Molotowcocktail

Ziel
Dieses Tutorial beschreibt, wie man mit Hilfe des Sourcecodes in Jagged Alliance 2 einen Molotowcocktail einfügt. Dieser Molotowcocktail soll wie eine Granate benutzt werden, eine kleine Explosion erzeugen und sich schließlich in Flammen ausbreiten. Außerdem soll er Schaden an Panzern anrichten können.
Um einen Molotowcocktail herstellen zu können, soll mit einem Messer ein T-Shirt in Stoffstreifen zerteilt werden können, die mit einer Flasche Alkohol kombiniert den Molotowcocktail ergeben.

Vorüberlegung
Im Spiel gibt es bereits einen unbenutzten Gegenstand, den „Creature Cocktail“. Dieser lässt sich umfunktionieren, sodass er als Ausgangsbasis verwendet werden kann. Das Flammeninferno simulieren wir durch eine Gaswolke, wie sie beispielsweise bei der Senfgasgranate entsteht. Allerdings brauchen wir ein „Schadensmodell“ für den Molotowcocktail, damit das Programm unterscheiden kann zwischen Molotowcocktail und Gasgranaten.

Umsetzung
Wie oben erwähnt soll der bereits vorhandene „Creature Cocktail“ zu Molotowcocktail umfunktioniert werden. Passen wir ihn also an:

Gelb unterlegte Zeilen werden durch die rot gedruckten ersetzt.

*Tactical\Items.c, INVTYPE Item[MAXITEMS]*

```
{ IC_GRENADE, 20, TOSSCURS, 0, 1, 98, 6, 4, 50,4, /* smoke hand grenade */ 0, -2, ITEM_DAMAGEABLE |
   ITEM_METAL | ITEM_REPAIRABLE},
{ IC_BOMB, 21, INVALIDCURS, 0, 1, 40, 41, 8, 450, 0, /* tank shell       */ 0, -4, ITEM_DAMAGEABLE |
   ITEM_METAL | ITEM_NOT_BUYABLE | ITEM_NOT_EDITOR },
{ IC_BOMB, 22, INVALIDCURS, 0, 1, 40, 41, 2, 450, 0, /* fake struct ignite*/ 0, -4, ITEM_DAMAGEABLE |
   ITEM_METAL | ITEM_NOT_BUYABLE },
{ IC_GRENADE, 23, TOSSCURS, 0, 2, 37, 6, 4, 50, 0, /* creature cocktail*/ 0, 0, ITEM_DAMAGEABLE |
   ITEM_METAL },
{ IC_GRENADE, 23, TOSSCURS, 0, 2, 37, 6, 4, 50, 0, /* creature cocktail*/ 0, -4, ITEM_DAMAGEABLE |
   ITEM_REPAIRABLE},
{ IC_BOMB, 24, INVALIDCURS, 0, 1, 40, 41, 2, 450, 0, /* fake struct xplod*/ 0, -4, ITEM_DAMAGEABLE |
   ITEM_METAL | ITEM_NOT_BUYABLE },
```

Weiterhin soll das “Deidranna rulez“-T-Shirt als Stoffsteifen dienen, von denen man vier erhält, wenn man das „I love Arulco“-T-Shirt mit dem Kampfmesser bearbeitet.

*Tactical\Items.c, INVTYPE Item[MAXITEMS]*

```
{ IC_ARMOUR, 32, INVALIDCURS, COND, 1, 0, 0, 0, 0, 0, /* old fem hide */ 0, 0, IF_STANDARD_ARMOUR |
   ITEM_NOT_BUYABLE | ITEM_NOT_EDITOR | ITEM_DEFAULT_UNDROPPABLE},
{ IC_ARMOUR, 33, INVALIDCURS, COND, 2, 25, 3, 1, 10, 1, /* t-shirt */ 0, 0, ITEM_DAMAGEABLE |
   ITEM_SHOW_STATUS | ITEM_UNAERODYNAMIC},
{ IC_ARMOUR, 33, INVALIDCURS, COND, 2, 34, 3, 1, 10, 1, /* t-shirt D. rules*/ 0, 0, ITEM_DAMAGEABLE |
   ITEM_SHOW_STATUS | ITEM_UNAERODYNAMIC},
{ IC_MISC, 33, INVALIDCURS, COND, 2, 34, 1, 8, 10, 1, /* t-shirt D. rules*/ 0, 0, ITEM_NOT_BUYABLE |
   ITEM_UNAERODYNAMIC | ITEM_HIDDEN_ADDON},
{ IC_ARMOUR, 34, INVALIDCURS, COND, 1, 137, 32, 0, 700, 6, /* Kevlar2 jacket  */ 0, -1,
   IF_STANDARD_ARMOUR},
```

