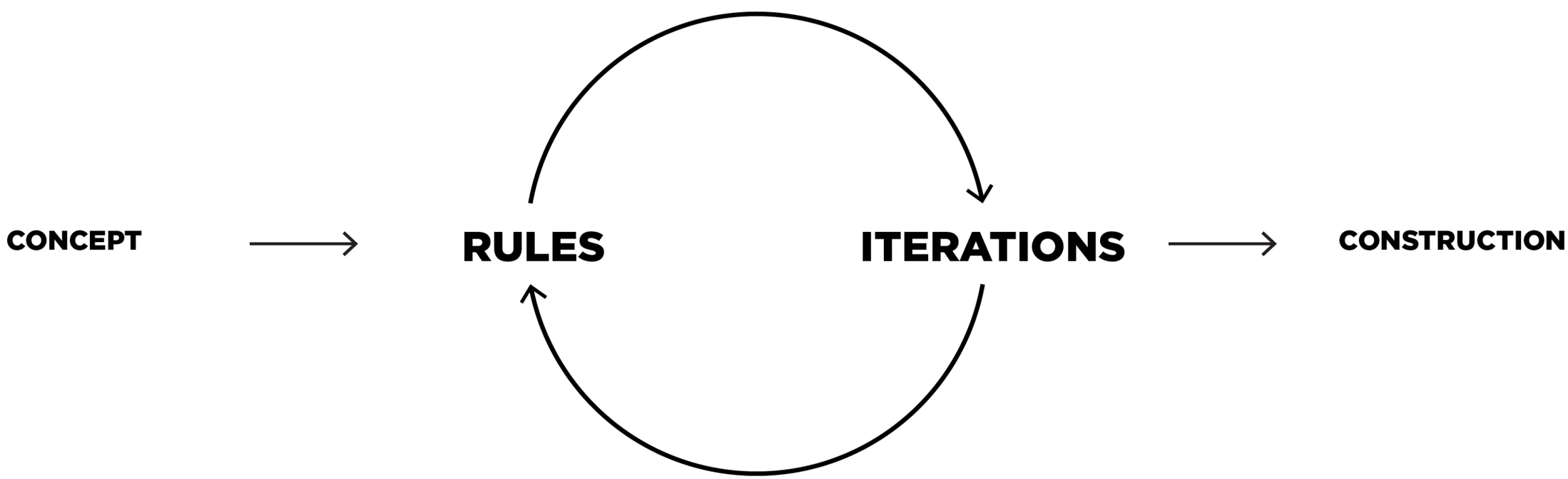
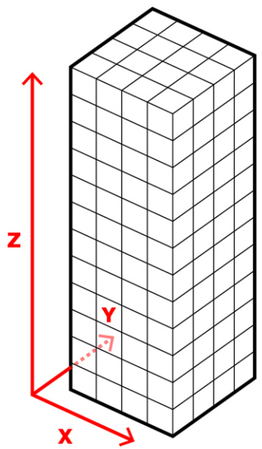
AUTOMATA TOWER

JAEYUAL LEE + JUNEY LEE

4.553 • PROGRAMMING SKETCHES | JUHONG PARK

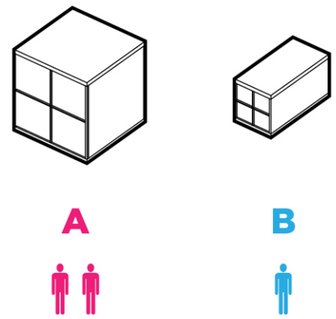
CONCEPTUAL DESIGN → **SCHEMATIC DESIGN** → **DESIGN DEVELOPMENT** → **CONSTRUCTION DOCUMENTS**


