Programming for Artists and Designers

Tutorial 5: HUDs - an Introduction

Aims
This tutorial aims to teach you about the Heads Up Display (HUD), especially including importing textures.
There is a lot to try out.

HUDs
Virtually all mods have made use of HUDs. They have either added additional functionality, such as to give them a scoreboard for flags captured, or to redesign the HUD and add new features such as the radar present in the Invasion bonus pack.

The Heads Up Display (HUD) is probably the most important part of a gametype (since it's normal function is to relay important information about the character's health, network status, scoring, etc).

It's also one of the most misunderstood components of the game engine. Many beginning coders have the misconception it exists on both client and server or that it can be used as part of the graphical user interface.
In reality, the Heads Up Display serves a simple purpose- to relay information about the state of the game to the client. However, because the HUD exists on the client, it's limited by the client's perception of the game state (network connection). Any information about the game should be specifically replicated from the server to the client.

Note
The HUD can look very confusing very quickly, if you open up nay HUD file you will see a plethora of functions. Don’t worry too much about this; a large portion of it is the manipulation of graphics – images, sprites and widgets.
It just means that you may have to play around in your own time if you want to do very fancy graphics.

Inheritence: Actor >> HUD (Package: Engine)

HUDbase then inherits from HUD.
HUDBDeathMatch then extends HUDbas.
HUDBTeamDeathMatch then extends HUDBDeathMatch
And finally the majority of other HUDs extend HUDBTeamDeathMatch.

All of these derived HUDs are stored in package *xInterface*
HUDs that we program will generally inherit from *HUDBTeamDeathMatch.*

Network Behaviour
Because the HUD is a purely clientside object, its accuracy is entirely dependant on the client's network connection to the server. Thus, if the client's connection is poor/lagged, the HUD's bBadConnectionAlert variable is set to true by the client's UNetConnection object during a Tick()*. The HUD then calls DisplayBadConnectionAlert() so that the client can somehow be notified the network connection is poor.

Even though the HUD does not have a server-side counterpart that replicates information about the game state, the HUD still has access to all data about the game via the owning PlayerController (Engine.PlayerController).

For instance, the following HUD code iterates through the list of Controllers on the client.

```
local Controller C;
foreach AllActors(class'Controller', C)
{
          //Do something with the Controller information
}
```

Notice, we cannot iterate through all the Controllers using the Level's ControllerList since this data isn't replicated.

A common source for accessing the game state is the PlayerController's GameReplicationInfo (GRI) object. Since it exists on both the client and the server, it can be used to get a lot useful information about the current state of the game on the server.

Gametype
The Heads Up Display is dynamically created on the server by the gametype's default HUDType during a GameInfo.PostLogin() event. However, the object is immediately replicated to the PlayerController that just logged into the game. The HUD object, from this point on, exists purely on the client machine.

GameReplicationInfo
A GameReplicationInfo is used by HUD objects to derive information about the game state. These objects usually store data like the GameName, the goal score, and Time limit. Pretty much any info about the gametype will probably end up here.

Importing Textures and Drawing to the HUD
Check out Sega 500’s (Jeff Giles) ppt lesson on HUDs and importing textures. It is stored in the week 5 folder at activehelix.com. This is an excellent lesson.

HUD Information and some of the HUD Functions

Canvas

Check Out: http://udn.epicgames.com/Technical/CanvasReference for a canvas reference.
The Canvas is basically the two dimensional surface that the HUD draws on. Therefore, most functions in the HUD class will accept a Canvas as an argument. The Canvas class contains a lot of important functions and you'll find that drawing on the Canvas is a bit different the two dimensional drawing in other languages. For instance, the Canvas should be repositioned before drawing different elements.

