UDMS Library Documentation

Nik and Thn

05-10-2020

Room (functionsRoom.lua)

All the functions have as first argument the variable `room`, which denotes a LuaRoom. In the following description of functions we usually omit mention of `room`.

Also, in **all** examples the name of `room` is actually `Room`. This is good practice to follow in your code and it is true by default inside all the individual, group and settings scripts.

getRoomName(room)

Returns the name of `room`.

```
local roomName = require('functionsRoom').getRoomName(Room)
```

getScriptPath(room)

Returns the containing folder of `room`'s basic script, which must always be named `settings.lua`.

```
local scriptPath = require('functionsRoom').getScriptPath(Room)
```

getSceneName(room)

Returns the name of the unity scene the `room` is using.

```
local sceneName = require('functionsRoom').getSceneName(Room)
```

useCameraScript(room)

Use the `camera.lua` script in the scenario of `room`. The camera.lua script must exist in `room`'s path.

```
require('functionsRoom').useCameraScript(Room)
```

addCamera(room)

Alias for useCameraScript(room)

```
require('functionsRoom').addCamera(Room)
```

getGroups(room)

Returns a dictionary of the groups in `room`.

Format: `key: groupName, value: group`.

```
local groups = require('functionsRoom').getGroups(Room)
```

getGroupNames(room)

Returns a list with the names of all the groups in `room`.

```
local groupNames = require('functionsRoom').getGroupNames(Room)
```

getGroup(room, groupName)

Returns the `room`'s group from its name `groupName`.

```lua
local groupName = 'dancers'
local group = require('functionsRoom').getGroup(Room, 'dancers')
```

addEmptyGroup(room, groupName, scriptPath)

Adds to the `room` an empty (i.e., without `Members`) group with name `groupName` and asscoiated group script `scriptPath`.

`example`

addGroupMember(room, groupName, assetPath, prefab)

Adds to the group with `groupName` a Member which is the `prefab` located in the `assetPath`.

`example`

runGroupScript(room, groupName)

Runs the lua script associated with the group with `groupName` (the group is located in `room`).

`example`

addGroup(room, assetBundle, prefabName, instanceCount, groupName, scriptPath)

Creates a group with name `groupName` which contains `instanceCount` copies of `prefabName` (located in `assetBundle`) and are associated with script 'scriptPath.

`example`

getDomains(room)

Returns a dictionary of the groups and individual domains in `room` (basically all the script associated objects)

Format: `key: domainName, value: domain`

`example`

getIndividualObjectNames(room)

Returns a list of the names of individual domains (basically all the script associated objects) in `room`.

`example`

getIndividualObject(room, domainName)

Returns the game object with name `domainName`.

`example`

addIndividualObject(room, assetBundle, prefabName, objectName, scriptPath)

Creates an instance of the `assetBundle, prefabName, objectName, scriptPath)` (located in `assetBundle`), assigns to it the `objectName` and attaches and runs the script in `scriptPath`.

`example`

getObjects(room)

Returns a dictionary of the registered objects in `room`.

Format: `key: objectKey, value: gameObject`

`example`

getObject(room, objectKey)

Returns an object (located in `room`) which has `objectKey`.

`example`

getObjectKeys(room)

Returns a list of the keys of all objects located in `room`.

`example`

addRegisteredObject(room, objectKey, objectType, components, activateObject)

Adds a registered object (with `objectKey` and of `objectType`. Optionally it addds components from the list `components` and activates the object iff `activateObject=true`.

Possible object types: `camera, cube, cylinder, light, plane, quad, sphere, vcamera`

`example`

addObject(room, objectType,components,activateObject)

Adds an (unregistered) object (of `objectType`. Optionally it addds components from the list `components` and activates the object iff `activateObject=true`.

Possible object types: `camera, cube, cylinder, light, plane, quad, sphere, vcamera`

`example`

registerObject(room, objectKey, object)

Registers `object` with `objectKey` to the `room`.

`example`

Group (functionsGRP.lua)

The 'LFG' variable in all the examples below is assumed to be defined like, where the 'Group' variable contains a LuaGroup.

