Requirements

**Use Case:** user wants to change selected room

**Use Case:** user wants to get updated furnishing suggestions

**Use Case:** selecting an interior style (modern, contemporary, traditional, industrial, minimalist, etc.)

**Use Case:** adding furnishing items to a room

Global FurnitureWeights { } #dictionary

Interface Furniture{

changeColor()

changeMaterial()

changeStyle()

editDimensions()

}

# create couch, tv, each will implement the Furniture interface differently as each piece of

# furniture is different … example of TV below

Class TV implements Furniture{

Private static FurnitureParts[ ] # list of parts ex. Screen, bezel, stand, these don’t

change

private Dimensions{ } # dictionary of dimensions of each part, ex. bezels: {height: 10in,

width: 10in, depth: 5in}

Private Colors{ } # ex. { stand: black, bezels: silver}

Private Materials{ } # ex. { stand: wood, bezels: steel}

Public void changeColor(part, newcolor){

Colors[part] = newcolor

}

Public void changeMaterial(part, newMaterial){

Material[part] = newMaterial

}

Public void changeStyle( getStyle(newStyle) ){

For part in FurnitureParts:

changeColor(part, newStyle[color] )

changeMaterial(part, newStyle[materials] )

}

Public dictionary getDimensions(){

Return Dimensions

}

# newDimensions is dictionary of new dimensions ex{height: 7in, width: 7in, depth: 2in}

Public void editDimensions( part: Furniture, newDimensions: dictionary){

Dimensions[part] = newDimensions

}

}

# here style values are hardcoded

Public Class Style{

Private final static Modern = {

AccentColors = [ black, dark brown ]

MainColors = [ white, silver, light brown ]

Materials = [ Glass, Wood, Steel ]

}

Private final static Rustic = {

AccentColors = [ white, light blue]

MainColors = [ brown, light brown, dark brown, black]

Materials = [ wood, leather, fur, stucco ]

}

Private final static Industrial = {

AccentColors = [ teal blue, light beige ]

MainColors = [ brown, red ]

Materials = [ tiles, concert, cement, brick ]

}

Public static getStyle( style ){

If style == modern:

Return Modern

Else If style == rustic:

Return Rustic

Else If style == industrial:

Return Industrial

}

}

# depending on room and style will output suggest furniture you might use

# the furniture you click on will have a positive association with the room and style and

# be weighted more heavily while the furniture that never gets clicked will have a negative

# Association and get weighted less, how recommendations get better over time

# in python if you have a matrix cast as a Dataframe and do an operation like +, -, \* or add (if # x == 1), python will intelligently apply that operation to every row in the matrix / Data Frame # … hence why I wrote the pseudo code as I did

Class RecommendationEngine{

Public void static RunEngine(){

Inputs = convertStringtoInput( getRoom() , getStyle() )

# algo’s neural engine, inputs,outputs, FurnitureWeights are all matrices

Outputs = inputs \* FurnitureWeights # matrix math

If outputs > 0

Outputdisp = 1 # display furniture

Else:

Ouptutdisp = 0 # don’t display furniture

# building Error matrices to update furniture weights

# clicks are stored for each object in each room

For itemOutput in Outputs:

If numclickonObject > 0:

Error.add( itemOutput + constant\*numclickonobejct )

Else:

Error.add( itemOutput - constant\*totalClicks )

FurnitureWeights += Error\*input

OutputdispList.add( furnitureItem if Outputdisp[furnitureItem] == 1)

Return OutputdispList

}

}

# converts the raw data to a nice UI for user

# On x,y,z will show the conditions under which UI will re-render and the Recommendation

# Engine will re-run to update what it displays to the user and train the algo so it gets better as

# user interacts with the program more

Class renderUI () {

public static void main(String [] args)

{

spinRender() # spin means re-run function continously as long as program is

alive

}

Private void spinRender(){

On roomChange:

newFurnitureList = RecommendationEngine.RunEngine()

reRender( recommendationList )

# recommednation engine updates on add and removes (learning what you

# Don’t want associated with a room

On furnitureItemAdd or furnitureItemRemove:

newFurnitureList = RecommendationEngine.RunEngine()

reRender( recommendationList )

reRender( Room )

On itemPropertyChange:

reRender( item )

On FloorPlanChange:

reRender( floorPlan)

}

}

Class Rooms( RoomName ) {

Private RoomName

Private FurnitureList[ ] # list of parts ex. Screen, bezel, stand, these don’t

change

private Dimensions{ } # dictionary of dimensions, ex. {height: 10in,

width: 10in, depth: 5in}

Private Walls[ ] # list of wall objects

Private Windows[ ] # list of wall objects

Private Floor

Public addFurniture( furniture, roomLocation ){

If ( roomLocation.check( furniture.getDimensions ) ):

FurnitureList.add( furniture )

}

Public removeFurniture( furniture ){

FurnitureList.remove( furniture )

}

Public changeRoomStyle( style ){

For furnitureItem in Room:

furnitureItem.changeStyle(style)

For wall in Walls:

wall.changeStyle(style)

}

Public string getRoom(){

Return RoomName

}

}

Class houseStructures (walls , roof,)