The Canvas defines many important variables. Some of the most important are:

ClipX = Screen horizontal size
ClipY = Screen vertical size
CurX = the 'cursor' horizontal position
CurY = the 'cursor' vertical position

DrawTile

DrawTile is probably one of the most confusing functions for beginning Canvas scripters. Perhaps because the definition looks sort of intimidating. However, contrary to it's first impression, Canvas.DrawTile is one of the easiest and powerful functions to use.

```
native(466) final function DrawTile( texture Tex, float XL, float YL, float U, float V, float UL, float VL );
```

Tex	Source Image
XL	Display Width
YL	Display Height
U	Starting X Position
V	Starting Y Position
UL	Horizontal Region
VL	Vertical Region

DrawTile accepts a lot of arguments because it can be used to grab small sections from an image and use that region as a texture. The U,V,UL,VL arguments allows us to specify the rectangular region of the image we desire to use. The XL,YL arguments allows us to define the dimensions of the image. Thus, if XL = Canvas.ClipX and YL = Canvas.ClipY, then the image will be stretched to fit the entire screen.

PostRender

Canvas.PostRender() is one of the most important functions that makes up the Heads Up Display. Images, Text, or even actors can be rendered via the Canvas argument. The HUD is active at all times, except during level loading (even though the last image rendered to the canvas will still be displayed).

The following code can be used to display the Level's screenshot during level loading (if the GUI were to explicitly set the texture before the client travels to the new map).

```
class TestHUD extends HUD;

#exec TEXTURE IMPORT NAME=pHUD FILE=Textures\traintag.pcx
GROUP="HUD" MIPS=OFF FLAGS=2

var texture T;

simulated event PostRender( canvas Canvas )
{
    Super.PostRender(Canvas);
    if (Level.TimeSeconds < 25.5)
       RenderScreenshot(Canvas);
}

function RenderScreenshot( Canvas C )
{
   local float XWidth, YHeight, XMod, YMod;


   XMod = C.ClipX;
   YMod = C.ClipY;

   //Reset the Canvas in case the rendering mode has been
changed
   C.Reset();
   C.SetPos(0,0);
   C.DrawTile(T, XMod, YMod, 0, 0, t.USize, t.VSize );

}

defaultproperties
{
   T=Texture'pHUD'
}
```

DrawHUD

```
simulated function DrawHudPassA (Canvas C); // Alpha Pass
simulated function DrawHudPassB (Canvas C); // Additive Pass
simulated function DrawHudPassC (Canvas C); // Alpha Pass
simulated function DrawHudPassD (Canvas C); // Alternate
Texture Pass
```

Each pass takes care of drawing a different aspect of the HUD.

NumericWidget and SpriteWidget:
In defaultproperties the graphical look of both elements are set. I'm going to specifically look at ShieldCount which is the total amount of shield points that a player has in the game.

ShieldCount=(RenderStyle=STY_Alpha,TextureScale=0.100000,DrawPivot=DP_LowerLeft,PosX=0.500000,PosY=0.500000,OffsetX=0,OffsetY=0,Tints[0]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

Now, let’s look specifically at what is being set here in detail and go blow-by-blow at each of these attributes.
RenderStyle

RenderStyle is an ERenderStyle enumeration. It controls the type of color blending or it's affect on the other units that it lays over in the HUD.
TextureScale

Texture Scale is the scale that is applied to the texture, relative to a screen resolution of 640 x 480 pixels (that means: at that screen resolution a sprite with TextureScale=1.0 will be drawn one-to-one; at other screen resolutions it will be proportionally scaled up or down).
Drawpivot

Drawpivot is a point on the texture or numeric image.

DP_UpperLeft
DP_UpperMiddle
DP_UpperRight
DP_LowerLeft
DP_LowerMiddle
DP_LowerRight
DP_MiddleLeft
DP_MiddleMiddle
DP_MiddleRight

**This is a very good thing for you play around with so that you get the hang of manipulating widgets.*

PosX, PosY, OffsetX, OffsetY
The values of PosX and PosY are decimal values representing the position on the screen. This frees the user from having to worry about the position of their HUD items in relation to the screen resolution. So if you set the PosX=0.5000000 and PosY=0.5000000 then you always get a position in the centre of the screen.

