Advanced FPS Kit

AI Lite Pack

Missile & Targeting Overlay Installation / Documentation

I. C++ Installation

If you have not installed a prior portion of the AFPSK that installs the team field on the ShapeBase object, you will need to make the following changes.

Please make the following edits to ShapeBase.h. First, locate the block of code where mDamage is defined, it will be right after a protected definition, add the mTeam definition above it:

```
protected:
S32 mTeam; //Phantom139: Added
/// @name Damage
/// @{
F32 mDamage;
```

Next, scroll down to `enum ShapeBaseMasks {` and add a new mask to the list like so:

```
enum ShapeBaseMasks {
   NameMask = Parent::NextFreeMask,
   TeamMask = Parent::NextFreeMask << 1,
   DamageMask = Parent::NextFreeMask << 2,
   NoWarpMask = Parent::NextFreeMask << 3,
   CloakMask = Parent::NextFreeMask << 4,
   ShieldMask = Parent::NextFreeMask << 5,
   InvincibleMask = Parent::NextFreeMask << 6,
   SkinMask = Parent::NextFreeMask << 7,
   MeshHiddenMask = Parent::NextFreeMask << 8,
   SoundMaskN = Parent::NextFreeMask << 9, ///< Extends + MaxSoundThreads
bits
   ThreadMaskN = SoundMaskN << MaxSoundThreads, ///< Extends +
MaxScriptThreads bits
   ImageMaskN = ThreadMaskN << MaxScriptThreads, ///< Extends +
MaxMountedImage bits
   NextFreeMask = ImageMaskN << MaxMountedImages
};
```

Then, scroll near the bottom of the ShapeBase class definition and you will see the following definitions:

```
F32 getLiquidHeight() { return mLiquidHeight; }
virtual WaterObject* getCurrentWaterObject();
```

```
void setCurrentWaterObject( WaterObject *obj );
virtual F32 getMass() const { return mMass; }
```

After these, add these two lines:

```
virtual S32 getTeam() { return mTeam; }
void setTeam(S32 newt) { mTeam = newt; setMaskBits(TeamMask); }
```

Save the file, then open and make the following edits to ShapeBase.cpp
First, scroll down to the ShapeBase constructor (It's located around line 900 with the format: ShapeBase::ShapeBase() : Stuff(), . . .

Inside the long list of definitions you will see this variable definition:

```
damageDir( 0.0f, 0.0f, 1.0f ),
mShapeBaseMount( NULL ),
mMass( 1.0f ),
mOneOverMass( 1.0f ),
```

Add a mTeam definition to the list:

```
damageDir( 0.0f, 0.0f, 1.0f ),
mShapeBaseMount( NULL ),
mMass( 1.0f ),
mTeam( 0 ), //< Add This
```

Next scroll down to ShapeBase::onNewDataBlock and add a mTeam definition to the list of pre-defined values as so:

```
//
mTeam = 0;
mEnergy = 0;
mDamage = 0;
mDamageState = Enabled;
```

Next, scroll way down in the file to: `U32 ShapeBase::packUpdate(NetConnection *con, U32 mask, BitStream *stream)` and make the following changes exactly as they appear here. Making a mistake here will lead to in-game crashing and access violations so you need to ensure everything here matches.

Near the bottom you will see the following "if" statement:

```
if (stream->writeFlag(mask & NameMask)) {
```

Above this if statement, add the following code block:

```
if (stream->writeFlag(mask & TeamMask)) {
// Phantom: team
stream->write( mTeam );
//Phantom139: Done
}
```

Then scroll down to the unpack update function and find this if statement:

```
if (stream->readFlag()) { // NameMask
```

Then add the following code block above it.

```
if (stream->readFlag()) { // TeamMask
//Phantom: Added Team
stream->read(&mTeam);
//Phantom139: End
}
```

Lastly, add the following to the bottom of the file:

```
//Phantom: Added Team Methods:
DefineEngineMethod( ShapeBase, getTeam, S32, (),,
"@brief Get the current team number used by this shape.\n\n"
"@return the team number\n\n") {
return object->getTeam();
}
DefineEngineMethod( ShapeBase, setTeam, void, (S32 newTeam),,
"@brief Set the current team number of this shape.\n\n"
"@return void\n\n") {
object->setTeam(newTeam);
}
```

Save and compile. This adds internal team numbers to the engine which will be used by the CTF example, and can also be used for numerous applications beyond game modes.

II. Missile Projectile

The MissileProjectile class uses a MissileProjectile datablock which has the exact same parameters as the standard Projectile datablock. The only difference is the introduction of the accuracy parameter, which is a number between 1 and 100 that determines how closely a missile will follow a target's movement vector.

The MissileProjectile has setTarget(Object o); and getTarget(); methods to use the targeting function on the Missile. TorqueScript examples are provided for usage of the MissileProjectile.

III. GuiTargetingOverlay

The GuiTargetingOverlay is the accompanying function to the MissileProjectile and is used to find and set targets for the MissileProjectile on the client end. The control provides functioning for both the client and server end. The following functions for the client-side of the control are directly applied to the control and the syntax follows:

- GuiTargetingOverlay::setupParams(bool enabled, String l1, String l2, F32 range, S32 neededTicks, U32 typemask): This controls the various functions and displays of the GUI. The recommended usage of this is directly through a clientCmd. For further documentation on the applications of each of these parameters, see the provided TS Example on the client-end.
- GuiTargetingOverlay::forceLockBreak(): This function directly breaks a lock that is currently occurring and enforces a 3-second delay between the client's next lock.
- GuiTargetingOverlay::getLockedTarget(): While all of the locking operations occur through transmitted commands, this is a method to return the ghosted ID of the client's locked target.

On the server end of the provided functions list, there are a few serverCmds provided by the control and a new function that can be called on the server end for parsing ghost information between a client and the server.

- serverCmdSendLockWarnEvent(S32 ghostID): This function is called when a client first acquires a target lock on an object. You will need to use the provided function below to translate this to the server's object ID.
- serverCmdSendCompleteLockEvent(S32 ghostID): This function is called when a client completes a target lock on an object. You will need to translate this into an object ID.
- serverCmdSendLockBreakEvent(S32 ghostID): This function is called when a client breaks lock on a target object.
- resolveGTCTarget(S32 clientID, S32 ghostID): This function translates a client's ghost object to the server object. It's a very useful function even beyond the gui itself to translate ghost ID's to server object ID's.

A full application example is provided in the TS folders. To use the gui, simply add it to the PlayGui (it must be a child of the PlayGui), and set it to cover the entire extent of the Gui and proceed.