

TeleportswithselectionGUI

GarageGames.com



DocVersion1.0.x

Author: beffy(c++-programmer)

NEW! Download/view this tutorial as PDF!

This is an alternative version to the multi-Teleport Trigger explained [here](#). I've written a new C++ class (guiTeleportCtrl) based on Frank Bignon's MapGui which I've already "tuned" to show interiors and other players/bots... This one now also shows the "Teleport Trigger"s (name comparison!), which are red on this pic - the little black outline rectangles are interiors, the white dots are bots running around and the little white arrow above "Target3" is the player... you can turn interior and enemy rendering on/off, and you can adjust all the colors (triggers, text, interiors, background, ...) via script... and the rest of the GUI is completely up to you... just throw some bitmaps together, so there is no such thing as "design" in this 1st version!... -P

The GUI pops up as soon as you enter a trigger, but the teleporting only starts if you click on one of the "Target" text fields on the right...

Please note:

To get the MapCtrl based stuff (guiMapCtrlExt, guiTeleportCtrl) to compile, you need this additional function:

```
in engine/interior/interiorInstance.cc, add...

const char* InteriorInstance::getInteriorFileName() {
    return mInteriorFileName;
}
```

and in interiorInstance.h, simply add

```
const char* getInteriorFileName();
```

at the end of the "public" section...

Furthermore, you'll have to add the following function to "dgl/dgl.cc" (and declare it in dgl.h):

Thanks to sabrecyclo for reminding me...; -)

```
void dglDrawBitmapRotated(TextureObject *texture, const RectI& dstRect, const RectI& srcRect, const U32 in_flip, F32 spinAngle)
{
    AssertFatal(texture != NULL, "GSurface::drawBitmapStretchSR: NULL Handle");
    if (!dstRect.isValidRect())
        return;
    AssertFatal(srcRect.isValidRect() == true,
```

```

"GSurface::drawBitmapRotated: routines assume normal rects");

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture->texGLName);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glDisable(GL_LIGHTING);

F32 texLeft = F32(srcRect.point.x) / F32(texture->texWidth);
F32 texRight = F32(srcRect.point.x + srcRect.extent.x) / F32(texture->texWidth);
F32 texTop = F32(srcRect.point.y) / F32(texture->texHeight);
F32 texBottom = F32(srcRect.point.y + srcRect.extent.y) / F32(texture->texHeight);

F32 screenLeft = dstRect.point.x;
F32 screenRight = dstRect.point.x + dstRect.extent.x;
F32 screenTop = dstRect.point.y;
F32 screenBottom = dstRect.point.y + dstRect.extent.y;

if(in_flip & GFlip_X)
{
    F32 temp = texLeft;
    texLeft = texRight;
    texRight = temp;
}
if(in_flip & GFlip_Y)
{
    F32 temp = texTop;
    texTop = texBottom;
    texBottom = temp;
}

glColor4ub(sg_bitmapModulation.red,
            sg_bitmapModulation.green,
            sg_bitmapModulation.blue,
            sg_bitmapModulation.alpha);

F32 X = 0;

// calculate the centroid of the rectangle (to use as rotation pivot)
if (screenLeft < screenRight)
{
    X = screenLeft + ((screenRight - screenLeft)*0.5f);
}
else

```

```

{
    X = screenRight + ((screenLeft - screenRight)*0.5f);
};
F32 Y = 0;
if (screenTop < screenBottom)
{
    Y = screenTop + ((screenBottom - screenTop)*0.5f);
}
else
{
    Y = screenBottom + ((screenTop - screenBottom)*0.5f);
};
// spin angle is in degree's so convert to radians...radians = degrees * pi /180
spinAngle = spinAngle * 3.14 / 180.0f;
Point2F screenPoint(X,Y);
F32 width = dstRect.extent.x;
width *= 0.5;
MatrixF rotMatrix( EulerF( 0.0, 0.0, spinAngle ) );
Point3F offset( screenPoint.x, screenPoint.y, 0.0 );
Point3F points[4];
points[0] = Point3F(-width, -width, 0.0);
points[1] = Point3F(-width, width, 0.0);
points[2] = Point3F( width, width, 0.0);
points[3] = Point3F( width, -width, 0.0);
for( int i=0; i<4; i++ )
{
    rotMatrix.mulP( points[i] );
    points[i] += offset;
}
glBegin(GL_TRIANGLE_FAN);
glTexCoord2f(texLeft,texTop);
glVertex2fv(points[0]);
glTexCoord2f(texLeft, texBottom);
glVertex2fv(points[1]);
glTexCoord2f(texRight, texBottom);