OffsetX and OffsetY are in actual pixels. You might be able to get to the point of the map that you want, but then you want to shift it a specific number of pixels and this is where OffsetX and OffsetY would come into play.

Tints
This is the colour of the mesh/text and can be dependant on the team (in three colour intensity format). Tint[0] is the first team and Tint[1] is the second team. For armour it isn’t important to have the difference in colour. I would recommend using a graphical colour chooser to find the Red-Green-Blue (three-color intensity) you are looking for.

SpriteWidget:
So here are the properties specific to the sprite widget call. Up to this point a lot of them are already covered in the NumericWidget.
WidgetTexture

WidgetTexture loads the texture from a texture pack for the SpriteWidget. You are able to view all the textures that UT2K3 uses through UnrealEd. Open up the unreal ED and have a look.

```
...WidgetTexture=Texture'InterfaceContent.HUD.SkinA'...
```

TextureCoords
Textures usually come with several on the same sheet. So you often need to specify the rectangle that surrounds the texture that you want. In this case we just want to grab the shield from that texture pack, and so there is a start point (x,y) and then a length in both directions.

The texture scale is a percentage at which you want the texture to be displayed. This is useful as the texture picture is usually larger than you want it to appear in your HUD.
ScaleMode

SM_None
SM_Up
SM_Down
SM_Left
SM_Right

This is the side that it is scale from. This is used for the bars that adjust with your health and ammo. Most of those start from the left side. But this is if you plan to have your texture start to get chopped off through use.

Scale
This is a float value that represents how much is cut off from the direction specified above. So for ammo depletion they probably calculate an amount appropriate based on the maximum and how much you are firing off.

Drawing with Numeric and Sprite Widget:

You may noticed that there are two function calls: DrawSpriteWidge and DrawNumericWidget.
These are native functions that take care of drawing the two widgets we defined above.

```
DrawSpriteWidget (Canvas C, out SpriteWidget W) [simulated,
final]
// Draws the HUD element described by the SpriteWidget W.

DrawNumericWidget (Canvas C, out NumericWidget W, out DigitSet
D) [simulated, final]
```

```
// Draws a number.
```

DrawScreenText:
```
DrawScreenText (String Text, float X, float Y, EDrawPivot
Pivot)
```

`String Text` is the literal string you wish to print to the screen; generally a string variable is passed to it. The variable declared at the top of the file and defined in the default section. If you don't need no how to declare variables in unreal script then look at
http://udn.epicgames.com/Technical/UnrealScriptReference

`float X & float Y` are co-ordinates.

`EDrawPivot Pivot` Is a pivot, see above in NumericWidget and SpriteWidget section.

Exercise

As per usual the format will be to work through the example so that by the end you have a working HUD.

If there are functions or variables that are called or referred to that aren't defined in the class we are scripting then they have been defined in other classes that this one inherits from.

- ***Important:*** *You will have to download the textures for this tutorial to work successfully – they can be found on the activehelix website*

The Scripting

We are going to be scripting a HUD called `HudBToughToTheTop` this will inherit its base information from `HudBTeamDeathMatch`.

The inheritance follows this pathway:

```
Object
 +-Actor
    +-HUD
       +-CinematicHud?
       +-HudBase
          +-HudBDeathMatch?
             +-HudBTeamDeathMatch?
             +-HudBBombingRun?
             +-HudBCaptureTheFlag?
             +-HudBDoubleDomination?
```

```
//=========================================================
// HUD Tough To The Top - Custom HUD for Tough To The Top
//
// A Custom HUD that strips most of the Deathmatch's default HUD
// away, adds in
// custom textures and text, whilst repositioning most of the HUD's
// Info, such as Health and GameScore.
//
// NOTE: The UT2003's Screen Resolution must be set to 800x600 for
// the HUD to be displayed out correctly.
//
//============================================================

//Custom HUD derives from HudBTeamDeathMatch
class HudBToughToTheTop extends HudBTeamDeathMatch;
```