Nun, da wir die nötigen Items haben, schauen wir, dass wir Attachments Messer an T-Shirt (für die Stoffstreifen) und Stoffstreifen an Alkoholflasche einfügen.

*Tactical\Items.c, UINT16 Attachment[][2]*

```
   {BATTERIES,              XRAY_DEVICE},
   {COPPER_WIRE,                  LAME_BOY},
   {COMBAT_KNIFE,                 TSHIRT},
   {TSHIRT_DEIDRANNA,       ALCOHOL},
```

Jetzt fehlt noch die Kombination von Alkohol + Stoffstreifen => Molotowcocktail. Wir ergänzen folgende Zeile.

*Tactical\Items.c, ComboMergeInfoStruct AttachmentComboMerge[]*

```
   // base item       attach 1                attach 2          result
   {ALUMINUM_ROD,           {SPRING,                NOTHING},         SPRING_AND_BOLT_UPGRADE },
   {STEEL_ROD,        {QUICK_GLUE,            DUCT_TAPE},       GUN_BARREL_EXTENDER },
   {FUMBLE_PAK,       {XRAY_BULB,             CHEWING_GUM},     FLASH_DEVICE },
   {LAME_BOY,         {COPPER_WIRE,           NOTHING},         DISPLAY_UNIT },
   {ALCOHOL,                {TSHIRT_DEIDRANNA,      NOTHING},         CREATURE_COCKTAIL },
   {NOTHING,                {NOTHING,               NOTHING},         NOTHING },
```

Bisher würde man jedoch nur ein „I love Arulco“-T-Shirt mit angebrachtem Messer erhalten, anstatt der vier Stoffstreifen. Hat man jedoch einen solchen Stoffstreifen, kann man mit diesem bereits den Molotowcocktail („Creature Cocktail“) herstellen, indem man jenen an eine Flasche Alkohol anbringt. Also müssen wir es einrichten, mit dem Messer aus dem T-Shirt die vier Stoffstreifen erzeugen zu können, wie es beabsichtigt ist.
Außerdem wollen wir das Messer behalten, und auch die Tatsache, dass wir gleich mehrere Stoffstreifen erzeugen wollen, macht es unmöglich, die standardmäßige Kombinationsfunktion zu verwenden.
Um dies Problem zu lösen, verändern wir die Funktion, die für das Attachen von Items zuständig ist.

*Tactical\Items.c, BOOLEAN AttachObject( SOLDIERTYPE * pSoldier, OBJECTTYPE * pTargetObj, OBJECTTYPE * pAttachment )*

```
INT8        bAttachPos, bSecondAttachPos;//, bAbility, bSuccess;
UINT16 usResult;
INT8        bLoop;
UINT8       ubType, ubLimit;
INT32       iCheckResult;
INT8        bAttachInfoIndex = -1, bAttachComboMerge;
BOOLEANfValidLaunchable = FALSE;

if ( pTargetObj->usItem == TSHIRT && pAttachment->usItem == COMBAT_KNIFE)
{
 pTargetObj->usItem = TSHIRT_DEIDRANNA;
 pTargetObj->ubNumberOfObjects = 4;
 pTargetObj->bStatus[0] = 100;
 pTargetObj->bStatus[1] = 100;
 pTargetObj->bStatus[2] = 100;
 pTargetObj->bStatus[3] = 100;
 return (FALSE);
}

fValidLaunchable = ValidLaunchable( pAttachment->usItem, pTargetObj->usItem );
```