```

```

glVertex2fv(points[2]);
    glTexCoord2f(texRight, texTop);
glVertex2fv(points[3]);
glEnd();
glDisable(GL_BLEND);
glDisable(GL_TEXTURE_2D);
}

```

You'll find all the **necessary files** (including the test bitmaps) in this zip file .

Here is a short description of what's going on and how to set it up...

First of all, let's look at the C++ stuff, which is based on Frank B.'s cool MapGui (or, to be exactly, the extended version) I've made which is also rendering the interiors and the other players/bots...

The function responsible for rendering the trigger objects is this one:

```

// now render TeleportTriggers:
F32 triggerScaleFactorSave = mTriggerScaleFactor;
SimSet * missionGroup = dynamic_cast(Sim::FindObject("MissionGroup"));
for(SimSetIterator itr(missionGroup); itr; ++itr)
{
    if ((*itr)->getType() & TriggerObjectType) {
        Trigger* trigger = static_cast(*itr);

        // if it's a "TeleportTrigger"...
        if (trigger->getType() & TriggerObjectType && (dstrcmp(trigger->getName(), "Teleport") != NULL)) {

            // Get center of object
            newCoord = trigger->getBoxCenter();
            shapeBox = trigger->getWorldBox();

            // Convert map coordinates to screen coordinates
            newCoord.x -= camCoord.x;
            newCoord.y -= camCoord.y;
            newCoord.y = -newCoord.y;
            float coord_z = newCoord.z - camCoord.z;
            newCoord.z = 0;

            // scale the position to the terrain size representation
            newCoord.x *= invSquareSize;
            newCoord.y *= -invSquareSize;

```

```

// Adjust object's vector
objectAngle = Vector3dToDegree(newCoord);
length = newCoord.len();
DegreeToVector2d( ( 360 - objectAngle ), length, newCoord);
newCoord.x = -newCoord.x;
newCoord.y = -newCoord.y;
newCoord.x += center.x;
newCoord.y += center.y;

// take care of the "player-bitmap" size
newCoord.x += texSize.x/2;
newCoord.y += texSize.y/2;

// check for a minimum size - every trigger below this will become scaled
if(static_cast(shapeBox.len_x()) < mMinTriggerSize || static_cast(shapeBox.len_y()) < mMinTriggerSize)
{
    // scale it
    mTriggerScaleFactor = 1/mMinTriggerScaleFactor;
}
else
{
    // leave unscaled - if not set otherwise through mScaleFactor (saved in triggerScaleFactorSave)
    mTriggerScaleFactor = triggerScaleFactorSave;
}

// start point is the object's center minus half the size (scaled according to the terrain block size,
// with additional scale factor to change the relative size of the triggers to the map)
Point2I boxStart(newCoord.x-((shapeBox.len_x()/(2*terrSquareSize))/mTriggerScaleFactor),
    newCoord.y-((shapeBox.len_y()/(2*terrSquareSize))/mTriggerScaleFactor));
// end point is the object's center plus half the size (scaled according to the terrain block size,
// with additional scale factor to change the relative size of the triggers to the map)
Point2I boxEnd(newCoord.x+((shapeBox.len_x()/(2*terrSquareSize))/mTriggerScaleFactor),
    newCoord.y+((shapeBox.len_y()/(2*terrSquareSize))/mTriggerScaleFactor));

// now draw the rectangle
dglDrawRectFill(boxStart, boxEnd, mTriggerColor);

// now draw the text with a little offset from the trigger rectangles and its own color ...
// change the name from "TeleportTriggerX" to "Target X" ...
char buf[256];
const char* name = trigger->getName();
const char* replace = dStrstr(name,"TeleportTrigger");
if ( !replace )
    continue;
char newName[256];
dStrncpy( newName, name, replace - name );
newName[replace - name] = 0;
dStrcat( newName, "Target ");