The 'Group' variable exists by default inside each group script. It can also be set on any script using `room`'s getGroup function.

Also, all the group member ids that appear are assumed to be in range.

```
local LFG = require('functionsGRP')(Group)
```

Miscellaneous functions

getMembers()

Returns a list of the group's members.

```
local members = LFG.getMembers()
```

dirToAgent(i, j)

Returns the direction (Vector3) of the agent with id i to the agent with id j.

```
local dir_0_to_2 = LFG.dirToAgent(0, 2)
```

groupFuns.grpGetRenderers(i)

Returns a list of renderers of Members[i]

groupFuns.grpSetFormation(frm)

Positions the `group` members to the positions described by formation `frm`.

groupFuns.grpSetNeighbors(nbr)

Sets the neighborhodd of all `group` members according to neighbourhood `nbr`.

groupFuns.grpSetStates(agents)

Sets to 1 the state of agents contained in list `agents`.

Formations functions

groupFuns.frmMakeFormation(formation, N, ...)

Returns an `N`-point formation of type `formation. The`...‘ allow to pass parameters of the specific formation.

Possible formation types and their parameters:

`circle; center, radius, angleOffset`

`ellipse; a,b,angleOffset`

`line; start, step`

`grid; columns; topLeftPoint; rowDistance; colDistance`

`lissajous; ax, wx, az, wz, angleOffset`

`nrose; center, a, , angleOffset`

groupFuns.frmKrosePoints(N,center,a,K,angleOffset)

Returns an `N`-point K-rosette (polar equation `a*math.cos(K*angle)`).

groupFuns.frmLissajousPoints(N,ax,wx,az,wz,angleOffset)

Returns an `N`-point Lissajous curve (parametric equation `(ax * math.cos(anglex), az * math.sin(anglez))`).

groupFuns.frmGridPoints(N, columns, topLeftPoint, rowDistance, colDistance)

Returns an `N`-point grid with `columns` columns.

groupFuns.frmLinePoints(N,start,step)

Returns an `N`-point line which starts at point `start` and has vector step `step`.

groupFuns.frmEllipsePoints(N, center, a, b, angleOffset)

Returns an `N`-point ellipse with `center` and parameters `a,b`.

groupFuns.frmCirclePoints(N, center, radius, angleOffset)

Returns an `N`-point ellipse with `center` and `radius`.

Generations Cellular Automata functions

groupFuns.gcaDefine(BirthConds,SurvConds,NrStates)

```
    return CS.LuaScripting.GCA(BirthConds,SurvConds,NrStates)
end
```

groupFuns.gcaUpdate(group,gca,typ)

```
    if typ=="type1" then
        group:GCAUpdate(gca)
    elseif typ=="type2" then
        group:AdaptiveStateUpdate(gca)
    else
    end
end
```

groupFuns.gcaMakeNbhd(nbhdType, N, . . .)

```
    if nbhdType=="rel1" then
        local relArray=select(1,...)
        local wrap=select(2,...)
        return groupFuns.gcaRelativeNeighbours(N, relArray, wrap)
    elseif nbhdType=="rel2" then
        local n1=select(1,...)
        local relArray=select(2,...)
        local wrap=select(3,...)
        return groupFuns.gcaRelativeGridNeighbours(N, n1, relArray, wrap)
    elseif nbhdType=="filePath" then
        local fpath=select(1,...)
        return groupFuns.gcaFilePathNeighbours(N,fpath)
    elseif nbhdType=="distBased" then
    end
end
```

groupFuns.gcaFilePathNeighbours(N, fpath)

```
    local file = io.open(fpath, 'r')
    --print(fpath, file)
    if file ~= nil then
        local nbrs = {}
        local counter = 1
        for line in file:lines() do
            nbrs[counter] = {}
            local nbrCounter = 1
            for nbr in line:gmatch("%d+") do
               nbrs[counter][nbrCounter] = tonumber(nbr)
               nbrCounter = nbrCounter + 1
            end

            if counter == N then break end
            counter = counter + 1
        end
        file:close()
        return nbrs
    else
        error('file not found')
    end
end
```