CONCEPT
RULES
ITERATIONS
CONSTRUCTION

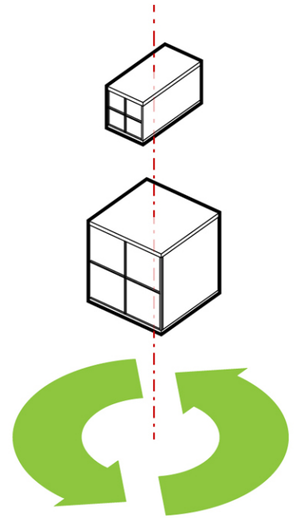


01 BOUNDARY

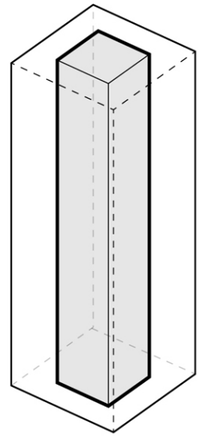
02 DISTRIBUTION RULE



03 CELL TYPES

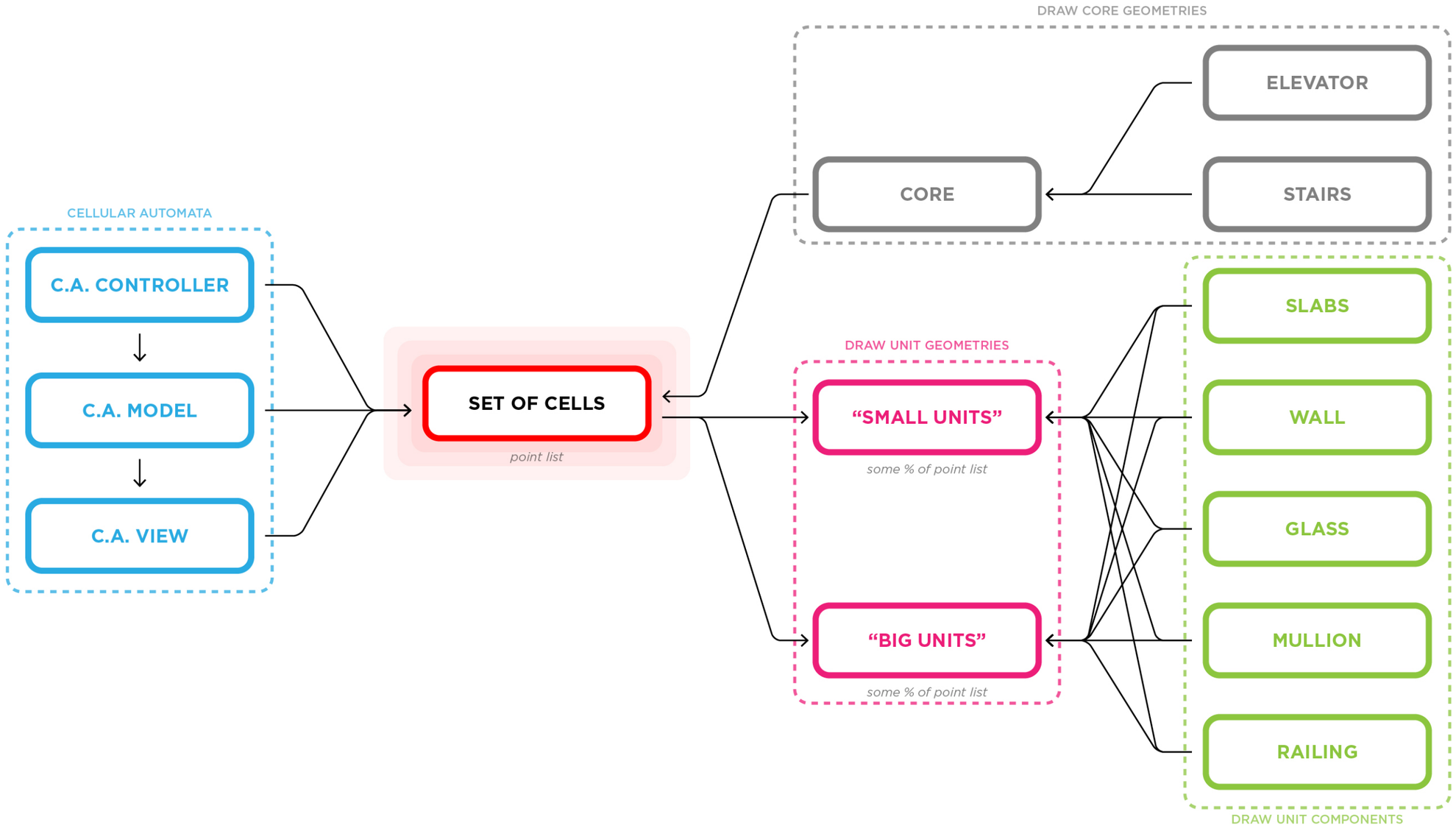


04 RANDOM ROTATE



05 CORE

t


DRAW CORE GEOMETRIES
ELEVATOR
CORE
STAIRS
CELLULAR AUTOMATA
C.A. CONTROLLER
C.A. MODEL
C.A. VIEW
SET OF CELLS
point list
DRAW UNIT GEOMETRIES
"SMALL UNITS"
some % of point list
"BIG UNITS"
some % of point list
SLABS
WALL
GLASS
MULLION
RAILING
DRAW UNIT COMPONENTS

GLOBAL VARIABLES

- Xnumber
- Ynumber
- Znumber
- smallunitpercentage
- iterations

CELLULAR AUTOMATA

- CELL
- SET OF CELLS
- AUTOMATA CONTROLLER
- AUTOMATA MODEL
- AUTOMATA VIEW

UNIT TYPES

- SMALL UNIT
- BIG UNIT

UNIT COMPONENTS

- SLABMAKER
- WALL
- GLASS
- MULLION
- RANDOMANGLE

CORE

- CORE
- ELEVATOR
- BUILDSTAIRS

GLOBAL VARIABLES

Xnumber

Ynumber

Znumber

smallunitpercentage

iterations

CELLULAR AUTOMATA

CELL

SET OF CELLS

AUTOMATA CONTROLLER

AUTOMATA MODEL

AUTOMATA VIEW

UNIT TYPES

SMALL UNIT

BIG UNIT

UNIT COMPONENTS

SLABMAKER

WALL

GLASS

MULLION

RANDOMANGLE

CORE

CORE

ELEVATOR

BUILDSTAIRS

```
class Cell():

def __init__(self, x, y, z):
    self.X = x
    self.Y = y
    self.Z = z
    self.Pos = [x,y,z]
    self.Visibility = False
    self.N = 0

def getVisibility(self):
    return self.Visibility

def setVisibility(self, v):
    self.Visibility = v

def getN(self):
    return self.N

def setN(self, n):
    self.N = n

def getX(self):
    return self.X

def getY(self):
    return self.Y

def getZ(self):
    return self.Z

def getPos(self):
    return self.Pos

def updateVisibility(self):
    if (self.Visibility == True):
        self.rule1_Visibility()
    else: #(self.Visibility == False)
        self.rule2_Population()
```

