```
GlowScript 2.6 VPython
## Written to deomo the use of recurssion
## to solve a problem.
## A. O. Hausknecht 2013 - 2017
## Updated to GlowScript 2.6 Fall 2017
N = 3
Board = None
BoardSize = None
BoardMinX = None
BoardMaxY = None
BoardLabel = None
Trominoes = []
NextFreeTromino = None
EmptyCell = None
BoardLabel = None
Running = False
Speed = 2
Reset = False


scene = canvas (title = "<h3>Tromino Puzzle V3 Fall 2017</h3>")
scene.width = 550
scene.height = 550
scene.center = vector(0,0,0)
scene.background = color.red
scene.userzoom = False
scene.usespin = False

MathJax.Hub.Queue(["Typeset",MathJax.Hub])


def setSpeed(menu):
    global Speed
    s = menu.selected
    if s == 'Slow':
        Speed = 1
    elif s == "Medium":
        Speed = 2
    elif s == "Fast":
        Speed = 4
    else:
        Speed = 8


def doSolveResume(btn):
```

```
    global Running
    Running = not Running
    if Running:
        solvePause.color = color.red
        solvePause.text = "Pause "
    else:
        solvePause.color = color.green
        solvePause.text = "Solve "

def resetPuzzle(btn):
    doReset(N)

def setBoardSize(m):
    global N, Reset
    s = m.selected
    if s == '4x4':
        N = 2
    elif s ==  '8x8':
        N = 3
    elif s == '16x16':
        N = 4
    else:
        N = 5
    Reset = True


solvePause = button(pos = scene.title_anchor, text="Solve ", background = color.green
button(pos = scene.title_anchor, text = "Reset", background = color.magenta, bind = r
scene.append_to_title("  Board Size:")
menu(pos = scene.title_anchor,  choices = ['4x4', '8x8', '16x16', '32x32'], selected
scene.append_to_title("  Solve Speed:")
menu(pos=scene.title_anchor, choices=["Slow", "Medium", "Fast", "Very Fast"], selecte



def createTromino(missingCorner):


    if  missingCorner == "TopLeft":
        blbox = box(pos = vector(-0.50+0.01, -0.50+0.01, 0), color = color.magenta, s
        trbox = box(pos = vector( 0.50-0.01,  0.50-0.01, 0), color = color.magenta, s
        brbox = box(pos = vector( 0.50-0.01, -0.50+0.01, 0), color = color.magenta, s
        tromino = compound( [ blbox,  trbox, brbox] )
    elif missingCorner == "BottomLeft":
        tlbox = box(pos = vector(-0.50+0.01,  0.50-0.01, 0), color = color.magenta, s
```

```
        trbox = box(pos = vector( 0.50-0.01,  0.50-0.01, 0), color = color.magenta, s
        brbox = box(pos = vector( 0.50-0.01, -0.50+0.01, 0), color = color.magenta, s
        tromino = compound( [ tlbox,  trbox, brbox ] )
    elif missingCorner == "TopRight":
        tlbox = box(pos = vector(-0.50+0.01,  0.50-0.01, 0), color = color.magenta, s
        blbox = box(pos = vector(-0.50+0.01, -0.50+0.01, 0), color = color.magenta, s
        brbox = box(pos = vector( 0.50-0.01, -0.50+0.01, 0), color = color.magenta, s
        tromino = compound( [ tlbox, blbox, brbox] )
    elif missingCorner == "BottomRight":
        tlbox = box(pos = vector(-0.50+0.01,  0.50-0.01, 0), color = color.magenta, s
        blbox = box(pos = vector(-0.50+0.01, -0.50+0.01, 0), color = color.magenta, s
        trbox = box(pos = vector( 0.50-0.01,  0.50-0.01, 0), color = color.magenta, s
        tromino = compound( [ tlbox, blbox, trbox] )
    else:
        alert("Error in createTromino")
    tromino.visible = False
    return tromino


def drawTromino(missingCorner, x, y):
    global Trominoes
    screenX = BoardMinX + x + 1
    screenY = BoardMaxY - y - 1
    tromino =  createTromino(missingCorner)
    tromino.pos = vector(screenX, screenY, 0)
    Trominoes.push(tromino)


def createPuzzle(n):
    global  BoardSize, BoardMaxY, BoardMinX
    global Board, EmptyCell, BoardLabel
    BoardSize = 2**n
    BoardMaxY =  BoardSize/2
    BoardMinX = -BoardMaxY
    scene.range = BoardMaxY+2
    if Board == None:
       Board     = box(pos = vector(0, 0, -0.1), color = color.black, size = vector(B
       EmptyCell = box(pos = vector(0, 0, -0.1), color = color.cyan,  size = vector(0
    else:
       Board.size = vector(BoardSize+0.2, BoardSize+0.2, 0.01)
       EmptyCell.visible = False

    if BoardLabel == None:
        BoardLabel = label(pos = vector(0, BoardMaxY+1, 0), text = "", height = 18, b
        BoardLabel.box = False
```

```
    elif n < 6:
        BoardLabel.pos = vector(0, BoardMaxY+1, 0)
    else:
        BoardLabel.pos = vector(0, BoardMaxY+1, 0)



def getMissingCorner(eX, eY):
    if (eY == 0) and (eX == 0):
        return "TopLeft"
    elif (eY == 0) and (eX == 1):
        return "TopRight"
    elif (eY == 1) and (eX == 0):
        return "BottomLeft"
    else:
        return "BottomRight"


def coverBoard(x0, y0, xE, yE, n):
    if n==1:
        drawTromino( getMissingCorner(xE-x0, yE-y0), x0, y0)
    else:
        newWidth = 2**(n-1)
        x1 = x0+newWidth; y1 = y0+newWidth
        dX = xE - x0;  dY = yE - y0
        if (0<=dX) and (dX<newWidth): ## Left-half of contains the empty cell
            if 0<=dY and dY<newWidth: ## Top Left contains empty cell
                coverBoard(x0, y0,   xE,   yE, n-1) ## TopLeft sub-board
                coverBoard(x0, y1, x1-1,   y1, n-1) ## BottomLeft sub-board
                coverBoard(x1, y1,   x1,   y1, n-1) ## BottomRight sub-board
                coverBoard(x1, y0,   x1, y1-1, n-1) ## TopRight sub-board
                drawTromino("TopLeft",  x1-1, y1-1) ## Cover missing Tromino
            else: ## Bottom Left contains empty cell
                coverBoard(x0, y0, x1-1, y1-1, n-1)   ## TopLeft sub-board
                coverBoard(x0, y1,   xE,   yE, n-1)   ## BottomLeft sub-board
                coverBoard(x1, y1,   x1,   y1, n-1)   ## BottomRight sub-board
                coverBoard(x1, y0,   x1, y1-1, n-1)   ## TopRight sub-board
                drawTromino("BottomLeft", x1-1, y1-1) ## Cover missing Tromino

        else: ## Right-half of contains the empty cell
            if (0<=dY) and (dY<newWidth): ## Top Right contains empty cell
                coverBoard(x0, y0, x1-1, y1-1, n-1) ## TopLeft sub-board
                coverBoard(x0, y1, x1-1,   y1, n-1) ## BottomLeft sub-board
                coverBoard(x1, y1,   x1,   y1, n-1) ## BottomRight sub-board
                coverBoard(x1, y0,   xE,   yE, n-1) ## TopRight sub-board
```

```
                drawTromino("TopRight", x1-1, y1-1) ## Cover missing Tromino
            else: ## bottom Right contains empty cell
                coverBoard(x0, y0, x1-1, y1-1, n-1) ## TopLeft sub-board
                coverBoard(x0, y1, x1-1,   y1, n-1) ## BottomLeft sub-board
                coverBoard(x1, y1,   xE,   yE, n-1) ## BottomRight sub-board
                coverBoard(x1, y0,   x1, y1-1, n-1) ## TopRight sub-board
                drawTromino("BottomRight", x1-1, y1-1) ## Cover missing Tromino


def doSolveWithRandomEmptyCell(n):
    global NextFreeTromino
    bSide = 2**n
    xE = Math.floor(bSide*Math.random())
    yE = Math.floor(bSide*Math.random())
    solvePuzzle(xE, yE, n)
    NextFreeTromino = 0



def solvePuzzle(xE, yE, n):
    global BoardSize, BoardLabel
    global EmptyCell, BoardMaxY, BoardMinX
    global depth, maxDepth
    createPuzzle(n)
    bMax  = BoardSize-1
    if (xE < 0)  or  (xE >= BoardSize):
        alert("the x-coordinate "+ xE + " of the empty cell must be between zero and
        return

    if (yE < 0)  or (yE >= BoardSize):
        alert("the y-coordinate "+ yE + " of the empty cell must be between zero and
        return

    tCount = (2**(2*n)-1)/3
    str  = "Empty Cell: ("+ xE + ", "+ yE + "); "
    str += "Board Size: "+ BoardSize +" x "+ BoardSize +"; "
    str += "Tromino Count: "+tCount
    BoardLabel.text = str
    EmptyCell.pos = vector(BoardMinX + xE + 0.5, BoardMaxY - yE - 0.5, 0)
    EmptyCell.visible = True
    coverBoard(0, 0, xE, yE, n)


def doReset(n):
    global Reset, Running, Trominoes, n
```

```
    global NextFreeTromino, solvePause
    Reset = False
    for i in range(len(Trominoes)):
        Trominoes[i].visible = False

    Trominoes = []
    NextFreeTromino = 0
    solvePause.text = "Solve "
    solvePause.background = color.green
    Running = False
    doSolveWithRandomEmptyCell(n)



def displayNextTromino():
    global NextFreeTromino, Running
    if NextFreeTromino < len(Trominoes):
        t = Trominoes[NextFreeTromino]
        t.visible = True
        NextFreeTromino += 1
    else:
        Running = False
        solvePause.text = "Solved!"
        solvePause.background = color.cyan


createPuzzle(N)
doSolveWithRandomEmptyCell(N)
## Animation loop
while True:
       rate(Speed, wait)
       if (Running): displayNextTromino()
       if (Reset): doReset(N)
```