- Here we are just declaring that our HUD is inheriting its base functionality from HudBTeamDeathMatch.

```
//Import Texture Files
#exec texture IMPORT name=HudHealth FILE=Textures\HudHealth.bmp
GROUP="HUD" MIPS=OFF FLAGS=2
#exec texture IMPORT name=HudItems FILE=Textures\HudItems.bmp
GROUP="HUD" MIPS=OFF FLAGS=2
```

- Here we are importing our textures – see below for the methods of texture importing – we are basically giving the image a name e.g. “HudHealth” and telling the exec function (#exec) where to find the image. The image is in our texture folder as part of our package.

```
//Declare Variables
var texture tex;
var texture tex2;

var localized string         Health;
var localized string         GameInfo;
var localized string         GameScore;
var localized string         Spread;
var localized string         Rank;
var localized string         Items;
```

- Here we are setting the variables to be used in the class. The texture variables will be used to refer to the textures that we are importing.

```
//Draw Text "Health" on Screen
//Declares that a function may execute on the client-side when an
//actor is either a simulated proxy or an
//autonomous proxy
simulated function drawHealthText(Canvas C)
{
 text = Health; //Use the String Health (defined in defaultpropertes)
to set text

    C.Font = LoadLevelActionFont(); //Load Font to write text with

   C.DrawColor = LevelActionFontColor; //Set the colour for the text

  C.Style = ERenderStyle.STY_Alpha; //How text will appear on screen
```

```
   C.DrawScreenText (text, 0.02, 0.002, DP_UpperLeft); //Set text's
Position
}
```

- Here we have coded our own function to draw the health text – we have moved it around from it usual position
- First we have set the font, font colour and the render style – the attributes of the text

```
simulated function drawGameInfoText(Canvas C)
{
                           text = GameInfo;

                   C.Font = LoadLevelActionFont();

                 C.DrawColor = LevelActionFontColor;

                  C.Style = ERenderStyle.STY_Alpha;

         C.DrawScreenText (text, 0.98, 0.01, DP_UpperRight);
}


simulated function drawGameScoreText(Canvas C)
{
                          text = GameScore;

                   C.Font = LoadLevelActionFont();

                 C.DrawColor = LevelActionFontColor;

                  C.Style = ERenderStyle.STY_Alpha;

         C.DrawScreenText (text, 0.86, 0.05, DP_UpperRIght);
}


simulated function drawSpreadText(Canvas C)
{
                            text = Spread;

                   C.Font = LoadLevelActionFont();

                 C.DrawColor = LevelActionFontColor;

                  C.Style = ERenderStyle.STY_Alpha;

          C.DrawScreenText (text, 0.86, 0.1, DP_UpperRight);
}


simulated function drawRankText(Canvas C)
{
                             text = Rank;

                   C.Font = LoadLevelActionFont();

                 C.DrawColor = LevelActionFontColor;
```

```
                  C.Style = ERenderStyle.STY_Alpha;

         C.DrawScreenText (text, 0.86, 0.15, DP_UpperRight);
}


simulated function drawItemsText(Canvas C)
{
                            text = Items;

                   C.Font = LoadLevelActionFont();

                  C.DrawColor = LevelActionFontColor;

                  C.Style = ERenderStyle.STY_Alpha;

         C.DrawScreenText (text, 0.02, 0.45, DP_LowerLeft);
}
```

- These functions act in a similar way to the drawHealthText() function in that we are writing information to the screen after setting the font and render style.

```
//Draw Text Functions on Screen
simulated function UserDrawText(Canvas c)
{
        drawHealthText(c);
        drawGameInfoText(c);
        drawGameScoreText(c);
        drawSpreadText(c);
        drawRankText(c);
        drawItemsText(c);
}
```