RULE FOR LIVE CELLS

```
def rule1_Visibility(self):
    if (self.N < 10):
        self.setVisibility(False)
    elif (self.N < 18):
        self.setVisibility(True)
    else:
        self.setVisibility(False)
```

RULE FOR DEAD CELLS

```
def rule2_Population(self):
    if (self.N == 0 or self.N == 5 or self.N == 10):
        self.setVisibility(True)
    elif (self.N < 10):
        self.setVisibility(False)
    elif (self.N < 18):
        self.setVisibility(True)
    else:
        pass
```

GLOBAL VARIABLES

Xnumber

Ynumber

Znumber

smallunitpercentage

iterations

CELLULAR AUTOMATA

CELL

SET OF CELLS

AUTOMATA CONTROLLER

AUTOMATA MODEL

AUTOMATA VIEW

UNIT TYPES

SMALL UNIT

BIG UNIT

UNIT COMPONENTS

SLABMAKER

WALL

GLASS

MULLION

RANDOMANGLE

CORE

CORE

ELEVATOR

BUILDSTAIRS

```
class SetOfCells():

    def __init__(self):
        self.ListCells = []
        self.generateCells()
        self.setSeeds()

    def generateCells(self):
        xCellList = []
        for x in range(Xnumber):
            yCellList = []
            for y in range(Ynumber):
                zCellList = []
                for z in range(Znumber):
                    c = Cell(x,y,z)
                    zCellList.append(c)
                yCellList.append(zCellList)
            xCellList.append(yCellList)
        self.ListCells = xCellList  # returns the nested list into self.ListCells

    def setSeeds(self):
        for x in range(Xnumber):
            for y in range(Ynumber):
                for z in range(Znumber):
                    cell = self.ListCells[x][y][z]
                    if (x > 0 and x < Xnumber and y > 0 and y < Ynumber and z > 0 and z < Znumber):
                        cell.setVisibility(True)
                    else:
                        cell.setVisibility(False)

    def getCells(self):
        return self.ListCells
```

GLOBAL VARIABLES

Xnumber

Ynumber

Znumber

smallunitpercentage

iterations

CELLULAR AUTOMATA

CELL

SET OF CELLS

AUTOMATA CONTROLLER

AUTOMATA MODEL

AUTOMATA VIEW

UNIT TYPES

SMALL UNIT

BIG UNIT

UNIT COMPONENTS

SLABMAKER

WALL

GLASS

MULLION

RANDOMANGLE

CORE

CORE

ELEVATOR

BUILDSTAIRS

```
class AutomataController():

    def __init__(self):
        self.ListCells = []
        self.SC = SetOfCells()
        self.M = AutomataModel()

    def initiateModel(self):
        self.ListCells = self.SC.getCells()
        self.M.setListCells(self.ListCells)
        self.M.repeatProcesses()
```

