrix)      if matrix[2][3] ==0 and matrix[4][3]==0 and matrix[4][3]==0 then          for i = 1, rows do              for j = 1, colums do                  matrix[i][j] =  1                end          end          --for the week 7 deliverable to get 2(a) figure state for the cell          matrix [3][2] = 0          matrix [3][3] = 0          matrix [3][4] = 0      end  end  function detectNeighbourCells(currentCell, x, y)      local neighbourCells = 0      for i = -1, 1 do          for j = -1, 1 do              if (i == 0 and j == 0) then              else              local row = ((x-1 + i + rows) % rows)+1              local colum = ((y-1 + j + colums) % colums)+1                 if currentCell[row][colum] == 0 then                  neighbourCells = neighbourCells + 1               end              end          end        end      return neighbourCells  end  ---[[  function nextState(currentCell)      local nextStateMatrix = currentCell      for i = 1, rows do          for j = 1, colums do              x = i              y= j              neighbourDetected = detectNeighbourCells(currentCell, x, y)              cellLocated = y              if (cellLocated == 1) then                  if neighbourDetected == 3 then                      nextStateMatrix[i][j] = 0                  else                      nextStateMatrix[i][j] = 1                  end              else                  if (neighbourDetected <2 or neighbourDetected>3) then                      nextStateMatrix[i][j] = 1                  else                   nextStateMatrix[i][j] = 0                  end              end          end        end      displayMatrix(nextStateMatrix)      return nextStateMatrix  end  --]]  --[[  function nextState(currentCell)      local nextStateMatrix = currentCell      for i = 1, rows do          for j = 1, colums do              x = i              y= j              neighbourDetected = detectNeighbourCells(currentCell, x, y)              cellLocated = y              if (cellLocated == 0) and (neighbourDetected <2 or neighbourDetected>3) then                      nextStateMatrix[i][j] = 1              elseif (cellLocated == 1 and neighbourDetected == 3) then                  nextStateMatrix[i][j] = 0              else                  nextStateMatrix[i][j] = 1                end          end        end          displayMatrix(nextStateMatrix)      return nextStateMatrix  end  --]]  --[[  function duplicateArrayMatrix(oldMatrix)      local copiedRows = rows      local copiedColoums = colums      local copiedMatrix ={}      for i = 1, rows do          copiedMatrix[i] ={}          for j = 1, colums do              copiedMatrix[i][j]= oldMatrix[i][j]          end        end      return copiedMatrix  end  --]]  function simulate(matrix)        local currentCell = matrix      iteration = 1;      while iteration <=4 do          print("")          currentCell = nextState(currentCell)          iteration = iteration +1        end  end  function iterate(matrix)      for i =1, 2 do          print("")          fillMatrix(matrix)          displayMatrix(matrix)          changePattern(matrix)          print("")          displayMatrix(matrix)      end  end  function main()     local matrix = createMatrixArray(rows, colums)      --fillMatrix(matrix)     -- displayMatrix(matrix)      --changePattern(matrix)      --print("")    --  displayMatrix(matrix)      --simulate(matrix)      iterate(matrix)  end  main() |
| 13/09/2023  10pm | 13/09/2023  12pm | 2 hours | 50 | -----------------------------------------------------------------------------------------  --  -- main.lua  -- Week 8 deliverable  -----------------------------------------------------------------------------------------  --Poorav Sharma  -- 0 and O are the cell  -- 1 and # is the space  -- i tried using the cell rules but coudn't make detect the neighbours properly -fixed this problem for week 8 deliverables i ended up with two ways to fix it one is to duplicate the old matrix and the other is to create a new metrix and fill it with values  -- i just outputed the iteration 4 for figure 2(b to d)  local colums = 5  local rows = 5  function createMatrixArray(rows, colums)  local matrixArray = {}  for i = 1, rows do  matrixArray[i] = {}  for j = 1, colums do  matrixArray[i][j] = ""  end  end  return matrixArray    end  function fillMatrix(matrix)  for i = 1, rows do  for j = 1, colums do  matrix[i][j] = 1    end  end  --for the week 7 deliverable to get 2(a) figure state for the cell  matrix [2][3] = 0  matrix [3][3] = 0  matrix [4][3] = 0  end  function displayMatrix(matrix)  print("")  for i = 1, rows do  for j = 1, colums do  if matrix[i][j] == 0 then  io.write("O")  else  io.write("#")  end  end  print("")  end  end  --[[  function changePattern(matrix)  if matrix[2][3] ==0 and matrix[4][3]==0 and matrix[4][3]==0 then  for i = 1, rows do  