- Here we have simply defined a function that will call our other functions in a row

```
simulated function UpdateRankAndSpread(Canvas C)
{
    local int i;

        if ( (Scoreboard == None) || !Scoreboard.UpdateGRI() )
                                                            return;

    for( i=0 ; i<PlayerOwner.GameReplicationInfo.PRIArray.Length ;
i++ )
         if(PawnOwnerPRI ==
PlayerOwner.GameReplicationInfo.PRIArray[i])
         {
            myRank.Value = (i+1);
            break;
         }

myScore.Value = Min (PawnOwnerPRI.Score, 999);  // max display space

  if ( PawnOwnerPRI == PlayerOwner.GameReplicationInfo.PRIArray[0] )
      {
      if ( PlayerOwner.GameReplicationInfo.PRIArray.Length > 1 )
              {
```

```
              mySpread.Value = Min (PawnOwnerPRI.Score -
PlayerOwner.GameReplicationInfo.PRIArray[1].Score, 999);
              }
              else
       {
                  mySpread.Value = 0;
       }
}
else
{
                  mySpread.Value = Min (PawnOwnerPRI.Score -
       PlayerOwner.GameReplicationInfo.PRIArray[0].Score, 999);
}

                                                   if( !bShowPoints )
            {
                           DrawNumericWidget (C, myScore, DigitsBig);

                if ( C.ClipX >= 640 )
                     {
                         DrawNumericWidget (C, mySpread, DigitsBig);
                               DrawNumericWidget (C, myRank, DigitsBig);
                   }
            }

                        if(myRank.Value > 9)
            {
                      myRank.TextureScale = 0.12;
                               myRank.OffsetX = 240;
                            myRank.OffsetY = 90;
       }
         else
            {
                            myRank.TextureScale = 0.18;
                            myRank.OffsetX = 150;
                      myRank.OffsetY = 40;
            }
}
```

- This large function is redefined from previous HUDs, the method has been inherited. It is responsible for drawing the updated scores to the screen.

```
//Draw Textures on Screen
function DrawHud(Canvas c)
{
    super.DrawHud(c);
    DrawTex(c);
    //DrawTex2(c);
    bShowPoints=false; //Set to false so bare minimum of DM HUD is
                       //shown.
    bShowPersonalInfo=false;
}
```

- This function has been defined in other classes, hence the call to super. We have set it to call our texture drawing function.

```
simulated function DrawHudPassC (Canvas C)
{
        if ( PawnOwner.Weapon.bShowChargingBar )
        {
            ReloadingFill.Scale = PawnOwner.Weapon.ChargeBar();

            DrawSpriteWidget (C, ReloadingFill);
            DrawSpriteWidget (C, ReloadingTeamTint);
            DrawSpriteWidget (C, ReloadingTrim);
        }


        if( (PawnOwner.Weapon.Ammo[0] != None) &&
(PawnOwner.Weapon.Ammo[0].IconMaterial != None) )
        {
            AmmoIcon.WidgetTexture =
PawnOwner.Weapon.Ammo[0].IconMaterial;
            AmmoIcon.TextureCoords =
PawnOwner.Weapon.Ammo[0].IconCoords;
            DrawSpriteWidget (C, AmmoIcon);
        }

                       if (!bShowPersonalInfo)
                                  {
            DrawNumericWidget (C, AdrenalineCount, DigitsBig);
            DrawSpriteWidget (C, AdrenalineIcon);

              DrawNumericWidget (C, HealthCount, DigitsBig);
                    DrawSpriteWidget (C, HealthIcon);

                       if( ShieldCount.Value > 0 )
                                    {
                                    DrawSpriteWidget (C, ShieldIcon);
                                  DrawNumericWidget (C, ShieldCount, DigitsBig);
                                    }

      if( PawnOwner.IsA ('XPawn') && (XPawn(PawnOwner).UDamageTime >
Level.TimeSeconds) )
                                    {
                                  UDamageFill.Scale = FMin((XPawn(PawnOwner).UDamage

                                   DrawSpriteWidget (C, UDamageFill);
                                  DrawSpriteWidget (C, UDamageTeamTint);
                                   DrawSpriteWidget (C, UDamageTrim);
                                    }

                            UserDrawText(c);
                            DrawCrosshair(C);

                                  }

        if( !bShowPersonalInfo && (ShieldCount.Value > 0) )
        {
           DrawSpriteWidget (C, ShieldIconGlow);
        }

    if( bShowWeaponInfo && (PawnOwner != None) && (PawnOwner.Weapon
!= None) )
                                  {
              DrawNumericWidget (C, AmmoCount, DigitsBig);
                   PawnOwner.Weapon.DrawWeaponInfo(C);
                                  }
}
```

