Jagged Alliance 2 Source

Wie man ein Attachment hinzufügt, das die Munitionskapazität erhöht

Vorüberlegung:
Wir wollen ein Attachment über den Source einfügen, das – ist es an einer Waffe angebracht – die Munitionskapazität dieser Waffe erhöht. Nun werfen sich direkt einige grundlegenden Fragen auf:

- Welches Item soll als Attachment dienen?
- In welcher Weise soll es die Munitionskapazität erhöhen?
- Wie stellen wir es an, dass sich die Muntionskapazität erhöht?

Wenn wir uns die WEAPONTYPE-Klasse ansehen, müssen wir mit Schrecken feststellen, dass es in dieser Klasse zwar einen Eigenschaft ubMagSize gibt, diese aber für alle Waffen des gleichen Typs gilt. Das heißt, dass wenn wir diesen Wert ändern, ändern wir die Werte für z.B. alle StyerAUGs im Spiel. Um dies zu umgehen, müssen wir überlegen, wie wir es anstellen können, für jede Waffe individuell die Magazingröße zu ändern, also nur für genau die Waffe, an der das Upgrade-Item angebracht wird.
Wir lösen dieses Problem, indem wir schauen, welche Klasse für jedes einzelne Item verwendet wird, also von welcher Klasse alle Items Instanzen sind. Nahe liegt die OBJECTTYPE-Klasse. Das einzige Problem ist, dass dort zwar Eigenschaften für z.B. verbliebene Schüsse im Magazin einer Waffe sind, aber keine Eigenschaft für die Magazingröße. Aber was es nicht gibt, erschaffen wir einfach. Legen wir dazu eine Eigenschaft ubMagSize an:

Tactical\Item Types.h, typedef struct OBJECTTYPE

```
typedef struct
{
        UINT16  usItem;
        UINT8           ubNumberOfObjects;

[...]

 // attached objects
        UINT16  usAttachItem[MAX_ATTACHMENTS];
        INT8            bAttachStatus[MAX_ATTACHMENTS];

        INT8            fFlags;
        UINT8           ubMission;
        INT8            bTrap;      // 1-10 exp_lvl to detect
        UINT8           ubImprintID;    // ID of merc that item is imprinted on
        UINT8           ubWeight;
        UINT8           fUsed;                          // flags for whether the item is used or not

        UINT8           ubMagSize;
} OBJECTTYPE;
```

Soweit, so gut. Das erste Problem hätten wir damit gelöst – jede Waffe hätte nun (zumindest theoretisch) eine eigene Magazingröße. Nur müssen wir dazu noch einige weitere Veränderungen vornehmen, damit das Programm sich überhaupt dieser Möglichkeit besinnt und sie dann auch nutzt. Im Klartext bedeutet das nun, dass wir, wann immer sich JA2 auf die Magazingröße in der Weapons table bezieht, ansetzen müssen und auf unsere neue ubMagSize Eigenschaft verweisen.

Im Folgenden wird aufgelistet, welche Stellen betroffen sind und wie wir sie abändern müssen. In grau dargestellte Passagen müssen durch die Rotgedruckten ersetzt werden.

Tactical\Weapons.c, void ReloadWeapon

```
void ReloadWeapon( SOLDIERTYPE *pSoldier, UINT8 ubHandPos )
{
	// NB this is a cheat function, don't award experience

	if ( pSoldier->inv[ ubHandPos ].usItem != NOTHING )
	{
		pSoldier->inv[ ubHandPos ].ubGunShotsLeft = Weapon[ pSoldier->inv[
ubHandPos ].usItem ].ubMagSize;
		pSoldier->inv[ ubHandPos ].ubGunShotsLeft = pSoldier->inv[ ubHandPos
	].ubMagSize;
		// Dirty Bars
		DirtyMercPanelInterface( pSoldier, DIRTYLEVEL1 );
	}
}
```