groupFuns.gcaRelativeGridNeighbours(N, NC, relArray2, wrap)

```
    local nbrs = {}
    local NR = math.ceil(N/NC)
    local K = #relArray2
    for n = 0, N - 1 do
        local nc = n % NC
        local nr = math.floor(n / NC)
        nbrs[n+1] = {}
        local nbCounter = 1
        for k = 1, K do
            local ax = nc + relArray2[k][1]
            local ay = nr + relArray2[k][2]

            if wrap then
                ax = ax % NC
                ay = ay % NR
                local m = ay * NC + ax
                if m < N then
                    nbrs[n+1][nbCounter] = m
                    nbCounter = nbCounter + 1
                end
            else
                local m = ay * NC + ax
                if ax > -1 and ax < NC and ay > -1 and ay < NR and m < N then
                    nbrs[n+1][nbCounter] = m
                    nbCounter = nbCounter + 1
                end
            end
        end
    end
    return nbrs
end
```

groupFuns.gcaRelativeNeighbours(n, relArray, wrap)

```
    if wrap == nil then
        wrap = true
    end
    local nbrs = {}
    if wrap then
        for i = 1, n do
            nbrs[i] = {}
            for j = 1, #relArray do
                nbrs[i][j] = (i - 1 + relArray[j] + n) % n
            end
        end
    else
        for i = 1, n do
            local nbsCounter = 1
            nbrs[i] = {}
            for j = 1, #relArray do
                local nbId = i - 1 + relArray[j]
                if nbId >= 0 and nbId < n then
                    nbrs[i][nbsCounter] = nbId
                    nbsCounter = nbsCounter + 1
                end
            end
        end
    end
    return nbrs
end
```

Animation functions

groupFuns.setAnim(i,anim,transDur)

```
    anims[i]:CrossFade(anim,transDur)
end
```

groupFuns.aniCrossFade(i,anim,transDur,rel)

```
    if not rel then anims[i]:CrossFadeInFixedTime(anim,transDur)
    else anims[i]:CrossFade(anim,transDur) end
end
```

groupFuns.aniCrossFadeDiff(i,anim,transDur,rel)

```
    local currentAnim = groupFuns.aniGetClipName(i)
    if currentAnim ~= "" and currentAnim ~= anim then
        groupFuns.aniCrossFade(i,anim,transDur,rel)
    end
end
```

groupFuns.aniGetAnimator(i)

```
    return anims[i]
end
```

groupFuns.aniGetClipLength(i)

```
    local currentClipInfo = anims[i]:GetCurrentAnimatorClipInfo(0)
    return currentClipInfo[0].clip.length
end
```

groupFuns.aniGetClipName(i)

```
    local currentClipInfo = anims[i]:GetCurrentAnimatorClipInfo(0)
    --return currentClipInfo[0].clip.name
    --
    if currentClipInfo.Length>0 then
        return currentClipInfo[0].clip.name
    else
        return ""
    end
    --]]
end
```

groupFuns.aniGetClipSpeed(i)

```
    return anims[i].speed
end
```

groupFuns.aniGetClipSpeedMultiplier(i)

```
    return anims[i].speedMultiplier
end
```

groupFuns.aniGetClipTime(i,typ)

```
    -- when typ="rel" returns the % (0-1) of progress in current loop
    local timnorm = anims[i]:GetCurrentAnimatorStateInfo(0).normalizedTime
    local timint,timfrac = math.modf(timnorm)
    local length=groupFuns.aniGetClipLength(i)
    if typ=="rel" then return timfrac
    else return timfrac*length end
end
```

groupFuns.aniGetSpeed(i)

```
    return anims[i]:GetCurrentAnimatorStateInfo(0).speed
end
```

groupFuns.aniSetClipSpeed(i,v)

```
    anims[i].speed=v
end
```

groupFuns.aniSetClipSpeedMultiplier(i,v)

```
    anims[i].speedMultiplier=v
end
```

groupFuns.aniSetRootMotion(i,rootMotion)

```
    anims[i].applyRootMotion=rootMotion
end
```

groupFuns.aniSetStabFeet(i,stabFeet)

```
    anims[i].stabilizeFeet=stabFeet
end
```