- We have altered this function so that the images and numbers are drawn in the position we wish. This function as mentioned above is defined in previous HUD classes and is one of main components for drawing information on the screen

```
function DrawTex(Canvas c)
{
    local color tempColor;
    local byte tempstyle;

    tempColor = c.DrawColor; //Store values
    tempstyle= c.Style;

    c.DrawColor=WhiteColor;
    c.Style=ERenderStyle.STY_Modulated;

    c.SetPos(8,25);   //Set Position
    c.DrawIcon(tex,1);  //Set Scale

    c.DrawColor = tempColor;

    c.SetPos(2,263);
    c.DrawIcon(tex2,1);

    c.DrawColor = tempColor;
    c.Style = tempstyle;
}
```

- This function simply draws our images (textures) to the screen. We set our style, colour and alpha (transparency values) first, along with the screen position.

```
defaultproperties
{

tex=texture'HudHealth'
tex2=texture'HudItems'

Health="Health" //Text for Health String
GameInfo="Game Info"
GameScore="Score"
Spread="Spread"
Rank="Rank"
Items="Items"

mySpread=(RenderStyle=STY_Alpha,MinDigitCount=2,TextureScale=0.150000
,DrawPivot=DP_UpperRight,PosX=0.913,PosY=0.0925,OffsetX=150,OffsetY=4
0,Tints[0]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=25
5))

myRank=(RenderStyle=STY_Alpha,MinDigitCount=2,TextureScale=0.150000,D
rawPivot=DP_UpperRight,PosX=0.91,PosY=0.135,OffsetX=150,OffsetY=40,Ti
nts[0]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

myScore=(RenderStyle=STY_Alpha,MinDigitCount=2,TextureScale=0.150000,
DrawPivot=DP_UpperRight,PosX=0.913,PosY=0.0425,OffsetX=150,OffsetY=40
,Tints[0]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255
))

AdrenalineCount=(RenderStyle=STY_Alpha,TextureScale=0.180000,DrawPivo
t=DP_LowerLeft,PosX=0.001000,PosY=0.450000,OffsetX=210,OffsetY=110,Ti
nts[0]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

AdrenalineIcon=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',Ren
derStyle=STY_Alpha,TextureCoords=(Y1=620,X2=142,Y2=749),TextureScale=
0.230000,DrawPivot=DP_LowerLeft,PosX=0.000100,PosY=0.475000,OffsetX=2
0,OffsetY=55,Scale=1.000000,Tints[0]=(B=255,G=255,R=255,A=255),Tints[
1]=(B=255,G=255,R=255,A=255))

HealthCount=(RenderStyle=STY_Alpha,TextureScale=0.250000,DrawPivot=DP
_UpperLeft,PosX=0.000001,PosY=0.000001,OffsetX=140,OffsetY=110,Tints[
0]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

HealthIcon=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',RenderS
tyle=STY_Alpha,TextureCoords=(Y1=750,X2=142,Y2=879),TextureScale=0.30
0000,DrawPivot=DP_UpperLeft,PosX=0.000001,PosY=0.005000,OffsetX=1,Off
setY=55,Scale=1.000000,Tints[0]=(G=255,A=255),Tints[1]=(G=255,A=255))

ShieldCount=(RenderStyle=STY_Alpha,TextureScale=0.250000,DrawPivot=DP
_LowerLeft,PosX=0.010000,PosY=0.590000,OffsetX=140,OffsetY=55,Tints[0
]=(B=255,G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

ShieldIconGlow=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',Ren
