LIFT SCRIPT

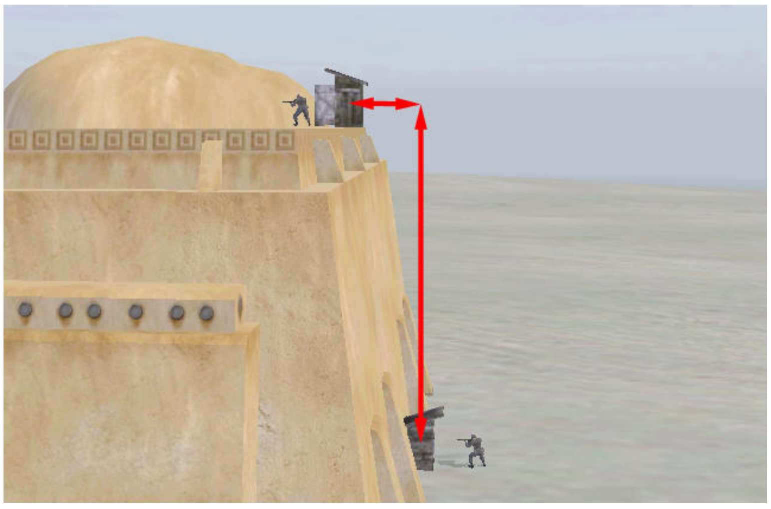
Welcome

If you are interested in how the lift from *TatooineCityCheck* works and want to use it in your mission, here you'll find information how it works.

1. Theory

I assume you have already have seen lift movement. It's not a move, actually. It's a teleportation in a little intervals for a short distance, so it would appear as a smooth move. Player is teleported as well.

Plan:

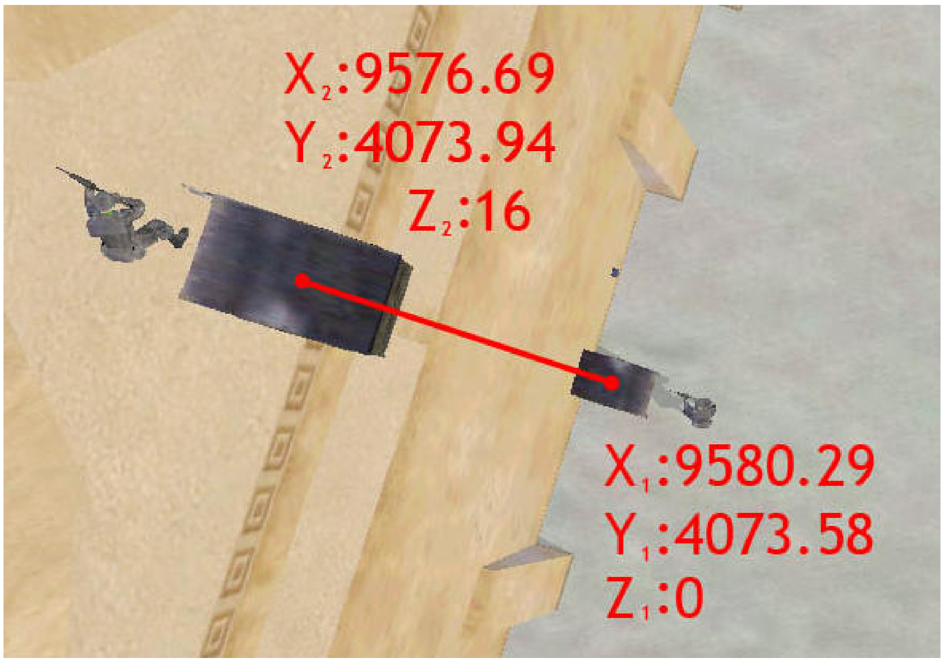


It moves: vertically (turning 180 degrees) and then horizontally or vice versa.

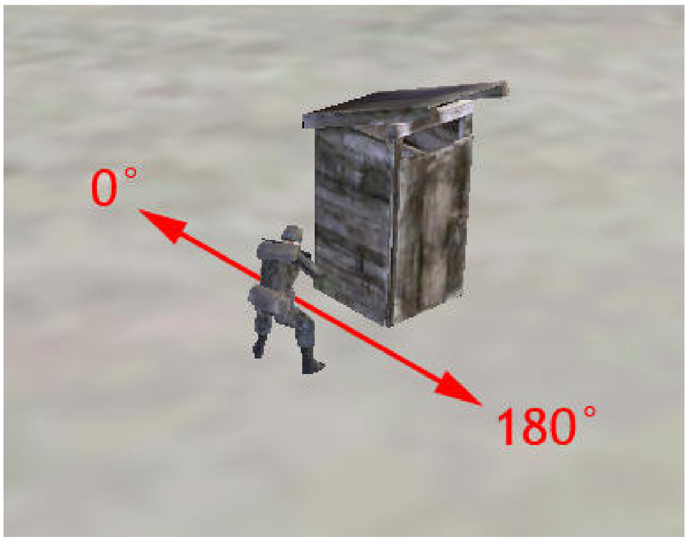
Movement will be performed by server while client only gives a signal to start.