for j = 1, colums do  matrix[i][j] = 1    end  end  --for the week 7 deliverable to get 2(a) figure state for the cell  matrix [3][2] = 0  matrix [3][3] = 0  matrix [3][4] = 0  end  end  --]]  function detectNeighbourCells(currentCell, x, y)  local neighbourCells = 0  for i = -1, 1 do  for j = -1, 1 do  if (i == 0 and j == 0) then  else  local row = ((x-1 + i + rows ) % rows)+1  local colum = ((y-1 + j +colums) % colums)+1    if currentCell[row][colum] == 0 then  neighbourCells = neighbourCells + 1  end    end  end    end  return neighbourCells  end  --[[ creates a empty matrix where it fills the matrix for the next state the according to the rules  function nextState(currentMatrix)  local nextStateMatrix = createMatrixArray(rows, colums)  for i = 1, rows do  for j = 1, colums do  x = i  y= j  neighbourDetected = detectNeighbourCells(currentMatrix, x, y)  cellLocated = currentMatrix[i][j]  if (cellLocated == 1) then  if neighbourDetected == 3 then  nextStateMatrix[i][j] = 0  else  nextStateMatrix[i][j] = 1  end  else  if (neighbourDetected <2 or neighbourDetected>3) then  nextStateMatrix[i][j] = 1  else  nextStateMatrix[i][j] = 0  end  end  end    end    displayMatrix(nextStateMatrix)  return nextStateMatrix  end  --]]  ---[[ duplicates the current matrix and changes the cells according to the rules  function nextState(currentMatrix)  local nextStateMatrix = duplicateArrayMatrix(currentMatrix)  for i = 1, rows do  for j = 1, colums do  neighbourDetected = detectNeighbourCells(currentMatrix, i, j)  cellLocated = currentMatrix[i][j]  if (cellLocated == 0) then  if(neighbourDetected <2 or neighbourDetected>3) then  nextStateMatrix[i][j] = 1  else  --cell stays the same  end  else  if(cellLocated == 1 and neighbourDetected == 3) then  nextStateMatrix[i][j] = 0  else  --cell stays the same  end  end  end    end      displayMatrix(nextStateMatrix)  return nextStateMatrix  end  function duplicateArrayMatrix(oldMatrix)  local copiedMatrix ={}  for i = 1, rows do  copiedMatrix[i] ={}  for j = 1, colums do  copiedMatrix[i][j]= oldMatrix[i][j]  end    end  return copiedMatrix  end  --]]  function simulate(matrix)    local currentMatrix = matrix  iterated =1;  iteration = 4;  while iterated <=iteration do  currentMatrix = nextState(currentMatrix)  iterated = iterated +1    end  end  function main()  local matrix = createMatrixArray(rows, colums)  ---[[  fillMatrix(matrix)  displayMatrix(matrix)    simulate(matrix)    end  main() |
| 14/09/2023  12am | 14/09/2023  01:45 | 1.75 hours | 70 | -----------------------------------------------------------------------------------------  --  -- main.lua  -- Week 8 deliverable  -----------------------------------------------------------------------------------------  --Poorav Sharma  -- 0 and O are the cell  -- 1 and # is the space  -- i tried using the cell rules but coudn't make detect the neighbours properly -fixed this problem for week 8 deliverables i ended up with two ways to fix it one is to duplicate the old matrix and the other is to create a new metrix and fill it with values  -- i just outputed the iteration 4 for figure 2(b to d)  local colums = 5  local rows = 5  function createMatrixArray(rows, colums)  local matrixArray = {}  for i = 1, rows do  matrixArray[i] = {}  for j = 1, colums do  matrixArray[i][j] = ""  end  end  return matrixArray    end  function spawnProbability()  randomValue = math.random(1, 40000)  if (randomValue <= 5000) then  randomValue = 0  else  randomValue = 1  end  return randomValue  end  function fillMatrix(matrix, patternNumber)  for i = 1, rows do  for j = 1, colums do  --for week 9 deliverable  if(patternNumber>4) then  randomValue = spawnProbability()  matrix[i][j] = randomValue  else  matrix[i][j] = 1  end  end  end  --for the week 7 deliverable to get 2(a) figure state for the cell  if (patternNumber == 1) then  print("")  print("Figure 2.a")  matrix [2][3] = 0  matrix [3][3] = 0  matrix [4][3] = 0  --for the week 8 deliverable to get 2(b-d) figure state for the cell  elseif (patternNumber == 2) then  print("")  print("Figure 2.b")  matrix [2][2] = 0  matrix [2][3] = 0  matrix [3][2] = 0  matrix [3][3] = 0  elseif (patternNumber == 3) then  print("")  print("Figure 2.c")  matrix [2][2] = 0  matrix [3][3] = 0  matrix [4][4] = 0  elseif (patternNumber == 4) then  print("")  print("Figure 2.d")  matrix [1][1] = 0  matrix [2][2] = 0  matrix [2][3] = 0  matrix [3][1] = 0  matrix [3][2] = 0  end    end  function displayMatrix(matrix)  print("")  for i = 1, rows do  for j = 1, colums do  if matrix[i][j] == 0 then  io.write("O")  else  io.write("#")  end  end  print("")  end  end  function detectNeighbourCells(currentCell, x, y)  local neighbourCells = 0  for i = -1, 1 do  for j = -1, 1 do  if (i == 0 and j == 0) then  else  local row = ((x-1 + i + rows ) % rows)+1  local colum = ((y-1 + j +colums) % colums)+1    