Inverse Kinematics functions

groupFuns.ikSetLookAtObject(i,go)

```
    anims[i]:ikSetLookAtPosition(go.transform.position);
end
```

groupFuns.ikSetLookAtAgent(i,j)

```
    anims[i]:SetLookAtPosition(group.Members[j].transform.position);
end
```

groupFuns.ikSetLookAtPnt(i,pnt)

```
    anims[i]:SetLookAtPosition(pnt);
end
```

groupFuns.ikSetLookAtWeight(i,ikWeight)

```
    anims[i]:SetLookAtWeight(ikWeight);
end
```

groupFuns.ikSetPosObject(i,ikGoal,ikTarget)

```
    if ikGoal=="LH" then anims[i]:SetIKPosition(UE.AvatarIKGoal.LeftHand,ikTarget.transform.position)
    elseif ikGoal=="RH" then anims[i]:SetIKPosition(UE.AvatarIKGoal.RightHand,ikTarget.transform.position)
    elseif ikGoal=="LF" then anims[i]:SetIKPosition(UE.AvatarIKGoal.LeftFoot,ikTarget.transform.position)
    elseif ikGoal=="RF" then anims[i]:SetIKPosition(UE.AvatarIKGoal.RightFoot,ikTarget.transform.position)
    end
end
```

groupFuns.ikSetPosVec(i,ikGoal,Pnt)

```
    if ikGoal=="BD" then anims[i].bodyPosition=ikTarget
    elseif ikGoal=="LH" then anims[i]:SetIKPosition(AvatarIKGoal.LeftHand,ikTarget)
    elseif ikGoal=="RH" then anims[i]:SetIKPosition(AvatarIKGoal.RightHand,ikTarget)
    elseif ikGoal=="LF" then anims[i]:SetIKPosition(AvatarIKGoal.LeftFoot,ikTarget)
    elseif ikGoal=="RF" then anims[i]:SetIKPosition(AvatarIKGoal.RightFoot,ikTarget)
    end
end
```

groupFuns.ikSetPosWeight(i,ikGoal,ikWeight)

```
    if ikGoal=="LH" then anims[i]:SetIKPositionWeight(AvatarIKGoal.LeftHand,ikWeight)
    elseif ikGoal=="RH" then anims[i]:SetIKPositionWeight(AvatarIKGoal.RightHand,ikWeight)
    elseif ikGoal=="LF" then anims[i]:SetIKPositionWeight(AvatarIKGoal.LeftFoot,ikWeight)
    elseif ikGoal=="RF" then anims[i]:SetIKPositionWeight(AvatarIKGoal.RightFoot,ikWeight)
    end
end
```

groupFuns.ikSetRot(i,ikGoal,ikTarget)

```
    if ikGoal=="LH" then anims[i]:SetIKRotation(AvatarIKGoal.LeftHand,ikTarget.transform.rotation)
    elseif ikGoal=="RH" then anims[i]:SetIKRotation(AvatarIKGoal.RightHand,ikTarget.transform.rotation)
    elseif ikGoal=="LF" then anims[i]:SetIKRotation(AvatarIKGoal.LeftFoot,ikTarget.transform.rotation)
    elseif ikGoal=="RF" then anims[i]:SetIKRotation(AvatarIKGoal.RightFoot,ikTarget.transform.rotation)
    end
end
```

groupFuns.ikSetRotWeight(i,ikGoal,ikWeight)

```
--Sets the ikGoal limb (options: "LH","RH","LF","RF") target rotation weight to be ikWeight.
    if ikGoal=="LH" then anims[i]:SetIKRotationWeight(AvatarIKGoal.LeftHand,ikWeight)
    elseif ikGoal=="RH" then anims[i]:SetIKRotationWeight(AvatarIKGoal.RightHand,ikWeight)
    elseif ikGoal=="LF" then anims[i]:SetIKRotationWeight(AvatarIKGoal.LeftFoot,ikWeight)
    elseif ikGoal=="RF" then anims[i]:SetIKRotationWeight(AvatarIKGoal.RightFoot,ikWeight)
    end
end
```