GLOBAL VARIABLES

- Xnumber
- Ynumber
- Znumber
- smallunitpercentage
- iterations

CELLULAR AUTOMATA

- CELL
- SET OF CELLS
- AUTOMATA CONTROLLER
- AUTOMATA MODEL
- AUTOMATA VIEW

UNIT TYPES

- SMALL UNIT
- BIG UNIT

UNIT COMPONENTS

- SLABMAKER
- WALL
- GLASS
- MULLION
- RANDOMANGLE

CORE

- CORE
- ELEVATOR
- BUILDSTAIRS

```
class AutomataModel():

    def __init__(self):
        self.ListCells = []
        self.V = AutomataView()

    def setListCells(self, lc):
        self.ListCells = lc

    def repeatProcesses(self):
        iterationNumber = iterations
        for i in range(iterationNumber):
            self.checkNeighboringCells()
            self.updateCellVisibility()
            self.V.setListCells(self.ListCells)
            points = self.V.getCoordinates()
            center = self.V.getCenter()
            self.V.makeUnits (points, center)
            self.V.moveGeometries()

    def checkNeighboringCells(self):
        for i in range(1,Xnumber-1):
            for j in range(1,Ynumber-1):
                for k in range(1,Znumber-1):
                    cell = self.ListCells[i][j][k]
                    x = cell.getX()
                    y = cell.getY()
                    z = cell.getZ()
                    n = self.calculateNumberOfNeighboringCells(x,y,z)
                    cell.setN(n)

    def calculateNumberOfNeighboringCells(self, x, y, z):
        cell0 = self.ListCells[x-1][y-1][z-1]
        cell1 = self.ListCells[x][y-1][z-1]
        cell2 = self.ListCells[x+1][y-1][z-1]
        cell3 = self.ListCells[x-1][y][z-1]
        cell4 = self.ListCells[x][y][z-1]
        cell5 = self.ListCells[x+1][y][z-1]
        cell6 = self.ListCells[x-1][y+1][z-1]
        cell7 = self.ListCells[x][y+1][z-1]
        cell8 = self.ListCells[x+1][y+1][z-1]

        cell9 = self.ListCells[x-1][y-1][z]
        cell10 = self.ListCells[x][y-1][z]
        cell11 = self.ListCells[x+1][y-1][z]
        cell12 = self.ListCells[x-1][y][z]
        cell13 = self.ListCells[x+1][y][z]
        cell14 = self.ListCells[x-1][y+1][z]
        cell15 = self.ListCells[x][y+1][z]
        cell16 = self.ListCells[x+1][y+1][z]

        cell17 = self.ListCells[x-1][y-1][z+1]
        cell18 = self.ListCells[x][y-1][z+1]
        cell19 = self.ListCells[x+1][y-1][z+1]
        cell20 = self.ListCells[x-1][y][z+1]
        cell21 = self.ListCells[x][y][z+1]
        cell22 = self.ListCells[x+1][y][z+1]
        cell23 = self.ListCells[x-1][y+1][z+1]
        cell24 = self.ListCells[x][y+1][z+1]
        cell25 = self.ListCells[x+1][y+1][z+1]

        v0 = cell0.getVisibility()
        v1 = cell1.getVisibility()
        v2 = cell2.getVisibility()
        v3 = cell3.getVisibility()
        v4 = cell4.getVisibility()
        v5 = cell5.getVisibility()
        v6 = cell6.getVisibility()
        v7 = cell7.getVisibility()
        v8 = cell8.getVisibility()
        v9 = cell9.getVisibility()
        v10 = cell10.getVisibility()
        v11 = cell11.getVisibility()
        v12 = cell12.getVisibility()
        v13 = cell13.getVisibility()
        v14 = cell14.getVisibility()
        v15 = cell15.getVisibility()
        v16 = cell16.getVisibility()
        v17 = cell17.getVisibility()
        v18 = cell18.getVisibility()
        v19 = cell19.getVisibility()
        v20 = cell20.getVisibility()
        v21 = cell21.getVisibility()
        v22 = cell22.getVisibility()
        v23 = cell23.getVisibility()
        v24 = cell24.getVisibility()
        v25 = cell25.getVisibility()

        sum =
v0+v1+v2+v3+v4+v5+v6+v7+v8+v9+v10+v11+v12+v13+v14+v15+v
16+v17+v18+v19+v20+v21+v22+v23+v24+v25

        return sum

    def updateCellVisibility(self):
        for x in range(1,Xnumber-1):
            for y in range(1,Ynumber-1):
                for z in range(1,Znumber-1):
                    cell = self.ListCells[x][y][z]
                    cell.updateVisibility()
```

MAKE GEOMETRIES
INHERITED FROM "VIEW"

CHECK NEIGHBORS

UPDATE VISIBILITY

GLOBAL VARIABLES

- Xnumber
- Ynumber
- Znumber
- smallunitpercentage
- iterations

CELLULAR AUTOMATA

- CELL
- SET OF CELLS
- AUTOMATA CONTROLLER
- AUTOMATA MODEL
- AUTOMATA VIEW

UNIT TYPES

- SMALL UNIT
- BIG UNIT

UNIT COMPONENTS

- SLABMAKER
- WALL
- GLASS
- MULLION
- RANDOMANGLE

CORE

- CORE
- ELEVATOR
- BUILDSTAIRS

smallunitpercetange = make smallunit
bigunitpercentage = makebigunit

```
class AutomataView():

def __init__(self):
    self.ListCells = []
    self.S = smallunit ()
    self.B = bigunit ()

def setListCells(self, lc):
    self.ListCells = lc

def getCoordinates(self):
    ptlist = []
    for x in range(Xnumber):
        for y in range(Ynumber):
            for z in range(Znumber):
                cell = self.ListCells[x][y][z]
                visibility = cell.getVisibility()
                if visibility == True:
                    pos = cell.getPos()
                    ptlist.append (pos)
    length = len (ptlist)
    return ptlist

def getCenter(self):
    ptlist = []
    for x in range(Xnumber):
        for y in range(Ynumber):
            for z in range(Znumber):
                cell = self.ListCells[x][y][z]
                visibility = cell.getVisibility()
                if visibility == True:
                    pos = cell.getPos()
                    shift = 0.5
                    center = [pos[0]+shift, pos[1]+shift, pos[2]]
                    ptlist.append (center)
    print ptlist
    return ptlist
```