2. Practice – Introduction

For your own lift you will need start and destination coordinates. Here are values for my lift:

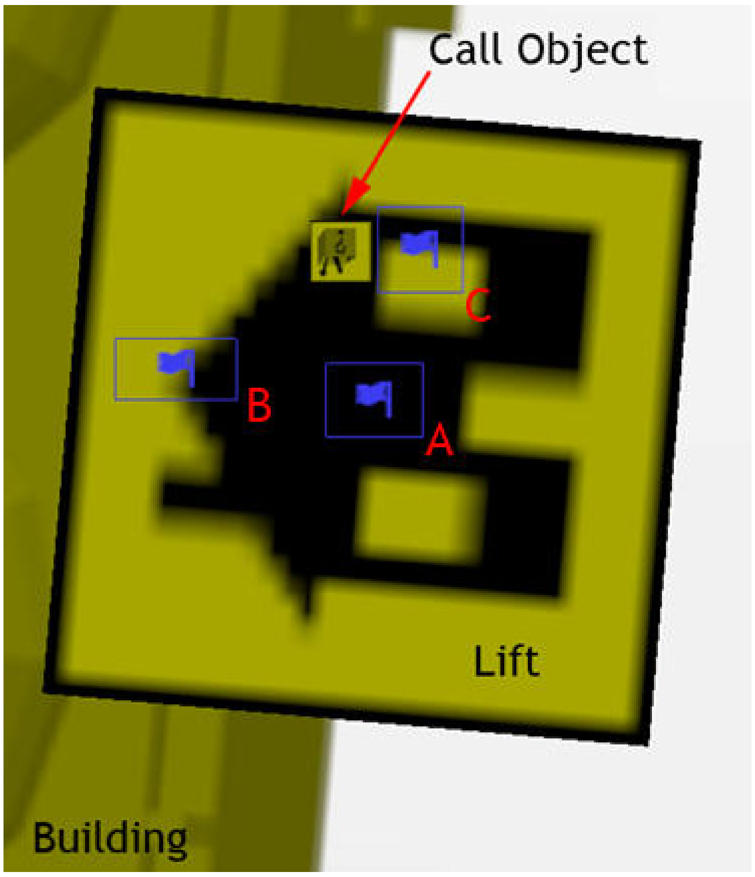


As you can see, on the picture below, lift door is on it's „back".



The building in my mission has azimuth 275° so the lift's will be 95°(when down) or 275° (when up). Player's direction will be [lift azimuth]-180 (he will face the exit).

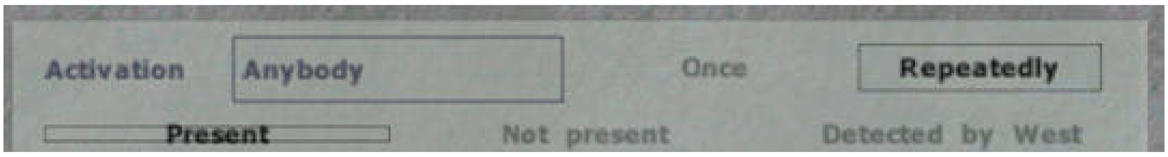
3. Practice – Mission Editor:



We have three triggers providing action to use the lift. Triggers A and B are placed to work only when someone is inside the lift.

In this mission call object's name is `telefon` and lift's - `lift`.

All triggers are set to:



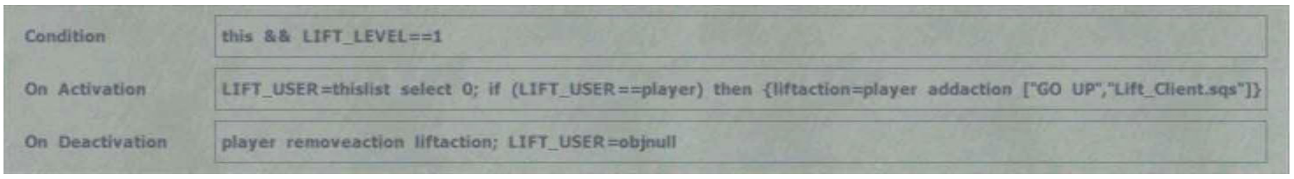
Activation	Anybody	Once	Repeatedly
Present	Not present	Detected by West	

Global variables are used in this part:

`LIFT_LEVEL` – indicates current lift position (1 – down, 2 – up). This value is less than zero when lift is moving.