Bevor der eigentliche Programmcode der Funktion zum Einsatz kommt, prüfen wir, ob das zu attachende Objekt das Messer ist, und ob es sich bei dem Item, an das attacht wird, um das „I love Arulco“-T-Shirt handelt. Wenn dem so ist, machen wir aus dem T-Shirt das Stoffstreifenobjekt (hier das „Deidranna rulez“-T-Shirt). Aus eins mach vier – verändern wir einfach die Anzahl der Objekte und schon haben wir vier Stoffstreifen – zumindest theoretisch. Damit dem auch so ist, setzen wir die Zustandswerte der Stoffstreifen auf jeweils 100 % und schon sind aus dem T-Shirt vier Stoffstreifen geworden. Damit wir das Messer behalten und dieses nicht attacht wird, brechen wir die Funktion ab.
Nun ist es möglich aus den vorhandenen Items den Molotowcocktail herzustellen. Doch dieser ist natürlich noch nicht einsatzbereit.

Passen wir zuerst die Eigenschaften des Items „Creature Cocktail“ an:

*Tactical\Weapons.c, EXPLOSIVETYPE Explosive[]*

```
{ EXPLOSV_NORMAL, 60, 20, 6, 60, 2, BLAST_2, /* Tank Shell         */},
{ EXPLOSV_NORMAL, 100, 0, 0, 0, 0, BLAST_1, /* Fake structure igniter*/},
{ EXPLOSV_NORMAL, 100, 0, 1, 0, 0, BLAST_1, /* creature cocktail */},
{ EXPLOSV_MOLOTOW, 15, 30, 4, 15, 0, BLAST_1, /* creature cocktail */},
{ EXPLOSV_NORMAL, 50, 10, 5, 50, 2, BLAST_2, /* fake struct explosion*/},
```

Wie sich unschwer erkennen lässt, wurde für den Molotowcocktail ein eigener Explosivtyp verwendet. Dieser muss jedoch vorher angelegt werden.

*Tactical\Weapons.h*

```
enum
{
   EXPLOSV_NORMAL,
   EXPLOSV_STUN,
   EXPLOSV_TEARGAS,
   EXPLOSV_MUSTGAS,
   EXPLOSV_FLARE,
   EXPLOSV_NOISE,
   EXPLOSV_SMOKE,
   EXPLOSV_CREATUREGAS,
   EXPLOSV_MOLOTOW,
};
```

Bis jetzt waren die notwendigen Änderungen recht übersichtlich. Um den Molotowcocktail völlig funktionsfähig zu machen, sind einige weitere Änderungen an verschiedenen Stellen notwendig.
Da der Molotowcocktail wie eine Gasgranate behandelt wird, ist es nötig ein eigenes Flag für die Rauchwolken zu setzen.

*TileEngine\worlddef.h*

```
#define MAPELEMENT_EXT_SMOKE                                    0x01
#define MAPELEMENT_EXT_TEARGAS                                  0x02
#define MAPELEMENT_EXT_MUSTARDGAS                               0x04
#define MAPELEMENT_EXT_DOOR_STATUS_PRESENT   0x08
#define MAPELEMENT_EXT_RECALCULATE_MOVEMENT 0x10
#define MAPELEMENT_EXT_NOBURN_STRUCT                    0x20
#define MAPELEMENT_EXT_ROOFCODE_VISITED                 0x40
#define MAPELEMENT_EXT_CREATUREGAS                      0x80
#define MAPELEMENT_EXT_MOLOTOW                                  0x100


#define FIRST_LEVEL 0
#define SECOND_LEVEL 1


#define ANY_SMOKE_EFFECT          ( MAPELEMENT_EXT_CREATUREGAS | MAPELEMENT_EXT_SMOKE |
   MAPELEMENT_EXT_TEARGAS | MAPELEMENT_EXT_MUSTARDGAS )
#define ANY_SMOKE_EFFECT          ( MAPELEMENT_EXT_CREATUREGAS | MAPELEMENT_EXT_SMOKE |
   MAPELEMENT_EXT_TEARGAS | MAPELEMENT_EXT_MUSTARDGAS | MAPELEMENT_EXT_MOLOTOW )
```

Nun ist es angebracht, erwähntes „Schadensmodell“ für den Molotowcocktail einzurichten.