GET COORDINATES FROM
CA OPERATION

```
def makeUnits (self, points, center):
    length = len (points)
    print "point list =", points
    print "point list length =", length
    percent = smallunitpercentage * length
    minority = int(percent)
    print "minority # =", minority
    majority = length - minority

    minorityrandomlist = []
    for i in range(minority):
        randomvalues = random.randint (0, length-1)
        minorityrandomlist.append (randomvalues)

    minorityrandomlist = list (set(minorityrandomlist))
    minorityrandomlist.sort()
    print "minority random list =", minorityrandomlist

    majorityrandomlist = []
    for i in range (length):
        majorityrandomlist.append (i)
    print majorityrandomlist =", majorityrandomlist

    for x in minorityrandomlist:
        majorityrandomlist.remove (x)

    self.S.makeunit (minorityrandomlist, points, center)
    self.B.makeunit (majorityrandomlist, points, center)

def moveGeometries(self):
    #rs.Sleep(1000)
    allObjIDs = rs.AllObjects()
    trans = [0,0,Znumber-1]
    rs.MoveObjects(allObjIDs, trans)
```

GLOBAL VARIABLES

- Xnumber
- Ynumber
- Znumber
- smallunitpercentage
- iterations

CELLULAR AUTOMATA

- CELL
- SET OF CELLS
- AUTOMATA CONTROLLER
- AUTOMATA MODEL
- AUTOMATA VIEW

UNIT TYPES

- SMALL UNIT
- BIG UNIT

UNIT COMPONENTS

- SLABMAKER
- WALL
- GLASS
- MULLION
- RANDOMANGLE

CORE

- CORE
- ELEVATOR
- BUILDSTAIRS

```
class smallunit():

    def __init__(self):
        self.slab = slabmaker()
        self.wall = wall()
        self.glass = glass()
        self.mullion = mullion_s()
        self.railing = railing()
        self.randomangle = randomangle()

        self.slabthickness = 0.05
        self.wallthickness = 0.02
        self.glassoffset = 0.02

        self.railingheight = 0.1
        self.floorheight = 0.5
        self.xdimension = 0.5
        self.ydimension = 1

    def makeunit (self, unitnumber, points, center):

        for r in unitnumber :
            rs.AddLayer("Small Unit", Color.Red)
            rs.CurrentLayer("Small Unit")

            x = points [r][0]
            y = points [r][1]
            z = points [r][2]

            pt0 = [x, y, z]
            pt1 = [x, y + self.ydimension , z]
            pt2 = [x + self.xdimension, y + self.ydimension, z]
            pt3 = [x + self.xdimension, y, z]

            rectangle = rs.AddRectangle(points[r], self.xdimension,
self.ydimension)

            slabs = self.slab.slabmaker (rectangle, self.slabthickness,
self.floorheight)
            wall1 = self.wall.wall(pt0, self.wallthickness,
self.ydimension, self.floorheight, self.slabthickness)
            wall2 = self.wall.wall(pt2, -self.wallthickness,
-self.ydimension, self.floorheight, self.slabthickness)

            glass1 = self.glass.glass(pt2, pt1, pt0, self.glassoffset,
self.floorheight, self.slabthickness)
            glass2 = self.glass.glass(pt0, pt3, pt1, self.glassoffset,
self.floorheight, self.slabthickness)

            mullion1 = self.mullion.mullion(pt2, pt1, pt0,
-self.glassoffset, self.floorheight, self.slabthickness)
            mullion2 = self.mullion.mullion(pt0, pt3, pt1,
self.glassoffset, self.floorheight, self.slabthickness)

            railing = self.railing.railingmaker(rectangle,
self.railingheight, self.floorheight)

            angle = self.randomangle.generator()

            verticalshift = [0, 0.5]
            shift = choice(verticalshift)
            path = [0, 0, shift]

            rs.RotateObjects(slabs, center[r], angle)
            rs.RotateObjects(wall1, center[r], angle)
            rs.RotateObjects(wall2, center[r], angle)
            rs.RotateObjects(glass1, center[r], angle)
            rs.RotateObjects(glass2, center[r], angle)
            rs.RotateObjects(mullion1, center[r], angle)
            rs.RotateObjects(mullion2, center[r], angle)
            rs.RotateObjects(railing, center[r], angle)

            rs.MoveObjects(slabs, path)
            rs.MoveObjects(wall1, path)
            rs.MoveObjects(wall2, path)
            rs.MoveObjects(glass1, path)
            rs.MoveObjects(glass2, path)
            rs.MoveObjects(mullion1, path)
            rs.MoveObjects(mullion2, path)
            rs.MoveObjects(railing, path)

            #rs.DeleteObject(path)

            rs.DeleteObject(rectangle)
```