`LIFT_USER` – indicates unit inside the lift.

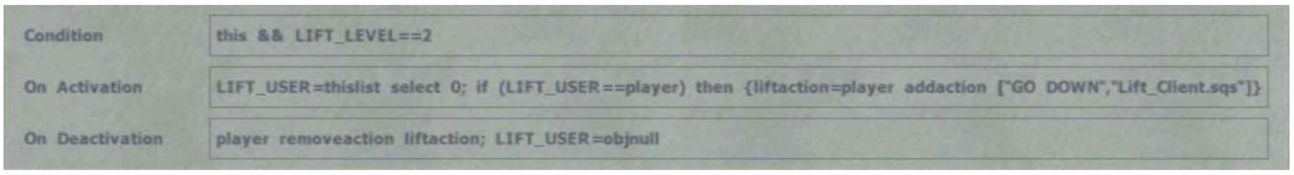
Trigger A:



Condition	this && LIFT_LEVEL==1
On Activation	LIFT_USER=thislist select 0; if (LIFT_USER==player) then {liftaction=player addaction ["GO UP","Lift_Client.sqs"]}
On Deactivation	player removeaction liftaction; LIFT_USER=objnull

Condition:	when the lift is down and anybody is inside
Activation:	if first unit inside is player then add action
Deactivation:	remove action and reset variable

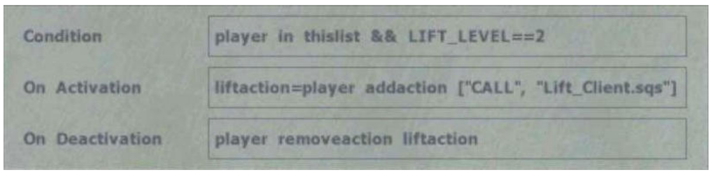
Trigger B:



Condition	this && LIFT_LEVEL==2
On Activation	LIFT_USER=thislist select 0; if (LIFT_USER==player) then {liftaction=player addaction ["GO DOWN","Lift_Client.sqs"]}
On Deactivation	player removeaction liftaction; LIFT_USER=objnull

Condition:	when the lift is up and anybody is inside
Activation:	if first unit inside is player then add action
Deactivation:	remove action and reset variable

Trigger C:



Condition	player in thislist && LIFT_LEVEL==2
On Activation	liftaction=player addaction ["CALL", "Lift_Client.sqs"]
On Deactivation	player removeaction liftaction

Condition:	when the lift is up and player is nearby
Activation:	add action
Deactivation:	remove action

Call object is dummy but in Trigger C condition field you can add:

```
&& alive telefon
```

So when it’s destroyed, trigger won’t work.

To use the lift with AI remove code checking if unit is a player. Now you may use radio command to execute the action.
Moreover the action will appear in your menu if you are close to the unit.

Place Game Logic and name it *server*.

4. Practice – Initialization

```
LIFT_LEVEL=1                                  //reset global variables
LIFT_USER=objnull
? local server : [] exec "Lift_Server.sqs"    //server executes script
```

5. Practice – Client-Side

```
player removeaction liftaction
LIFT_LEVEL=LIFT_LEVEL*-1              //change to negative value
publicvariable "LIFT_LEVEL"           //signal server

? LIFT_USER!=player : exit            //code below works only if unit is inside

#Loop                                 //loop fixing unit's direction while lift is moving
player setdir (getdir lift-190)
~0.001
? LIFT_LEVEL>0 : exit
goto "Loop"
```

Direction fixing has to be done on client’s computer. Otherwise player’s screen would warp.