*Tactical\Soldier Control.h*

```
#define TAKE_DAMAGE_GUNFIRE                  1
#define TAKE_DAMAGE_BLADE                    2
#define TAKE_DAMAGE_HANDTOHAND               3
#define TAKE_DAMAGE_FALLROOF                 4
#define TAKE_DAMAGE_BLOODLOSS                5
#define TAKE_DAMAGE_EXPLOSION                6
#define TAKE_DAMAGE_ELECTRICITY              7
#define TAKE_DAMAGE_GAS                      8
#define TAKE_DAMAGE_TENTACLES                9
#define TAKE_DAMAGE_STRUCTURE_EXPLOSION     10
#define TAKE_DAMAGE_OBJECT                  11
#define TAKE_DAMAGE_MOLOTOW                 12
```

*Tactical\Soldier Control.h*

```
enum
{
   HIT_BY_TEARGAS = 0x01,
   HIT_BY_MUSTARDGAS = 0x02,
   HIT_BY_CREATUREGAS = 0x04,
   HIT_BY_MOLOTOW = 0x08,
};
```

Damit haben wir einen weiteren Schadensgrund und ein Söldnerflag für Gasangriffe hinzugefügt, womit wir weiter arbeiten können. Zuerst jedoch müssen wir ein paar Routinen für den Rauch von Granaten anpassen, aber auch hier zuerst ein neues Typus:

*TileEngine\SmokeEffects.h*

```
enum
{
   NO_SMOKE_EFFECT,
   NORMAL_SMOKE_EFFECT,
   TEARGAS_SMOKE_EFFECT,
   MUSTARDGAS_SMOKE_EFFECT,
   CREATURE_SMOKE_EFFECT,
   MOLOTOW_SMOKE_EFFECT,
};
```

*TileEngine\SmokeEffects.c, INT8 FromWorldFlagsToSmokeType( UINT8 ubWorldFlags )*

```
INT8 FromWorldFlagsToSmokeType( UINT8 ubWorldFlags )
{
   if ( ubWorldFlags & MAPELEMENT_EXT_SMOKE )
   {
    return( NORMAL_SMOKE_EFFECT );
   }
   else if ( ubWorldFlags & MAPELEMENT_EXT_TEARGAS )
   {
    return( TEARGAS_SMOKE_EFFECT );
   }
   else if ( ubWorldFlags & MAPELEMENT_EXT_MUSTARDGAS )
   {
    return( MUSTARDGAS_SMOKE_EFFECT );
   }
   else if ( ubWorldFlags & MAPELEMENT_EXT_CREATUREGAS )
   {
    return( CREATURE_SMOKE_EFFECT );
   }
   else if ( ubWorldFlags & MAPELEMENT_EXT_MOLOTOW )
   {
    return( MOLOTOW_SMOKE_EFFECT );
   }
   else
   {
    return( NO_SMOKE_EFFECT );
   }
}
```

*TileEngine\SmokeEffects.c, UINT8 FromSmokeTypeToWorldFlags( INT8 bType )*

```
    case MUSTARDGAS_SMOKE_EFFECT:
          return( MAPELEMENT_EXT_MUSTARDGAS );
          break;
     case CREATURE_SMOKE_EFFECT:
          return( MAPELEMENT_EXT_CREATUREGAS );
          break;
    case MOLOTOW_SMOKE_EFFECT:
          return( MAPELEMENT_EXT_MOLOTOW );
          break;
    default:
          return( 0 );
```

*TileEngine\SmokeEffects.c, INT32 NewSmokeEffect( INT16 sGridNo, UINT16 usItem, INT8 bLevel, UINT8 ubOwner )*

```
 case SMOKE_GRENADE:
 case GL_SMOKE_GRENADE:
       bSmokeEffectType  =     NORMAL_SMOKE_EFFECT;
       ubDuration                    = 5;
       ubStartRadius                 = 1;
       break;

 case CREATURE_COCKTAIL:
       bSmokeEffectType  =     MOLOTOW_SMOKE_EFFECT;
       ubDuration                    = 5;
       ubStartRadius                 = 1;
       break;

  case SMALL_CREATURE_GAS:
       bSmokeEffectType  =     CREATURE_SMOKE_EFFECT;
       ubDuration                    = 3;
       ubStartRadius                 = 1;
       break;
```