```

```

dStrcat( newName, replace + 15 );
dPrintf(buf, sizeof(buf), "%s", newName);
dglSetBitmapModulation(mTriggerTextColor);
dglDrawText(mProfile->mFont, Point2I(boxStart.x+1, boxStart.y+1), buf, mProfile->mFontColors);
    }
}

```

What's important here is that it only searches for triggers with "Teleport" in their name, so once again make sure you set up the names of your objects correctly (in the editor or in script)! The last piece of code simply replaces the trigger name with "TargetX" ...

There are some new warnings now compared to the MapGui, e.g. scale factors for the triggers, a minimum size of the triggers "mMinTriggerSize" (if below, they are scaled by this "mMinTriggerScaleFactor" - you wouldn't really see them otherwise in the little MapGui...) You should only have to change the various color values and maybe the "mMinTriggerScaleFactor", but the default values should work pretty well... all of them can be accessed via script/the GUI editor, so you can just play with them and see what happens...

Now let's look at the new trigger, I've made a new file "fps/server/target/TeleportTrigger.cs" which executes in "fps/server/game.cs", the important things are:

```

// new sound file played when entering the trigger now
dataBlock AudioProfile(TeleportInitSound)
{
    fileName = "~/data/sound/fx/zzap.wav";
    description = AudioClose3d;
    preload = true;
};
...
function TargetTeleportTrigger::onEnterTrigger(%data, %obj, %colObj)
{
    %client = %colObj.client;
    if(!%client)
    {
        echo("not a client!");
        return;
    }
    %checkName = %obj.getName();
    if($numTeleports == 0)
    {
        // search for Triggers by their name once, so every trigger
        // which has "TeleportTrigger" in its name is counted
        $numTeleports = getMultiTriggerCount("TeleportTrigger");
        echo("$numTeleports:" SPC $numTeleports);
        // set the text fields in the TeleportGui
        TeleportGui.initTextControls($numTeleports);
    }
}

```

```

}
if(%checkname != $currMultiTeleTrigger)
{
    // show and initialize TeleportGui
    $teleInitSound = serverPlay3D(TeleportInitSound,%client.player.getTransform());
    Canvas.pushDialog(TeleportGui);
    TeleportGui.initializeBeam(%client, %checkname);
}
}

```

Lookat

```

TeleportGui.initTextControls($numTeleports);
Canvas.pushDialog(TeleportGui);
TeleportGui.initializeBeam(%client, %checkname);

```

These are popping up the gui and calling functions in "fps/client/ui/TeleportGui.cs", which you can see here:

```

// set the text fields the first time a trigger is entered:
function TeleportGui::initTextControls(%this, %num)
{
    for(%i = 1; %i <= %num; %i++)
    {
        %actText = "TeleportLocation" @ %i;
        %actText.setText("Target" SPC %i);
    }
}
function TeleportGui::initializeBeam(%this, %callingClient, %triggerName)
{
    // save references to client and current trigger
    $teleportClient = %callingClient;
    $teleportCheckname = %triggerName;
}

```

They are simply setting the text fields in the gui and saving the current client and the current trigger name in global vars for later use.