GLOBAL VARIABLES

- Xnumber
- Ynumber
- Znumber
- smallunitpercentage
- iterations

CELLULAR AUTOMATA

- CELL
- SET OF CELLS
- AUTOMATA CONTROLLER
- AUTOMATA MODEL
- AUTOMATA VIEW

UNIT TYPES

- SMALL UNIT
- BIG UNIT

UNIT COMPONENTS

- SLABMAKER
- WALL
- GLASS
- MULLION
- RANDOMANGLE

CORE

- CORE
- ELEVATOR
- BUILDSTAIRS

```
class bigunit():

    def __init__(self):
        self.slab = slabmaker()
        self.wall = wall()
        self.glass = glass()
        self.mullion = mullion_b()
        self.railing = railing()
        self.randomangle = randomangle()

        self.slabthickness = 0.05
        self.wallthickness = 0.02
        self.glassoffset = 0.02

        self.railingheight = 0.1
        self.floorheight = 1
        self.xdimension = 1
        self.ydimension = 1

    def makeunit (self, unitnumber, points, center):
        for r in unitnumber :
            rs.AddLayer("Large Unit", Color.Blue)
            rs.CurrentLayer("Large Unit")

            x = points [r][0]
            y = points [r][1]
            z = points [r][2]

            pt0 = [x, y, z]
            pt1 = [x, y + self.ydimension , z]
            pt2 = [x + self.xdimension, y + self.ydimension, z]
            pt3 = [x + self.xdimension, y, z]

            rectangle = rs.AddRectangle(points[r], self.xdimension,
self.ydimension)
            slab = self.slab.slabmaker (rectangle, self.slabthickness,
self.floorheight)

            wall1 = self.wall.wall(pt0, self.wallthickness,
self.ydimension, self.floorheight, self.slabthickness)
            wall2 = self.wall.wall(pt2, -self.wallthickness,
-self.ydimension, self.floorheight, self.slabthickness)

            glass1 = self.glass.glass(pt2, pt1, pt0, self.glassoffset,
self.floorheight, self.slabthickness)
            glass2 = self.glass.glass(pt0, pt3, pt1, self.glassoffset,
self.floorheight, self.slabthickness)

            mullion1 = self.mullion.mullion(pt2, pt1, pt0,
-self.glassoffset, self.floorheight, self.slabthickness)
            mullion2 = self.mullion.mullion(pt0, pt3, pt1,
self.glassoffset, self.floorheight, self.slabthickness)

            railing = self.railing.railingmaker(rectangle,
self.railingheight, self.floorheight)

            angle = self.randomangle.generator()

            slab = rs.RotateObjects(slab, center[r], angle)

            wall1 = rs.RotateObjects(wall1, center[r], angle)
            wall2 = rs.RotateObjects(wall2, center[r], angle)
            glass1 = rs.RotateObjects(glass1, center[r], angle)
            glass2 = rs.RotateObjects(glass2, center[r], angle)
            mullion1 = rs.RotateObjects(mullion1, center[r], angle)
            mullion2 = rs.RotateObjects(mullion2, center[r], angle)
            railing = rs.RotateObjects(railing, center[r], angle)

            rs.DeleteObject(rectangle)
```