*TileEngine\SmokeEffects.c, void AddSmokeEffectToTile( INT32 iSmokeEffectID, INT8 bType, INT16 sGridNo, INT8 bLevel )*

```
 case MUSTARDGAS_SMOKE_EFFECT:
 case MOLOTOW_SMOKE_EFFECT:

    if ( !( gGameSettings.fOptions[ TOPTION_ANIMATE_SMOKE ] ) )
    {
          strcpy( AniParams.zCachedFile, "TILECACHE\\mustchze.STI" );
    }
    else
    {
      if ( fDissipating )
      {
           strcpy( AniParams.zCachedFile, "TILECACHE\\smalmust.STI" );
      }
      else
      {
           strcpy( AniParams.zCachedFile, "TILECACHE\\MUSTARD2.STI" );
      }
    }
       break;
```

An dieser Stelle wird festgelegt, welche .sti-Datei für die Rauchwolken verwendet wird. Im Beispiel wird das der Senfgasgranate genutzt, alternativ kann aber auch eine andere Datei nach gleichem Schema verwendet werden.

Jetzt wird es endlich interessant, denn die lästigen Vorbereitungen sind getroffen, es folgt der Teil, von dem man im Spiel auch was merkt.

*TileEngine\Explosion Control.c, BOOLEAN DishOutGasDamage( SOLDIERTYPE * pSoldier, EXPLOSIVETYPE * pExplosive, INT16 sSubsequent, BOOLEAN fRecompileMovementCosts, INT16 sWoundAmt, INT16 sBreathAmt, UINT8 ubOwner )*

```
 else if ( pExplosive->ubType == EXPLOSV_MUSTGAS )
 {
  if ( AM_A_ROBOT( pSoldier ) )
  {
       return( fRecompileMovementCosts );
  }

  if ( sSubsequent && pSoldier->fHitByGasFlags & HIT_BY_MUSTARDGAS )
  {
       // already affected by creature gas this turn
       return( fRecompileMovementCosts );
  }

 }
 else if ( pExplosive->ubType == EXPLOSV_MOLOTOW )
 {
       if ( sSubsequent && pSoldier->fHitByGasFlags & HIT_BY_MOLOTOW )
       {
       return( fRecompileMovementCosts );
       }
 }

 if ( pSoldier->inv[ HEAD1POS ].usItem == GASMASK && pSoldier->inv[ HEAD1POS ].bStatus[0] >= USABLE )
 {
       bPosOfMask = HEAD1POS;
 }
 else if ( pSoldier->inv[ HEAD2POS ].usItem == GASMASK && pSoldier->inv[ HEAD2POS ].bStatus[0] >=
USABLE )
 {
       bPosOfMask = HEAD2POS;
 }

 if ( bPosOfMask != NO_SLOT  )
 if ( bPosOfMask != NO_SLOT && pExplosive->ubType != EXPLOSV_MOLOTOW )
```

```
    {
     if ( pSoldier->inv[ bPosOfMask ].bStatus[0] < GASMASK_MIN_STATUS )
     {
           // GAS MASK reduces breath loss by its work% (it leaks if not at least 70%)
           sBreathAmt = ( sBreathAmt * ( 100 - pSoldier->inv[ bPosOfMask ].bStatus[0] ) ) / 100;
           if ( sBreathAmt > 500 )
           {
                     // if at least 500 of breath damage got through
                     // the soldier within the blast radius is gassed for at least one