derStyle=STY_Alpha,TextureCoords=(X1=333,Y1=584,X2=457,Y2=749),Textur
eScale=0.260000,DrawPivot=DP_LowerLeft,PosX=0.000100,PosY=0.600000,Of
fsetX=57,OffsetY=55,ScaleMode=SM_Left,Scale=1.000000,Tints[0]=(B=255,
G=255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

ShieldIcon=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',RenderS
tyle=STY_Alpha,TextureCoords=(X1=458,Y1=584,X2=583,Y2=749),TextureSca
le=0.250000,DrawPivot=DP_LowerLeft,PosX=0.000100,PosY=0.600000,Offset
X=60,OffsetY=55,Scale=1.000000,Tints[0]=(B=255,G=255,R=255,A=255),Tin
ts[1]=(B=255,G=255,R=255,A=255))


UDamageTeamTint=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',Re
nderStyle=STY_Alpha,TextureCoords=(X1=836,Y1=490,X2=450,Y2=454),Textu
reScale=0.300000,DrawPivot=DP_LowerLeft,PosX=0.000100,PosY=0.500000,O
ffsetX=20,OffsetY=55,ScaleMode=SM_Right,Scale=1.000000,Tints[0]=(R=10
0,A=100),Tints[1]=(B=102,G=66,R=37,A=150))

UDamageTrim=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',Render
Style=STY_Alpha,TextureCoords=(X1=836,Y1=453,X2=450,Y2=415),TextureSc
ale=0.300000,DrawPivot=DP_LowerLeft,PosX=0.000100,PosY=0.500000,Offse
tX=20,OffsetY=55,ScaleMode=SM_Right,Scale=1.000000,Tints[0]=(B=255,G=
255,R=255,A=255),Tints[1]=(B=255,G=255,R=255,A=255))

UDamageFill=(WidgetTexture=Texture'InterfaceContent.HUD.SkinA',Render
Style=STY_Alpha,TextureCoords=(X1=836,Y1=490,X2=450,Y2=454),TextureSc
ale=0.300000,DrawPivot=DP_LowerLeft,PosX=0.000100,PosY=0.500000,Offse
tX=20,OffsetY=55,ScaleMode=SM_Right,Scale=1.000000,Tints[0]=(B=255,R=
255,A=255),Tints[1]=(B=255,R=255,A=255))
}
```

- Here we set the properties for our variables as normal

- It is larger than normal as we have to redefine the positions of many of the HUD widgets

Including you HUD in Your Gametype

To include your HUD in your gametype you need to add this line to the default properties of your UC game file.

```
HUDType="MyPackage.HUDname"
```

`MyPackage` is simply the package that you create and are saving your work.
`HUDname` is the actual name of your HUD class.

So if we wished to use this HUD as part of our TopKiller game type stored in CVGPackage we would add the line:

```
HUDType="CVGPackage.HudBToughToTheTop"
```

To the default properties of our TopKiller game type class.

INT Files

You will be pleased to know that you don’t include a line in the .int file for your HUD.

Textures

Loading Textures

I have found that loading textures through the editor and the using
`#exec OBJ LOAD FILE="..\textures\MyHUDImage.utx"`
Is a much better way of doing it; it caused a lot less problems….with the exception that sometimes it refuses to save it.

Textures in General

There are many good unreal resources as regards textures at 3dbuzz.com

Terms

Simulated

Declares that a function may execute on the client-side when an actor is either a simulated proxy or an autonomous proxy. All functions that are both native and final are automatically simulated as well.