if currentCell[row][colum] == 0 then  neighbourCells = neighbourCells + 1  end    end  end    end  return neighbourCells  end  --[[ creates a empty matrix where it fills the matrix for the next state the according to the rules  function nextState(currentMatrix)  local nextStateMatrix = createMatrixArray(rows, colums)  for i = 1, rows do  for j = 1, colums do  x = i  y= j  neighbourDetected = detectNeighbourCells(currentMatrix, x, y)  cellLocated = currentMatrix[i][j]  if (cellLocated == 1) then  if neighbourDetected == 3 then  nextStateMatrix[i][j] = 0  else  nextStateMatrix[i][j] = 1  end  else  if (neighbourDetected <2 or neighbourDetected>3) then  nextStateMatrix[i][j] = 1  else  nextStateMatrix[i][j] = 0  end  end  end    end    displayMatrix(nextStateMatrix)  return nextStateMatrix  end  --]]  ---[[ duplicates the current matrix and changes the cells according to the rules  function nextState(currentMatrix)  local nextStateMatrix = duplicateArrayMatrix(currentMatrix)  for i = 1, rows do  for j = 1, colums do  neighbourDetected = detectNeighbourCells(currentMatrix, i, j)  cellLocated = currentMatrix[i][j]  if (cellLocated == 0) then  if(neighbourDetected <2 or neighbourDetected>3) then  nextStateMatrix[i][j] = 1  else  --cell stays the same  end  else  if(cellLocated == 1 and neighbourDetected == 3) then  nextStateMatrix[i][j] = 0  else  --cell stays the same  end  end  end    end      displayMatrix(nextStateMatrix)  return nextStateMatrix  end  function duplicateArrayMatrix(oldMatrix)  local copiedMatrix ={}  for i = 1, rows do  copiedMatrix[i] ={}  for j = 1, colums do  copiedMatrix[i][j]= oldMatrix[i][j]  end    end  return copiedMatrix  end  --]]  function simulate(matrix)    local currentMatrix = matrix  iterated =1;  iteration = 4;  while iterated <=iteration do  currentMatrix = nextState(currentMatrix)  iterated = iterated +1    end  end  function main()  patternNumber = 4  if patternNumber > 4 then  rows = 200  colums = 200  matrix = createMatrixArray(rows, colums)  else  matrix = createMatrixArray(rows, colums)  end    for startNumber = 2, patternNumber do  fillMatrix(matrix, startNumber)  displayMatrix(matrix)    simulate(matrix)  startNumber = startNumber+1  end    end  main() |
| 21/09/2023  7:00am | 21/09/2023  17:00am | 10 hours | 94 | -----------------------------------------------------------------------------------------  --  -- main.lua  -- Week 8 deliverable  -----------------------------------------------------------------------------------------  --Poorav Sharma  -- 0 and O are the cell  -- 1 and # is the space  -- i tried using the cell rules but coudn't make detect the neighbours properly -fixed this problem for week 8 deliverables i ended up with two ways to fix it one is to duplicate the old matrix and the other is to create a new metrix and fill it with values  -- i just outputed the iteration 4 for figure 2(b to d)  local display = require("display")  display.setDefault("background", 1, 1, 1)  local screenWidth = display.actualContentWidth  local screenHeight = display.actualContentHeight   -- CELL\_SIZE = 1.6  local cell\_Size = 10  local colums = 200  local rows = 200  local cellWidth = (screenWidth - 2 \* cell\_Size) / colums  local cellHeight = (screenHeight - 2 \* cell\_Size) / rows  local cellGroup = display.newGroup()  function createMatrixArray(rows, colums)      local matrixArray = {}      for i = 1, rows do          matrixArray[i] = {}          for j = 1, colums do              matrixArray[i][j] = ""          end      end      return matrixArray    end  function spawnProbability()      randomValue = math.random(1, 40000)      if (randomValue <= 5000) then          randomValue = 0      else          randomValue = 1      end      return randomValue  end  function fillMatrix(matrix, patternNumber)      for i = 1, rows do          for j = 1, colums do              --for week 9 deliverable              if(patternNumber>4) then                  randomValue = spawnProbability()                  matrix[i][j] = randomValue              else                  matrix[i][j] =  1              end          end      end      --for the week 7 deliverable to get 2(a) figure state for the cell      if (patternNumber == 1) then          print("")          print("Figure 2.a")          matrix [2][3] = 0          matrix [3][3] = 0          matrix [4][3] = 0      --for the week 8 deliverable to get 2(b-d) figure state for the cell      elseif (patternNumber == 2) then          print("")          print("Figure 2.b")          matrix [2][2] = 0          matrix [2][3] = 0          matrix [3][2] = 0          matrix [3][3] = 0      elseif (patternNumber == 3) then          print("")          print("Figure 2.c")          matrix [2][2] = 0          matrix [3][3] = 0          matrix [4][4] = 0      elseif (patternNumber == 4) then          print("")          