Tactical\Points.c, INT8 GetAPsToReloadGunWithAmmo(…)

```
INT8 GetAPsToReloadGunWithAmmo( OBJECTTYPE * pGun, OBJECTTYPE * pAmmo )
{
	if (Item[ pGun->usItem ].usItemClass == IC_LAUNCHER)
	{
		// always standard AP cost
		return( AP_RELOAD_GUN );
	}
	if ( Weapon[pGun->usItem].ubMagSize == Magazine[Item[pAmmo-
>usItem].ubClassIndex].ubMagSize )
	if ( pGun->ubMagSize == Magazine[Item[pAmmo->usItem].ubClassIndex].ubMagSize )
	{
		// normal situation
		return( AP_RELOAD_GUN );
	}
	else
	{
		// trying to reload with wrong size of magazine
		return( AP_RELOAD_GUN + AP_RELOAD_GUN );
	}
}
```

Tactical\Interface Items.c, void GenerateProsString(…)

```
void GenerateProsString( UINT16 * zItemPros, OBJECTTYPE * pObject, UINT32 uiPixLimit )
{
	UINT32		uiStringLength = 0;
	UINT16 *	zTemp;
	UINT16		usItem = pObject->usItem;
	UINT8			ubWeight;

[...]

	if (Weapon[usItem].ubMagSize > EXCEPTIONAL_MAGAZINE)
	if (pObject->ubMagSize > EXCEPTIONAL_MAGAZINE)
	{
		zTemp = Message[STR_LARGE_AMMO_CAPACITY];
		if ( ! AttemptToAddSubstring( zItemPros, zTemp, &uiStringLength, uiPixLimit ) )
		{
			return;
		}
	}

[...]
```

Tactical\Interface Items.c, void GenerateConsString(…)

```
void GenerateConsString( UINT16 * zItemCons, OBJECTTYPE * pObject, UINT32 uiPixLimit )
{
        UINT32                  uiStringLength = 0;
        UINT16 *                zTemp;
        UINT8                           ubWeight;
        UINT16                  usItem = pObject->usItem;

[...]

        if (Weapon[usItem].ubMagSize < BAD_MAGAZINE)
        if (pObject->ubMagSize < BAD_MAGAZINE)
        {
                zTemp = Message[STR_SMALL_AMMO_CAPACITY];
                if ( ! AttemptToAddSubstring( zItemCons, zTemp, &uiStringLength, uiPixLimit ) )
                {
                        return;
                }
        }

[...]
```

Tactical\Interface Items.c, BOOLEAN InternalInitItemDescriptionBox(…)

```
BOOLEAN InternalInitItemDescriptionBox( OBJECTTYPE *pObject, INT16 sX, INT16 sY, UINT8
ubStatusIndex, SOLDIERTYPE *pSoldier )
{
        VOBJECT_DESC    VObjectDesc;
        UINT8 ubString[48];
        INT32           cnt;
        INT16           pStr[10];
        UINT16  usX, usY;
        INT16           sForeColour;
        INT16 sProsConsIndent;

[...]

if ( (Item[ pObject->usItem ].usItemClass & IC_GUN) && pObject->usItem != ROCKET_LAUNCHER )
{
        // Add button
        //    if( guiCurrentScreen != MAP_SCREEN )
        //if( guiCurrentItemDescriptionScreen != MAP_SCREEN )
         swprintf( pStr, L"%d/%d", gpItemDescObject->ubGunShotsLeft, Weapon[
gpItemDescObject->usItem ].ubMagSize );
         swprintf( pStr, L"%d/%d", gpItemDescObject->ubGunShotsLeft, gpItemDescObject-
>ubMagSize );
         FilenameForBPP("INTERFACE\\infobox.sti", ubString);
         sForeColour = ITEMDESC_AMMO_FORE;

[...]
```

Wenden wir uns nun der Items.c zu.