Trail Renderer functions

```
-- obsolete, do not use ...
```

groupFuns.attachTrail(i, color, aliveTime, width)

```
    if trailRenderers[i] == nil then
        trailRenderers[i] = UT.attachTrailRenderer(group.Members[i].gameObject)
    end
    trailRenderers[i].startColor = color
    trailRenderers[i].endColor = color
    trailRenderers[i].time = aliveTime
    trailRenderers[i].startWidth = width
    trailRenderers[i].endWidth = width
    return trailRenderers[i]
end
```

groupFuns.trailAttach(i,offset,color,tim,width)

```
    local go=UE.GameObject("Trail")
    go.transform.parent=group.Members[i].transform
    go.transform.position=group.Members[i].transform.position+offset
    trail = UT.attachTrailRenderer(go)
    trail.startColor = color
    trail.endColor = color
    trail.time = tim
    trail.startWidth = width
    trail.endWidth = width
    return trailRenderers[i]
end
```

groupFuns.trailGetEndColor(i)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    return trail.endColor
end
```

groupFuns.trailGetEndWidth(i)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    return trail.endWidth
end
```

groupFuns.trailGetStartColor(i)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    return trail.startColor
end
```

groupFuns.trailGetStartWidth(i)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    return trail.startWidth
end
```

groupFuns.trailGetTime(i)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    return trail.time
end
```

groupFuns.trailGetTrail(i)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    return trail
end
```

groupFuns.trailSetEndColor(i,color)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    --if not trail==nil then trail.endColor=color end
    trail.endColor=color
end
```

groupFuns.trailSetEndWidth(i,width)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    --if not trail==nil then trail.endWidth=width end
    trail.endWidth=width
end
```

groupFuns.trailSetStartColor(i,color)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    --if not trail==nil then trail.startColor=color end
    trail.startColor=color
end
```

groupFuns.trailSetStartWidth(i,width)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    --if not trail==nil then trail.startWidth=width end
    trail.startWidth=width
end
```

groupFuns.trailSetTime(i,tim)

```
    local trail=group.Members[i]:GetComponentInChildren(typeof(UE.TrailRenderer))
    --if not trail==nil then trail.time=tim end
    trail.time=tim
end
```

Group and Group Member functions

groupFuns.addMember(name,folder)

```
    return group:AddMember(name,folder)
end
```

groupFuns.dirAgentToPnt(i,point)

```
    return group.Members[i]:DirAgentToPnt(point)
end
```

groupFuns.dirAttractRepel(i,j,dist)

```
    return group.Members[i]:DirAttractRepel(j,dist)
end
```

groupFuns.dirAvoidAgent(i,j)

```
    return group.Members[i]:DirAvoidAgent(j)
end
```

groupFuns.dirAvoidNearestAgent(i)

```
    return group.Members[i]:DirAvoidDirAvoidNearestAgentAgent()
end
```

groupFuns.dirMine(i)

```
    return group.Members[i]:DirMine()
end
```

groupFuns.dirOfAgent(i)

```
    return group:DirOfAgent(i)
end
```

groupFuns.dirOfNearest(i)

```
    return group.Members[i]:DirOfNearest()
end
```

groupFuns.dirStayInDisc(i,radius)

```
    return group.Members[i]:DirStayInDisc(radius)
end
```

groupFuns.dirToAgent(i,j)

```
    return group.Members[i]:DirToAgent(j)
end
```

groupFuns.dirToHood(i,radius)

```
    return group.Members[i]:DirToHood(radius)
end
```

groupFuns.dirToNearest(i)

```
    return group.Members[i]:DirToNearest()
end
```

groupFuns.dirToNearestActive(i)

```
    return group.Members[i]:DirToNearestActive()
end
```

```
--?????????????????????????????????????????????
```

groupFuns.dirToNearestWithState(i,s)

```
    return group.Members[i]:DirToNearestActive()
end
```

groupFuns.distAgentToPnt(i,point)

```
    return group.Members[i]:DistAgentToPnt(point)
end
```

groupFuns.distOfAgents(i,j)

```
    return group:DistOfAgents(i,j)
end
```

groupFuns.distToAgent(i,j)

```
    return group.Members[i]:DistToAgent(j)
end
```