Now in the gui there are 6 of these text fields (you can add as many as you want) which look like this (file "fps/client/ui/TeleportGui.gui"):

```

new GuiTextCtrl(TeleportLocation1) {
    profile = "GuiBoldGrayTextProfile";
    horzSizing = "right";
    vertSizing = "bottom";
    position = "253 168";
    extent = "38 16";
    minExtent = "8 8";
}

```

```

    visible = "1";
    helpTag = "0";
    maxLength = "255";
};

andfor eachofthem ,therearemouseeventhandlersin"TeleportGui.cs":

// mouse handlers for the target text fields to make them clickable
// if you need more triggers / teleports, add more text fields to TeleportGui.gui
// *and* here!

function TeleportLocation1:onMouseDown(%this, %obj) {
    TargetTeleportTrigger.startAction($teleportClient, $teleportCheckname, "1");
}
function TeleportLocation2:onMouseDown(%this, %obj) {
    TargetTeleportTrigger.startAction($teleportClient, $teleportCheckname, "2");
}
...

```

NOTE: The GuiTextCtrl doesn't have mouse event handlers by default, so you have to change this. Simply open up `"engine/gui/guiTextCtrl.h"` and add the following: (see http://tork.thesoulnomads.de/tutorials/teleport/teleport_select.php for details)

```

#ifdef _GIMOUSEEVENTCTRL_H_
#include "gui/guiMouseEventCtrl.h"
#endif

andthenchange

class GuiTextCtrl : public GuiControl

to

class GuiTextCtrl : public GuiMouseEventCtrl

andalsochange

typedef GuiControl Parent;

to

```

"engine/gui/guiTextCtrl.h" and add the following: (see http://tork.thesoulnomads.de/tutorials/teleport/teleport_select.php for details)

this thread

```
typedef GuiMouseEventCtrl Parent;
```

So if you click on one of the text entries in the GUI, "startAction()" in targetTeleportTrigger.cs is called, which checks the trigger name and if it is not the current trigger it starts the teleporting schedule, plays the sound, etc....

```
function TargetTeleportTrigger::startAction(%this, %callingClient, %checkname, %targetNum)
{
    %target = "TeleportTrigger" @ %targetNum;
    // we don't want to beam to the current teleport, do we?
    if(%target != %checkname)
    {
        CommandToClient(%callingClient, 'bottomprint', "Teleporter initializing... good luck... buahahaha!!", 2, 10);
        $teleSched = schedule(2000, 0, "goScotty", %callingClient, %target);
        $teleSound = serverPlay3D(TargetTeleportBuzz, %callingClient.player.getTransform());
        %callingClient.player.setCLOaked(true);
        // save the target - until the teleported client leaves it, then reset
        $currMultiTeleTrigger = %target;
    }
}
```

All the other stuff is pretty much the same as in the multiTeleportTrigger...

This time, you have to change your trigger data block in the mission file (if you've already used one of the other TeleportTriggers, if not, please see the "Teleportscript" tutorial for further explanations) from

```
dataBlock = "TeleportTrigger";

or

dataBlock = "MultiTeleportTrigger";

to

dataBlock = "TargetTeleportTrigger";
```

You could also use this for some transportation system, I guess we will end up using it as a subway system for our game... if the player enters a subway station, he can pop up the "subwayschedule GUI" or something like that and choose his target location - and if he enters the subway then, he gets "transported" to his destination.... -)

To "install" it,
 1) copy the **.ccand.hfiles** into "engine/gui", add them to your VC++ project and compile (if you encounter problems, do a CLEAN build)
 2) copy all the **bitmaps** into "fps/client/ui"

- 3) copy "TeleportGui.gui" and "TeleportGui.cs" into "fps/client/ui"
- 4) copy "zzap.wav" into "fps/data/sound/fx"
- 5) copy "targetTeleportTrigger.cs" into "fps/server/scripts"
- 6) execute scripts in "game.cs" and "init.cs" respectively
- 7) set up your triggers (or change the triggers' data blocks if you already have them in your mission), particle effects and shapes in the mission editor (or edit your "misfile" accordingly)...
- 8) **Note:** if you're missing some of the font profiles the GUI is using, you could either copy them from the "defaultProfiles.cs" provided with the zip or you can use different profiles for your GUI, of course...

Here's another little screenshot:



Well, that's pretty much it, I'm sure you've got plenty of own ideas as to how to use it

-hope you like it! -))

[<<Previous](#)

[Table of Contents](#)

[Author: beffy](#)

[Next >>](#)