                     // turn, possibly more if it's tear gas (which hangs around a while)
                     pSoldier->uiStatusFlags |= SOLDIER_GASSED;
           }

           if ( pSoldier->uiStatusFlags & SOLDIER_PC )
           {

                 if ( sWoundAmt > 1 )
                 {
                      pSoldier->inv[ bPosOfMask ].bStatus[0] -= (INT8) Random( 4 );
                      sWoundAmt = ( sWoundAmt * ( 100 -  pSoldier->inv[ bPosOfMask ].bStatus[0] ) ) / 100;
                 }
                 else if ( sWoundAmt == 1 )
                 {
                      pSoldier->inv[ bPosOfMask ].bStatus[0] -= (INT8) Random( 2 );
                 }
           }
     }
     else
     {
          sBreathAmt = 0;
          if ( sWoundAmt > 0 )
          {
           if ( sWoundAmt == 1 )
           {
                pSoldier->inv[ bPosOfMask ].bStatus[0] -= (INT8) Random( 2 );
           }
           else
           {
                // use up gas mask
                pSoldier->inv[ bPosOfMask ].bStatus[0] -= (INT8) Random( 4 );
           }
          }
          sWoundAmt = 0;
```

```
    }

  }
 }

 if ( sWoundAmt != 0 || sBreathAmt != 0 )
 {
  switch( pExplosive->ubType )
  {
        case EXPLOSV_CREATUREGAS:
              pSoldier->fHitByGasFlags |= HIT_BY_CREATUREGAS;
              break;
        case EXPLOSV_TEARGAS:
              pSoldier->fHitByGasFlags |= HIT_BY_TEARGAS;
              break;
        case EXPLOSV_MUSTGAS:
              pSoldier->fHitByGasFlags |= HIT_BY_MUSTARDGAS;
              break;
        case EXPLOSV_MOLOTOW:
              pSoldier->fHitByGasFlags |= HIT_BY_MOLOTOW;
              break;
        default:
              break;
  }
  // a gas effect, take damage directly...
  SoldierTakeDamage( pSoldier, ANIM_STAND, sWoundAmt, sBreathAmt, TAKE_DAMAGE_GAS, NOBODY, NOWHERE, 0,
 TRUE );
  if ( pExplosive->ubType == EXPLOSV_MOLOTOW )
  {
        SoldierTakeDamage( pSoldier, ANIM_STAND, sWoundAmt, sBreathAmt, TAKE_DAMAGE_MOLOTOW, NOBODY,
 NOWHERE, 0, TRUE );
  }
  else
  {
        SoldierTakeDamage( pSoldier, ANIM_STAND, sWoundAmt, sBreathAmt, TAKE_DAMAGE_GAS, NOBODY,
 NOWHERE, 0, TRUE );
  }
  if ( pSoldier->bLife >= CONSCIOUSNESS )
  {
        DoMercBattleSound( pSoldier, (INT8)( BATTLE_SOUND_HIT1 + Random( 2 ) ) );
  }
```

```
    if ( ubOwner != NOBODY && MercPtrs[ ubOwner ]->bTeam == gbPlayerNum && pSoldier->bTeam != gbPlayerNum
  )
    {
     ProcessImplicationsOfPCAttack( MercPtrs[ ubOwner ], &pSoldier, REASON_EXPLOSION );
    }
  }
  return( fRecompileMovementCosts );
```