groupFuns.distToHood(i,radius)

```
    return group.Members[i]:DistToHood(radius)
end
```

groupFuns.distToNearest(i)

```
    return group.Members[i]:DistToNearest()
end
```

groupFuns.distToNearestActive(i)

```
    return group.Members[i]:DistToNearestActive()
end
```

groupFuns.distTravelled(i)

```
    return group.Members[i]:DistTravelled()
end
```

groupFuns.doScript()

```
    return group:DoScript()
end
```

```
--?????????????????????????????????????????
```

groupFuns.getAgentsInState(s)

```
end
```

groupFuns.getAllComponents(i)

```
    return  group.Members[i]:GetComponents(typeof(UE.Component))
end
```

groupFuns.getAgentNearest(i)

```
    return group.Members[i]:GetAgentNearest()
end
```

groupFuns.getColor(i,j)

```
    if j==nil then j=0 end
    return renderers[i][j].material.color
end
```

groupFuns.getDisplacement(i)

```
    return group.Members[i]:Displacement()
end
```

groupFuns.getDist()

```
    return group.Dist
end
```

groupFuns.getDomainName()

```
    return group.DomainName
end
```

groupFuns.getEulerAngles(i)

```
    return group.Members[i].transform.eulerAngles
end
```

groupFuns.getEulerAnglesOld(i)

```
    return group.Members[i].EulerAnglesOld
end
```

groupFuns.getGroupCenter()

```
    return group:GetGroupCenter()
end
```

groupFuns.getMemberIdsInCircle(center,radius)

```
    return group:GetMemberIdsInCircle(center, radius)
end
```

groupFuns.getMembers()

```
    return group.Members
end
```

groupFuns.getNearestAgentWithState(position,state)

```
    return group:GetNearestAgentWithState(position,state)
end
```

groupFuns.getNearestActive(i)

```
    return group.Members[i]:GetNearestActive()
end
```

groupFuns.getNeighbours(i)

```
    return group.Members[i].Neighbours
end
```

```
--??????????????????????????????????????????
```

groupFuns.getNumAgentsInState(s)

```
end
```

groupFuns.getPosition(i)

```
    return group.Members[i]:GetPos()
end
```

groupFuns.getPos(i)

```
    return group.Members[i].transform.position
end
```

groupFuns.getPosOld(i)

```
    return group.Members[i].PositionOld
end
```

groupFuns.getRot(i)

```
    return group.Members[i]:GetAngles()
end
```

groupFuns.getState(i)

```
    return group.Members[i].State
end
```

groupFuns.getStateOld(i)

```
    return group.Members[i].StateOld
end
```

groupFuns.getTurnToMoveDir(i)

```
    return group.Members[i].TurnToMoveDir
end
```

groupFuns.goToAgent(i,j,dist)

```
    group.Members[i]:GoToAgent(i,j,dist)
end
```

groupFuns.goToCenter(i,dist)

```
    group.Members[i]:GoToCentert(dist)
end
```

groupFuns.goToPoint(i,point,dist)

```
    group.Members[i]:GoToPoint(point,dist)
end
```

groupFuns.highlightNeigbours(i)

```
    return group:HighlightNeigbours(i)
end
```

groupFuns.isActive(i)

```
    return group.Members[i].IsActive
end
```

groupFuns.moveFwd(i,dist)

```
    group.Members[i]:MoveFwd(dist)
end
```

groupFuns.moveInDir(i,dir,dist,normalized)

```
    group.Members[i]:MoveInDir(dir,dist,normalized)
end
```

groupFuns.moveRight(i,dist)

```
    group.Members[i]:MoveRight(dist)
end
```

groupFuns.moveUp(i,dist)

```
    group.Members[i]:MoveUp(dist)
end
```

groupFuns.registerGridNeighbours(N1)

```
    return group:RegisterGridNeighbours(N1)
end
```

groupFuns.setColor(i,color,j)

```
    if j==nil then j=0 end
    renderers[i][j].material.color=color
end
```

groupFuns.setColorState(i,bool)

```
    group.Members[i].ColorState = bool
end
```