Tactical\Items.c, BOOLEAN ReloadGun(…)

```
BOOLEAN ReloadGun( SOLDIERTYPE * pSoldier, OBJECTTYPE * pGun, OBJECTTYPE * pAmmo )
{
[...]

else
	{
		fEmptyGun = (pGun->ubGunShotsLeft == 0);
		fReloadingWithStack = (pAmmo->ubNumberOfObjects > 1);
		fSameAmmoType = ( pGun->ubGunAmmoType == Magazine[Item[pAmmo->usItem].ubClassIndex].ubAmmoType );
		fSameMagazineSize = ( Magazine[ Item[ pAmmo->usItem ].ubClassIndex ].ubMagSize == Weapon[pGun->usItem].ubMagSize );
		fSameMagazineSize = ( Magazine[ Item[ pAmmo->usItem ].ubClassIndex ].ubMagSize == pGun->ubMagSize );

[...]

if (fSameMagazineSize)
{
	// record new ammo item for gun
	usNewAmmoItem = pAmmo->usItem;
	if (bReloadType == RELOAD_TOPOFF)
	{
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], Weapon[pGun->usItem].ubMagSize - pGun->ubGunShotsLeft );
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], pGun->ubMagSize - pGun->ubGunShotsLeft );
	}
	else
	{
		ubBulletsToMove = pAmmo->ubShotsLeft[0];
	}
}
else if (Magazine[Item[pAmmo->usItem].ubClassIndex].ubMagSize > Weapon[pGun->usItem].ubMagSize)
else if (Magazine[Item[pAmmo->usItem].ubClassIndex].ubMagSize > pGun->ubMagSize)
{
	usNewAmmoItem = pAmmo->usItem - 1;
	if (bReloadType == RELOAD_TOPOFF)
	{
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], Weapon[pGun->usItem].ubMagSize - pGun->ubGunShotsLeft );
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], pGun->ubMagSize - pGun->ubGunShotsLeft );
	}
	else
	{
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], Weapon[pGun->usItem].ubMagSize );
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], pGun->ubMagSize );
	}
}
else // mag is smaller than weapon mag
{
	usNewAmmoItem = pAmmo->usItem + 1;
	if (bReloadType == RELOAD_TOPOFF)
	{
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], Weapon[pGun->usItem].ubMagSize - pGun->ubGunShotsLeft );
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], pGun->ubMagSize - pGun->ubGunShotsLeft );
	}
	else
	{
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], Weapon[pGun->usItem].ubMagSize );
		ubBulletsToMove = __min( pAmmo->ubShotsLeft[0], pGun->ubMagSize );
	}
}

[...]
```

Tactical\Items.c, INT8 FindAmmoToReload(…)

```
INT8 FindAmmoToReload( SOLDIERTYPE * pSoldier, INT8 bWeaponIn, INT8 bExcludeSlot )
{

[…]

// look for any ammo that matches which is of the same calibre and magazine size
                bSlot = FindAmmo( pSoldier, Weapon[pObj->usItem].ubCalibre, Weapon[pObj-
>usItem].ubMagSize, bExcludeSlot );
                bSlot = FindAmmo( pSoldier, Weapon[pObj->usItem].ubCalibre, pObj-
>ubMagSize, bExcludeSlot );

[…]
```

Bis jetzt haben wir nur bestehende Bezugnahmen auf das alte Munitionssystem im Sinne unserer neuen Eigenschaft in der OBJECTTYPE-Klasse überarbeitet. Jetzt müssen wir erreichen, dass allen Waffen auch ein sinnvoller Wert für diese Eigenschaft zugewiesen wird. Wir bedienen uns daher bei der Funktion CreateGun(…) und ergänzen wie folgt:

Tactical\Items.c, BOOLEAN CreateGun(…)

```
BOOLEAN CreateGun( UINT16 usItem, INT8 bStatus, OBJECTTYPE * pObj )
{
        UINT16 usAmmo;


        Assert( pObj != NULL);
        if ( pObj == NULL )
        {
                return( FALSE );
        }

        memset( pObj, 0, sizeof( OBJECTTYPE ) );
        pObj->usItem = usItem;
        pObj->ubNumberOfObjects = 1;
        pObj->bGunStatus = bStatus;
        pObj->ubImprintID = NO_PROFILE;
        pObj->ubWeight = CalculateObjectWeight( pObj );

        pObj->ubMagSize = Weapon[ usItem ].ubMagSize;

[…]
```