print("Figure 2.d")          matrix [1][1] = 0          matrix [2][2] = 0          matrix [2][3] = 0          matrix [3][1] = 0          matrix [3][2] = 0      end    end  function displayMatrix(matrix)      print("")      for i = 1, rows do          for j = 1, colums do              if matrix[i][j] ==  0 then                  io.write("O")              else                  io.write("#")              end          end          print("")      end  end  function detectNeighbourCells(currentCell, x, y)      local neighbourCells = 0      for i = -1, 1 do          for j = -1, 1 do              if (i == 0 and j == 0) then              else               local row = ((x-1 + i + rows ) % rows)+1               local colum = ((y-1 + j +colums) % colums)+1                 if currentCell[row][colum] == 0 then                  neighbourCells = neighbourCells + 1               end                end          end        end      return neighbourCells  end  --[[ creates a empty matrix where it fills the matrix for the next state the according to the rules  function nextState(currentMatrix)      local nextStateMatrix = createMatrixArray(rows, colums)      for i = 1, rows do          for j = 1, colums do              x = i              y= j              neighbourDetected = detectNeighbourCells(currentMatrix, x, y)              cellLocated = currentMatrix[i][j]              if (cellLocated == 1) then                  if neighbourDetected == 3 then                      nextStateMatrix[i][j] = 0                  else                      nextStateMatrix[i][j] = 1                  end              else                  if (neighbourDetected <2 or neighbourDetected>3) then                      nextStateMatrix[i][j] = 1                  else                   nextStateMatrix[i][j] = 0                  end              end          end        end        displayMatrix(nextStateMatrix)      return nextStateMatrix  end  --]]  ---[[ duplicates the current matrix and changes the cells according to the rules  function nextState(currentMatrix)        local nextStateMatrix = duplicateArrayMatrix(currentMatrix)      for i = 1, rows do          for j = 1, colums do              neighbourDetected = detectNeighbourCells(currentMatrix, i, j)              cellLocated = currentMatrix[i][j]              if (cellLocated == 0) then                  if(neighbourDetected <2 or neighbourDetected>3) then                      nextStateMatrix[i][j] = 1                  else                      --cell stays the same                  end              else                  if(cellLocated == 1 and neighbourDetected == 3) then                      nextStateMatrix[i][j] = 0                  else                     --cell stays the same                  end              end          end        end        appDisplay(nextStateMatrix, rows, colums)    --  displayMatrix(nextStateMatrix)      return nextStateMatrix  end  function duplicateArrayMatrix(oldMatrix)      local copiedMatrix ={}      for i = 1, rows do          copiedMatrix[i] ={}          for j = 1, colums do              copiedMatrix[i][j]= oldMatrix[i][j]          end        end      return copiedMatrix  end  --]]  function simulate(matrix)        local currentMatrix = matrix      iterated =1;      iteration = 4;      local delayBetweenIterations = 1000 -- timer delay miliseconds      local function performNextIteration()         if  iterated <=iteration then              currentMatrix = nextState(currentMatrix)              iterated = iterated +1              timer.performWithDelay(delayBetweenIterations, performNextIteration)              print(iterated)          end      end      performNextIteration()  end  function appDisplay(matrix, rows, colums)      --CELL\_SIZE = 1.6      cellGroup:removeSelf()      cellGroup = nil      cellGroup = display.newGroup()      for i = 1, rows do          for j = 1, colums do              if matrix[i][j] == 0 then                  local square = display.newRect((j - 1) \* cellWidth, (i - 1) \* cellHeight, cellWidth, cellHeight)                  square:setFillColor(1, 0.75, 0.8) -- pink color for live cells                  cellGroup:insert(square)              else                  local square = display.newRect((j - 1) \* cellWidth, (i - 1) \* cellHeight, cellWidth, cellHeight)                  square:setFillColor(0, 0, 0) -- black color for live cells                  cellGroup:insert(square)              end          end      end        end  function main()            matrix = createMatrixArray(rows, colums)              fillMatrix(matrix, 5)             -- displayMatrix(matrix)              appDisplay(matrix, rows, colums)              timer.performWithDelay(1000, function()                  simulate(matrix)              end)             -- simulate(matrix)              --startNumber = startNumber+1      end    ----main----  main() |