groupFuns.setDir(i,dir)

```
    group.Members[i]:SetDir(dir)
end
```

groupFuns.setNeighbours(i,nbrs)

```
    group.Members[i]:SetNeighbours(nbrs)
end
```

groupFuns.setPos(i,pos)

```
    group.Members[i].transform.position = pos
end
```

groupFuns.setPosX(i,xpos)

```
    group.Members[i]:SetPosX(xpos)
end
```

groupFuns.setPosY(i,ypos)

```
    group.Members[i]:SetPosY(ypos)
end
```

groupFuns.setPosZ(i,zpos)

```
    group.Members[i]:SetPosZ(zpos)
end
```

groupFuns.setRot(i,rot)

```
    group.Members[i].transform.eulerAngles =rot
end
```

groupFuns.setRotX(i,xrot)

```
    group.Members[i]:SetRotX(xrot)
end
```

groupFuns.setRotY(i,yrot)

```
    group.Members[i]:SetRotY(yrot)
end
```

groupFuns.setRotZ(i,zrot)

```
    group.Members[i]:SetRotZ(zrot)
end
```

groupFuns.setScale(i,scale)

```
    group.Members[i]:SetScale(scale)
end
```

groupFuns.setState(i,s)

```
    group.Members[i].State=s
end
```

groupFuns.setTurnToMoveDir(i,turn)

```
    group.Members[i].TurnToMoveDir=turn
end
```

groupFuns.toGridFormation(N1,center,rowDist,colDist)

```
    local N2=math.ceil(Members.Count/N1)
    local topLeftPoint=center+UE.Vector3(N1*rowDist/2,0,N2*colDist/2)
    return group:ToGridFormation(N1,topLeftPoint,rowDist,colDist)
end
```

groupFuns.toggleIndices(toggle)

```
    return group:ToggleIndices(toggle)
end

--?????????????????????????????????????????
```

groupFuns.turnByDir(i,dir,wr)

```
-- Rotates with speed wr the calling Agent towards the direction vector Dir1.
    local ang0 = group.Members[i].transform.eulerAngles.y
    local dir0 = UE.Vector3(math.cos(ang0),0,math.sin(ang0))
    local dir1 = dir/math.sqrt(dir.x^2+dir.z^2)
    local dir2 = dir0+dir1
    group.Members[i]:TurnToDirSoft(dir2,wr)
end
```

groupFuns.turnFwd(i,a)

```
-- Rotates calling Agent a degrees around z axis.
    --group.Members[i].transform:Rotate(a*UE.Vector3.forward)
    group.Members[i].transform:Rotate(a*UE.Vector3(1,0,0))
end

--?????????????????????????????????????????
```

groupFuns.turnRght(i,a)

```
-- Rotates calling Agent a degrees around x axis.
    --group.Members[i].transform:Rotate(a*UE.Vector3.right)
    group.Members[i].transform:Rotate(a*UE.Vector3(0,1,0))

end

--?????????????????????????????????????????
```

groupFuns.turnToAgent(i,j,wr)

```
-- Rotates the calling Agent to the direction of Agents[n1] with angular speed wr (per update).
    local point=group.Members[j].transform.position
    local meRot=group.Members[i].transform.rotation
    local trgRot = UE.Quaternion.LookRotation(point);
    group.Members[i].transform.rotation = UE.Quaternion.Slerp(meRot,trgRot,wr*UE.Time.deltaTime);
end
```

groupFuns.turnToAngle(i,targetAngle,degrees)

```
    group.Members[i]:TurnToAngle(targetAngle,degrees)
end
```

```
--?????????????????????????????????????????
```

groupFuns.turnToAngle1(i,targAng,dAng)

```
-- Rotates calling Agent by dAng (in degrees) towards making his y Euler angle targAng.
    if(group.Members[i].transform.eulerAngles.y>targAng) then group.Members[i].transform:Rotate(-dAng*UE.Vect
    if(group.Members[i].transform.eulerAngles.y<targAng) then group.Members[i].transform:Rotate( dAng*UE.Vect
end
```

groupFuns.turnToDir(i,dir,speed)

```
    group.Members[i]:TurnToDir(dir,speed)
end
```