6. Practice – Server-Side

Code plan:

Define variables

```
_pX=958029
_pY=407358
_pZ=0


~0.001
_fixX=getpos lift select 0; _fixY=getpos lift select 1
~0.001
lift setpos [9580.29,4073.58,0]
_fixX=(getpos lift select 0)-_fixX; _fixY=(getpos lift select 1)-_fixY
lift setpos [_pX/100-_fixX,_pY/100-_fixY,_pZ/10]
```

Wait for signal

```
#Wait
@LIFT_LEVEL<0
_user=LIFT_USER
_reverse=1
_move="V"
? LIFT_LEVEL==-2 : _reverse=-1; _move="H"
```

Move loop

```
#Movement
~0.01
? _move=="V" : _pZ=_pZ+(2*_reverse); lift setdir (getdir lift-(2.25*_reverse))
? _move=="H" : _pX=_pX-(10*_reverse); _py=_py+(1*_reverse)

{_x setpos [_pX/100-_fixX, _pY/100-_fixY, _pZ/10]} foreach [lift,_user]
_fixX2=(getpos lift select 0)-(_pX/100); _fixY2=(getpos lift select 1)-(_pY/100)
{_x setpos [_pX/100-_fixX-_fixX2, _pY/100-_fixY-_fixY2, _pZ/10]} foreach [lift,_user]

? _move=="V" && _pZ>=160 : _move="H"; goto "Movement"
? _move=="H" && _py<=407358 : _move="V"; goto "Movement"

? _move=="V" && _pZ<=0 : LIFT_LEVEL=1; publicvariable "LIFT_LEVEL"; goto "Wait"
? _move=="H" && _py>=407394 : LIFT_LEVEL=2; publicvariable "LIFT_LEVEL"; goto "Wait"
goto "Movement"
```

a) Variables

`_pX, _pY, _pZ` //lift position

Assign them start coordinates multiplied by 100 (X, Y axis) and by 10 (Z axis).

In OFP operations on fractions are inacurate if being looped.

That's why I've converted coordinates to natural numbers. Later, when placing object, they will be divided back to fractions.

Following code finds values to fix the lift position.

In OFP real object position is not equal to numbers returned by `getPos` command. Comparing `getPos` value to number is inaccurate.
If you place object by `setPos` it won't be placed in the exact position you've typed.

`_fixX` and `_fixY` indicates the difference between real position and typed coordinates.

b) Wait for activation

Server-side script is looped checking if the lift has been activated (`LIFT_LEVEL<0`). If so, then it resets local variables:

`_user`	//unit inside the lift
`_reverse`	//movement modifier (1 – lift is going up, -1 – going down)
`_move`	//indicates move type („V" – vertical, „H" – horizontal)

Variables are set for ascending by default but if the lift is moving down the values get switched.

c) Move

```
? _move=="V" : _pZ=_pZ+(2*_reverse); lift setdir (getdir lift-(2.25*_reverse))
? _move=="H" : _pX=_pX-(10*_reverse); _pY=_pY+(1*_reverse)
```

Change position depending on type (`_move`). Moving down is a reversed move up (rate is multiplied by -1). Movement rates are the numbers in brackets.

I haven't converted turning rate to natural number because it's less important than object position.

Once you set one of the movement rates, you can calcutate the other one using simultaneous equations:

$$\begin{cases} X_1 + LR_x = X_2 \\ Y_1 + LR_y = Y_2 \end{cases}$$

X_1, Y_1 - initial lift coordiantes

X_2, Y_2 - lift destination

R_X – movement rate in X axis

R_Y – movement rate in Y axis

L – number of loops

Same goes to turning rate:

$$\begin{cases} Z_1 + LZ_r = Z_2 \\ A_1 + LA_r = A_2 \end{cases}$$

Z_1, Z_2 - initial and final height

A_1, A_2 – initial and final direction

Z_r, A_r - movement rate in Z axis and turning rate

```
{_x setpos [_pX/100-_fixX, _py/100-_fixY, _pZ/10]} foreach [lift,_user]
_fixX2=(getpos lift select 0)-(_pX/100); _fixY2=(getpos lift select 1)-(_pY/100)
{_x setpos [_pX/100-_fixX-_fixX2, _py/100-_fixY-_fixY2, _pZ/10]} foreach [lift,_user]
```

This code places lift and its user, calculates difference between current position and ordered position (`_fixX2`, `_fixY2`) and finally sets object in the right position.

When object's direction is changed its position is modified as well. That's why it needs to be fixed again.

```
? _move=="V" && _pZ>=160 : _move="H"; goto "Movement"
? _move=="H" && _py<=407358 : _move="V"; goto "Movement"
```

When lift reaches its destination the script switches movement type.

```
? _move=="V" && _pZ<=0 : LIFT_LEVEL=1; publicvariable "LIFT_LEVEL"; goto "Wait"
? _move=="H" && _py>=407394 : LIFT_LEVEL=2; publicvariable "LIFT_LEVEL"; goto "Wait"
```

When lift reaches its final destination the script changes `LIFT_LEVEL` variable and synchronizes it with other clients.