In diesem umfangreicherem Auszug wird folgendes gemacht: Zuerst kommt unser oben geschaffenes „HIT_BY_MOLOTOW“-Flag zum Einsatz, wenn es sich um die Auswirkungseffekte des Molotowcocktails handelt. Die zweite Änderung verhindert, dass das Tragen einer Gasmaske gegen die Verbrennungen durch die Flammen schützt. Als drittes wird dem betreffenden Soldaten das Flag aus erstens zugewiesen, bevor zuletzt unterschieden wird, ob die Schadensfunktion durch normales Gas oder den Molotowcocktail aufgerufen wird.

Für die Flammen des Molotowcocktails müssen allerdings noch zwei andere Funktionen überarbeitet werden, die das Ausbreiten der Gaswolken kontrollieren. Auch hier erfolgt eigentlich nur eine Anpassung auf den Molotowcocktail. Die vordefinierten Strukturen können übernommen werden.

*TileEngine\Explosion Control.c, BOOLEAN ExpAffect( INT16 sBombGridNo, INT16 sGridNo, UINT32 uiDist, UINT16 usItem, UINT8 ubOwner,  INT16 sSubsequent, BOOLEAN *pfMercHit, INT8 bLevel, INT32 iSmokeEffectID )*

```
              case STUN_GRENADE:
              case GL_STUN_GRENADE:
                    fStunEffect                 = TRUE;
                    break;

              case CREATURE_COCKTAIL:
                    fSmokeEffect                = TRUE;
                    bSmokeEffectType  =     MOLOTOW_SMOKE_EFFECT;
                    fBlastEffect                = FALSE;
                    break;

       case SMALL_CREATURE_GAS:
       case LARGE_CREATURE_GAS:
       case VERY_SMALL_CREATURE_GAS:

                    fSmokeEffect                = TRUE;
                    bSmokeEffectType  =     CREATURE_SMOKE_EFFECT;
                    fBlastEffect                = FALSE;
                    break;
```

*TileEngine\Explosion Control.c, void SpreadEffect( INT16 sGridNo, UINT8 ubRadius, UINT16 usItem, UINT8 ubOwner, BOOLEAN fSubsequent, INT8 bLevel, INT32 iSmokeEffectID  )*

```
switch( usItem )
 {
    case MUSTARD_GRENADE:
    case TEARGAS_GRENADE:
    case GL_TEARGAS_GRENADE:
    case BIG_TEAR_GAS:
    case SMOKE_GRENADE:
    case GL_SMOKE_GRENADE:
     case SMALL_CREATURE_GAS:
     case LARGE_CREATURE_GAS:
     case VERY_SMALL_CREATURE_GAS:
    case CREATURE_COCKTAIL:

       fSmokeEffect = TRUE;
       break;
 }
```

Indem wir einen eigenen Schadenstyp kreiert haben, ist es möglich auch an Fahrzeugen, also Panzern, Schaden anzurichten.

*Tactical\Verhicles.c, void VehicleTakeDamage( UINT8 ubID, UINT8 ubReason, INT16 sDamage, INT16 sGridNo, UINT8 ubAttackerID )*

```
   // check if there was in fact damage done to the vehicle
   if( ( ubReason == TAKE_DAMAGE_HANDTOHAND ) || ( ubReason == TAKE_DAMAGE_GAS ) )
   {
    // nope
    return;
   }
   if( pVehicleList[ ubID ].fDestroyed == FALSE )
   {
    switch( ubReason )
    {
          case( TAKE_DAMAGE_GUNFIRE ):
          case( TAKE_DAMAGE_EXPLOSION):
          case( TAKE_DAMAGE_STRUCTURE_EXPLOSION):
          case( TAKE_DAMAGE_MOLOTOW ):
                HandleCriticalHitForVehicleInLocation( ubID, sDamage, sGridNo, ubAttackerID );
                break;
    }
   }
```

Dieser Schaden an Fahrzeugen ist allerdings durch das spezielle Schadensmuster der Panzer zu groß, daher verringern wir diesen.

*Tactical\Soldier Control.c, UINT8 SoldierTakeDamage( SOLDIERTYPE *pSoldier, INT8 bHeight, INT16 sLifeDeduct, INT16 sBreathLoss, UINT8 ubReason, UINT8 ubAttacker, INT16 sSourceGrid, INT16 sSubsequent, BOOLEAN fShowDamage )*

```
// OK, If we are a vehicle.... damage vehicle...( people inside... )
if ( pSoldier->uiStatusFlags & SOLDIER_VEHICLE )
{
 if ( TANK( pSoldier ) )
 {
       //sLifeDeduct = (sLifeDeduct * 2) / 3;
 }
 else
 {
       if ( ubReason == TAKE_DAMAGE_GUNFIRE )
       {
             sLifeDeduct /= 3;
       }
       else if ( ubReason == TAKE_DAMAGE_EXPLOSION && sLifeDeduct > 50 )
       {
             // boom!
             sLifeDeduct *= 2;
       }
       else if ( ubReason == TAKE_DAMAGE_MOLOTOW )
       {
             sLifeDeduct /= 4;
       }
 }

 VehicleTakeDamage( pSoldier->bVehicleID, ubReason, sLifeDeduct, pSoldier->sGridNo, ubAttacker );
 HandleTakeDamageDeath( pSoldier, bOldLife, ubReason );
 return( 0 );
}
```

Eine wichtige Änderung ist außerdem noch zu tätigen, damit das ganze auch wie gewünscht funktioniert.