groupFuns.turnToDirSoft(i,dir,speed)

```
    group.Members[i]:TurnToDirSoft(dir,speed)
end

--?????????????????????????????????????????
```

groupFuns.turnToMoveDir(i,wr)

```
-- Rotates calling Agent to the Agent's move direction. Turn angular speed is wr (per update)
    local meRot=group.Members[i].transform.rotation
    local d=group.Members[i]:Displacement()
    local a=math.atan(d.x,d.z)*180/3.14159;
    local trgRot = UE.Quaternion.Euler(0,a,0);
    group.Members[i].transform.rotation = UE.Quaternion.Slerp(meRot,trgRot,wr*UE.Time.deltaTime);
end

--?????????????????????????????????????????
```

groupFuns.turnToPnt(i,point,wr)

```
-- Rotates calling Agent towards point Pnt. Turn angular speed is wr (per update).
    local meRot=group.Members[i].transform.rotation
    local trgRot = UE.Quaternion.LookRotation(point);
    group.Members[i].transform.rotation = UE.Quaternion.Slerp(meRot,trgRot,wr*UE.Time.deltaTime);
end

--?????????????????????????????????????????
```

groupFuns.turnUp(i,a)

```
-- Rotates calling Agent a degrees around y axis.
    --group.Members[i].transform:Rotate(a*UE.Vector3.up)
    group.Members[i].transform:Rotate(a*UE.Vector3(0,0,1))
end
```

groupFuns.updateStates(rule,. . .)

```
end
```

Navigation functions

groupFuns.navAddSurface(ground)

```
    return UT.navAddSurface(ground)
end
```

groupFuns.navBakeSurface(surface)

```
    UT.navBuildSurface(surface)
end
```

groupFuns.navToAgent(i,j)

```
    navs[i].destination = group.Members[j].transform.position
end
```

groupFuns.navToPoint(i,point)

```
    navs[i].destination = point
end
```

groupFuns.navAttachAgent(i)

```
    if navs[i] == nil then
        navs[i] = group.Members[i].gameObject:AddComponent(typeof(UE.AI.NavMeshAgent))
    end
    return navs[i]
end
```

groupFuns.navActive(i, status)

```
    navs[i].isStopped = not status
end
```

groupFuns.navGetDestination(i)

```
    return navs[i].destination
end
```

groupFuns.navGetVelocity(i)

```
    return navs[i].velocity
end
```

groupFuns.navSetDestination(i, point)

```
    navs[i].destination = point
end
```

groupFuns.navSetSpeed(i, speed)

```
    navs[i].speed = speed
end
```

Objects (functionsOBJ.lua)

setParent(go, parentGo)

Sets the go as a child of parentGo.

```
-- Assuming go and parentGo are gameObjects or MonoBehaviours
require(functionsOBJ').setParent(go, parentGo)
```

TODO: etc

## Lights

lgtGetColor(go)

Sets the color of the light component of the go.

```lua
-- Assuming go is a gameObject/MonoBehaviour on a gameObject with a light component
require(functionsOBJ').lgtGetColor(go)
```

TODO: etc

Trails

trailGetTime(go)

Returns the alive time of the trail particles.

```lua
-- Assuming go is a gameObject/MonoBehaviour on a gameObject with a trail component IN CHILDREN
local time = require(functionsOBJ').trailGetTime(go)
```

TODO: etc

Camera (functionsCAM.lua)

stateUpdate(TIME, state)

Updates the camera based on the TIME and state parameters. If the state isn't given, it uses the camera's current state.

```lua
-- Updates the camera using the update3 function.
require('functionsCAM')(self).stateUpdate(TIME, 3)
```

TODO: etc

Effects (effects.lua)

setProperty(effectProperty, value)

Sets the effectProperty to the value specified.

```lua
require('effects').setProperty(lutEffect.Saturation, 0.5)
```

TODO: etc

Utilities (utils.lua)

TODO: etc

Logic (logic.lua)

Animations List (animations.lua)

Look Up Textures List (luts.lua)

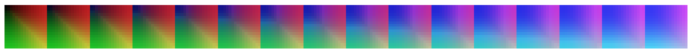

Figure 1: Action3D16