GLOBAL VARIABLES

- Xnumber
- Ynumber
- Znumber
- smallunitpercentage
- iterations

CELLULAR AUTOMATA

- CELL
- SET OF CELLS
- AUTOMATA CONTROLLER
- AUTOMATA MODEL
- AUTOMATA VIEW

UNIT TYPES

- SMALL UNIT
- BIG UNIT

UNIT COMPONENTS

- SLABMAKER
- WALL
- GLASS
- MULLION
- RANDOMANGLE

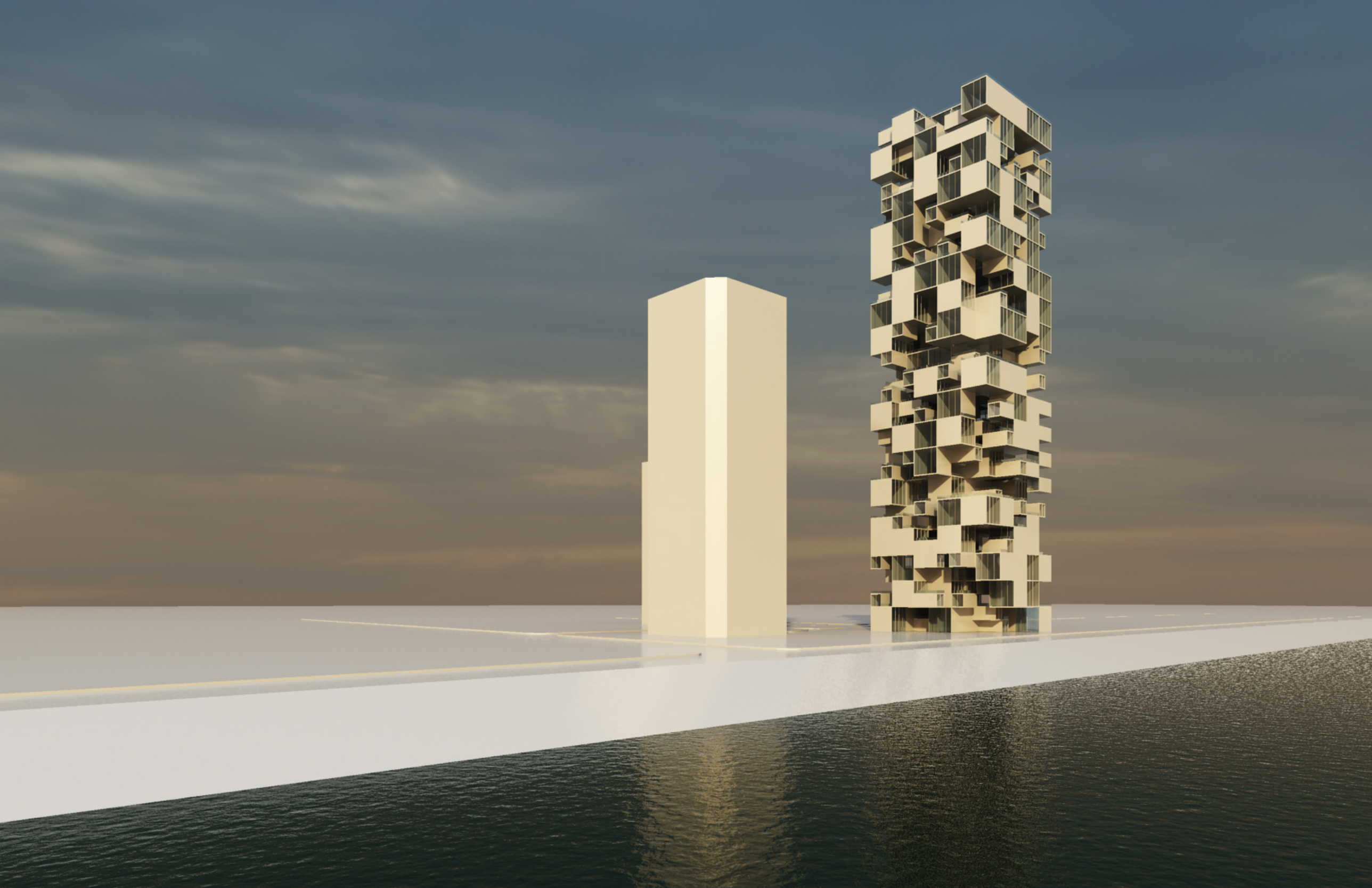
CORE

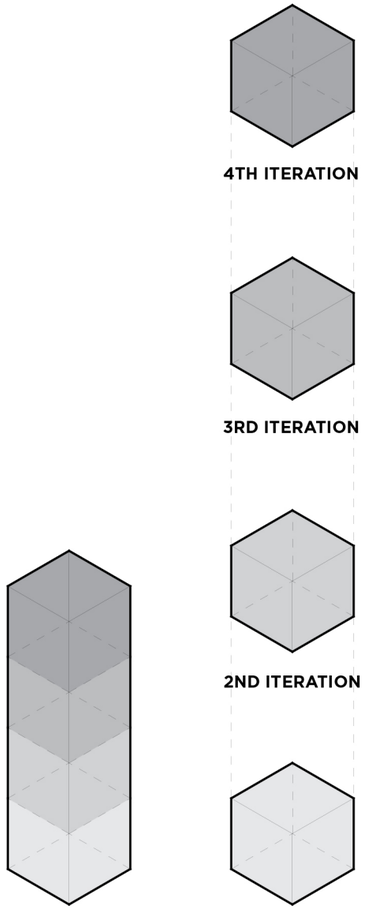
- CORE
- ELEVATOR
- BUILDSTAIRS

```
class Site():

    def __init__ (self):
        self.p = rs.WorldXYPlane()
        self.w = 100
        self.h = 100

    def makeSite(self):
        site_boundary = rs.AddRectangle(self.p, self.w, self.h)
        site = rs.AddPlanarSrf(site_boundary)

s = Site()
site = s.makeSite()
```

```
class Core():

    def __init__ (self):
        self.center = [40,40,0]
        self.p = rs.WorldXYPlane()
        self.w = 20
        self.h = 20
        self.core_h = 400

    def makeCore(self):
        core_boundary = rs.AddRectangle(self.p, self.w, self.h)
        core_boundary = rs.MoveObject(core_boundary,
self.center)
        path = rs.AddLine([0,0,0], [0,0,self.core_h])
        core = rs.ExtrudeCurve(core_boundary, path)
        rs.DeleteObject(path)
        core = rs.CapPlanarHoles(core)
        return core

c = Core()
core = c.makeCore()
```

```
class Elevator():

    def __init__ (self):
        self.core_h = 400
        self.ptS = [50,0,0]
        self.ptE = [50,100,0]
        self.p = rs.WorldZXPlane()
        self.h = 6
        self.w = 7
        self.trans = [42,60,0]
        self.trans2 = [52,60,0]
        self.n_floors = 40
        self.f_h = 10

    def makeECoreWall(self):
        pt1 = [41,51,0]
        pt2 = [49,51,0]
        pt3 = [49,59,0]
        pt4 = [41,59,0]
        pts = [pt1,pt2,pt3,pt4,pt1]
        ecorefloor_l = rs.AddPolyline(pts)
        path = rs.AddLine([0,0,0], [0,0,self.core_h])
        ecorefloor = rs.ExtrudeCurve(ecorefloor_l, path)
        rs.DeleteObject(path)
        ecorefloor2 = rs.MirrorObject(ecorefloor, self.ptS, self.ptE,
True)
        return ecorefloor
        return ecorefloor2

    def makeEdoor(self):
        Edoor_l = rs.AddRectangle(self.p, self.w, self.h)
        Edoor_l = rs.MoveObject(Edoor_l, self.trans)
        path = rs.AddLine([0,0,0],[0,-1,0])
        Edoor = rs.ExtrudeCurve(Edoor_l, path)
        rs.DeleteObject(path)
        rs.DeleteObject(Edoor_l)
        return Edoor

    def makeEdoor2(self):
        Edoor_l = rs.AddRectangle(self.p, self.w, self.h)
        Edoor_l = rs.MoveObject(Edoor_l, self.trans2)
        path = rs.AddLine([0,0,0],[0,-1,0])
        Edoor2 = rs.ExtrudeCurve(Edoor_l, path)
        rs.DeleteObject(path)
        rs.DeleteObject(Edoor_l)
        return Edoor2

    def copyEdoor(self, Edoor):
        index = range(1, self.n_floors)
        for i in index:
            trans2 = [0, 0, self.f_h * i]
            objID2 = rs.CopyObject(Edoor, trans2)

    def copyEdoor2(self, Edoor2):
        index = range(1, self.n_floors)
        for i in index:
            trans2 = [0, 0, self.f_h * i]
            objID2 = rs.CopyObject(Edoor2, trans2)

    def makeECore(self):
        ecore_wall = self.makeECoreWall()
        edoor = self.makeEdoor()
        edoors = self.copyEdoor(edoor)
        edoor2 = self.makeEdoor2()
        edoors2 = self.copyEdoor2(edoor2)

e = Elevator()
e.makeECore()
```