*TileEngine\Tile Animation.c, void UpdateAniTiles( )*

```
                if ( ubExpType == EXPLOSV_TEARGAS || ubExpType == EXPLOSV_MUSTGAS ||
                     ubExpType == EXPLOSV_SMOKE )
                if ( ubExpType == EXPLOSV_TEARGAS || ubExpType == EXPLOSV_MUSTGAS ||
                     ubExpType == EXPLOSV_SMOKE || ubExpType == EXPLOSV_MOLOTOW )
```

Die zwei verbleibenden Änderungen sind zwar nötig, aber eher unspektakulärer Natur.

*Tactical\LOS.c, INT32 LineOfSightTest( FLOAT dStartX, FLOAT dStartY, FLOAT dStartZ, FLOAT dEndX, FLOAT dEndY, FLOAT dEndZ, UINT8 ubTileSightLimit, UINT8 ubTreeSightReduction, INT8 bAware, INT8 bCamouflage, BOOLEAN fSmell, INT16 * psWindowGridNo )*

```
 // leaving a tile, check to see if it had gas in it
 if ( pMapElement->ubExtFlags[0] & (MAPELEMENT_EXT_SMOKE | MAPELEMENT_EXT_TEARGAS |
MAPELEMENT_EXT_MUSTARDGAS) )
 if ( pMapElement->ubExtFlags[0] & (MAPELEMENT_EXT_SMOKE | MAPELEMENT_EXT_TEARGAS |
MAPELEMENT_EXT_MUSTARDGAS | MAPELEMENT_EXT_MOLOTOW) )
 {
       if ( (pMapElement->ubExtFlags[0] & MAPELEMENT_EXT_SMOKE) && !fSmell )
       {
             bSmoke++;
```

*Tactical\Overhead.c, BOOLEAN HandleGotoNewGridNo( SOLDIERTYPE *pSoldier, BOOLEAN *pfKeepMoving, BOOLEAN fInitialMove, UINT16 usAnimState )*

```
                    if ( gpWorldLevelData[ pSoldier->sGridNo ].ubExtFlags[pSoldier->bLevel] &
MAPELEMENT_EXT_MUSTARDGAS )
                    {
                          if ( !(pSoldier->fHitByGasFlags & HIT_BY_MUSTARDGAS) && bPosOfMask == NO_SLOT
)
                          {
                                pExplosive = &(Explosive[ Item[ MUSTARD_GRENADE ].ubClassIndex ]);
                          }
                    }
             }
             if ( gpWorldLevelData[ pSoldier->sGridNo ].ubExtFlags[pSoldier->bLevel] &
MAPELEMENT_EXT_MOLOTOW )
             {
                    if ( !(pSoldier->fHitByGasFlags & HIT_BY_MOLOTOW) )
                    {
                          pExplosive = &(Explosive[ Item[ CREATURE_COCKTAIL ].ubClassIndex ]);
                    }
             }
             if ( gpWorldLevelData[ pSoldier->sGridNo ].ubExtFlags[pSoldier->bLevel] &
MAPELEMENT_EXT_CREATUREGAS )
             {
                    if ( !(pSoldier->fHitByGasFlags & HIT_BY_CREATUREGAS) ) // gas mask doesn't help vs
creaturegas
                    {
                          pExplosive = &(Explosive[ Item[ SMALL_CREATURE_GAS ].ubClassIndex ]);
```

Kaum zu glauben, aber das war es. Nun sollte der Molotowcocktail fertig und einsatzbereit sein.

Resümee
Sicherlich kann man an einigen Stellen weitaus eleganter vorgehen, doch dies hätte teilweise Änderungen in noch weitaus umfangreicherem Maße erfordert. Trotzdem sollte dieses Lösung durchaus akzeptabel sein, da sie weiterhin Raum lässt für eventuelle weitergehende Veränderungen.

Schlusswort
Dieses Tutorial ist nach einer Idee von Wulfy301 aus dem JA2 Basis Forum (http://ja-forum.gamigo.de) entstanden. Wulfy301 war es auch, der mit seiner Hilfe entscheidend dazu beigetragen hat, dass dieses Tutorial überhaupt gelungen ist. An dieser Stelle Herzlichen Dank dafür!

December, 29th, 2004
1st edition
© 2004 by Realist
all rights reserved