Nachdem wir nun die Grundvorraussetzungen geschaffen haben, können wir uns dem Wichtigsten zuwenden.
Es geht uns darum, dass die Magazingröße einer Waffe durch ein Attachment erweitert wird. Zu diesem Zweck ist äußerst sinnvoll, die beiden Funktionen zu bearbeiten, die das Attachen und De-Attachen handhaben. Um ein wenig mehr Realitätsnähe einzubringen, wollen wir das Anbringen bzw. Abmachen des betreffenden Items nur gestatten, wenn nicht bereits ein Magazin eingelegt ist, d.h. wenn die Zahl der verblieben Schüsse einer Waffe 0 beträgt. In diesem Tutorial wollen wir die Magazingröße verdoppeln bzw. halbieren, wenn das Attachment hinzugefügt bzw. entfernt wird. Fügen wir daher in AttachObject(…) und RemoveAttachment(…) Code hinzu:

Tactical\Items.c, BOOLEAN AttachObject(…)

```
BOOLEAN AttachObject( SOLDIERTYPE * pSoldier, OBJECTTYPE * pTargetObj, OBJECTTYPE * pAttachment )
{

[…]

if (bAttachPos == ITEM_NOT_FOUND)
{
        return( FALSE );
}
else
{
        if ( pSoldier )
        {
                bAttachInfoIndex = GetAttachmentInfoIndex( pAttachment->usItem );
                // in-game (not behind the scenes) attachment
                if ( bAttachInfoIndex != -1 && AttachmentInfo[ bAttachInfoIndex ].bAttachmentSkillCheck
!= NO_CHECK )
                {
                        iCheckResult = SkillCheck( pSoldier, AttachmentInfo[ bAttachInfoIndex
].bAttachmentSkillCheck, AttachmentInfo[ bAttachInfoIndex ].bAttachmentSkillCheckMod );
                        if (iCheckResult < 0)
                        {
                                // the attach failure damages both items
                                DamageObj( pTargetObj, (INT8) -iCheckResult );
                                DamageObj( pAttachment, (INT8) -iCheckResult );
                                // there should be a quote here!
                                DoMercBattleSound( pSoldier, BATTLE_SOUND_CURSE1 );
                                if ( gfInItemDescBox )
                                {
                                        DeleteItemDescriptionBox();
                                }
                                return( FALSE );
                        }
                }


        if ( pAttachment->usItem == WALKMAN )
        {
                if ( pTargetObj->ubGunShotsLeft == 0 && FindAttachment( pTargetObj,
WALKMAN ) == ITEM_NOT_FOUND )
                {
                        pTargetObj->ubMagSize = pTargetObj->ubMagSize * 2;
                }
                else
                {
                        return( FALSE );
                }
        }

[…]
```

Tactical\Items.c, BOOLEAN RemoveAttachment[…]

```
BOOLEAN RemoveAttachment( OBJECTTYPE * pObj, INT8 bAttachPos, OBJECTTYPE * pNewObj )
{

[...]

        if ( Item[ pObj->usAttachItem[bAttachPos] ].fFlags & ITEM_INSEPARABLE )
        {
                return( FALSE );
        }

        if ( pObj->usAttachItem[bAttachPos] == WALKMAN )
        {
                if ( pObj->ubGunShotsLeft == 0 )
                {
                        pObj->ubMagSize = pObj->ubMagSize / 2;
                }
                else
                {
                        return( FALSE );
                }
        }
```

In diesem Tutorial wird als Attachment, das für die Munitionskapazitätserweiterung eingesetzt wird, der Walkman verwendet. Alternativ kann auch jedes andere bestehende Item oder auch ein neues eingesetzt werden.
Was nun noch fehlt, ist die Möglichkeit jenes Item an eine Waffe anzubringen. Dazu verwenden wir das Attachment-Array.

Tactical\Items.c, UNIT16 Attachment[][2]

```
UINT16 Attachment[][2] =
{

[...]

        {WALKMAN,    GLOCK_18},
        {0, 0}
};
```

Worauf man noch achten sollte, ist dass es ein passendes Magazin erweiterter Größe gibt. Im Beispiel wurde die Glock 18 gewählt, da es zum 15-schüssigen 9mm Magazin standardmäßig auch ein 30-schüssiges 9mm Magazin gibt. Bei Bedarf kann man selbstverständlich eigene Magazine hinzufügen und verwenden.

Revised release
October, 11th, 2004