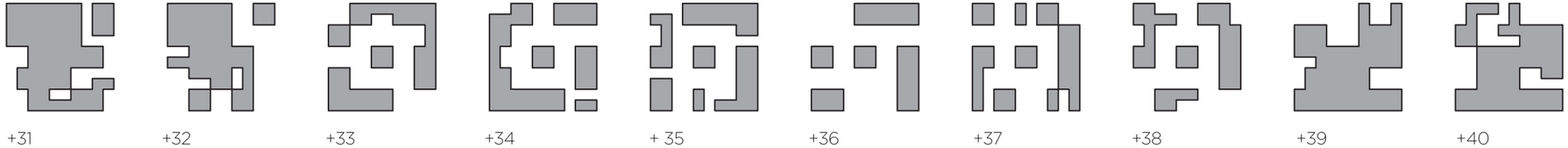


4TH ITERATION

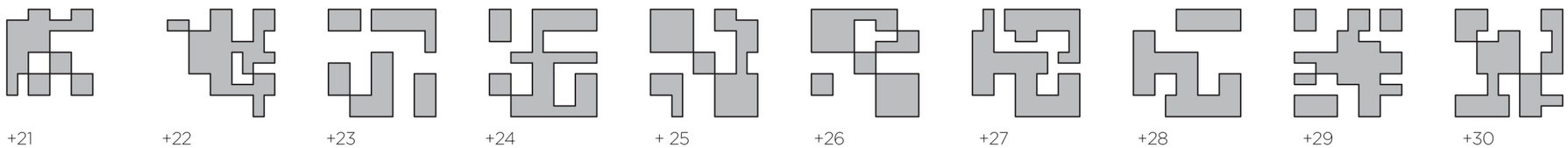
3RD ITERATION

2ND ITERATION

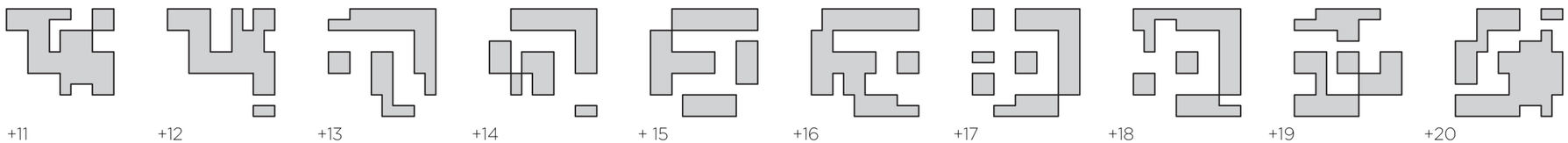
1ST ITERATION



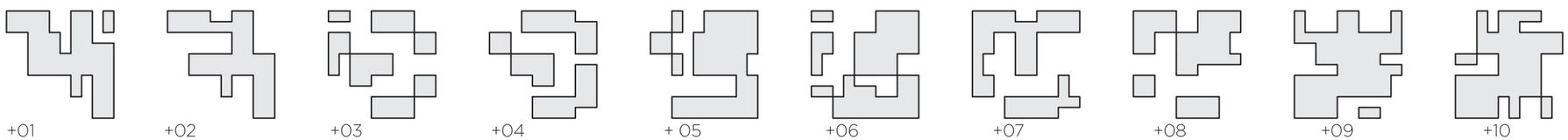
+31 +32 +33 +34 + 35 +36 +37 +38 +39 +40



+21 +22 +23 +24 + 25 +26 +27 +28 +29 +30



+11 +12 +13 +14 + 15 +16 +17 +18 +19 +20



+01 +02 +03 +04 + 05 +06 +07 +08 +09 +10