SE|ED

Settlers Editor

Script Tutorials

SE|ED – Settlers Editor

Script-Tutorials

Inhalt:

1. Generell

Um die hier beschriebenen Tutorials auszuführen müsst ihr den Experten-Modus des Editors aktiviert haben. Bitte bedenkt, dass bei der Arbeit im Expertenmodus durchaus zunächst schwer nachvollziehbare Probleme auftreten können

a) **Exportiert das vorhandene Script:** Wählt im Menü unter Experten-Modus > Exportiere Karten-Script.
b) Passt euer **Script** mit einem für Lua geeigneten Editor an, beispielsweise mit Notepad++.
c) **Importiert das veränderte Karten Script:** Wählt im Menü unter Experten-Modus > Importiere Karten-Script. Ab Patch 1.3 werden hierbei auch bei der Bearbeitung möglicherweise entstandene Syntaxfehler inklusive Zeilennummer angezeigt.

2. Handelspartner & Söldner

Um aus vorhandenen Dörfern, Klöstern oder Banditenlagern **Handelspartner** zu erzeugen die Waren und Söldner anbieten, müssen einige Zeilen in eine bestehende Funktion eingefügt werden.

a) Sucht nach folgender Funktion (die voraussichtlich leer sein wird):

```
function Mission_InitMerchants()

end
```

b) In die InitMerchants Funktion könnt ihr nun die gewünschten Handelspartner und Handelsgüter eintragen. Bitte bedenkt hierbei, dass es **maximal** 4 Handelsgüter pro Handelspartner geben darf:

```
function Mission_InitMerchants()
    local traderId = Logic.GetStoreHouse(AIPlayerID)
    AddOffer( traderId, NumberOfOffers, GoodType )
end
```

Die kursiven Variablennamen müssen durch eigene Werte ersetzt werden:

- *AIPlayerID*: Nummer des KI Spielers der den Handel anbieten soll.
- *NumberOfOffers*: Menge der angebotenen Ware.
- *GoodType*: Art der angebotenen Ware. Die konkreten Waren können der nachfolgenden Liste entnommen werden. Jedem hier eingetragenen Wert muss ein „`Goods.`“ vorangestellt werden.

```
G_Beer G_Bow G_Bread G_Broom G_Candle G_Carcass G_Cheese G_Clothes G_Cow
G_Grain G_Herb G_Honeycomb G_Iron G_Leather G_Medicine G_Milk G_RawFish
G_Sausage G_Sheep G_SmokedFish G_Soap G_Stone G_Sword G_Wood G_Wool
```

Beispiel für einen **Handelspartner** (Dorf mit Lagerhaus) mit der Spieler ID 3, der 2 Holz, 3 Eisen, 5 Kühe und 5 Schafe anbieten soll:

```
function Mission_InitMerchants()
    local traderId = Logic.GetStoreHouse(3)
    AddOffer( traderId, 2, Goods.G_Wood )
    AddOffer( traderId, 3, Goods.G_Iron )
    AddOffer( traderId, 5, Goods.G_Cow )
    AddOffer( traderId, 5, Goods.G_Sheep )
end
```

Neben Handelswaren können auch Söldner angeboten werden. Bitte bedenkt, dass es **maximal** 4 Handelsgüter- **und/oder** Söldner-Angebote pro Handelspartner geben darf.

Es ist recht hilfreich wenn man in einem Dorf, welches Söldner anbietet, auch eine **Kaserne** (z.B. B_NPC_Barracks_ME) im Editor setzt. Diese ist zwar keine zwingende Vorraussetzung damit Söldner angeboten werden können, aber es ist eine gute Visualisierung dafür, dass es hier Söldner gibt. Wenn man eine Kaserne erstellt hat, patrouillieren nun auch Söldner in diesem Dorf (z.B. wenn man eine Palisade mit Toren um das Dorf baut).

Söldner können mit diesem Aufruf hinzugefügt werden:

```
AddMercenaryOffer( traderId, Amount, UnitType)
```

- *Amount*: Anzahl angebotener Söldner.
- *UnitType*: Art der angebotenen Einheiten. Die konkreten Typen können der nachfolgenden Liste entnommen werden. Jedem hier eingetragenen Wert muss ein „Entities.“ vorangestellt werden.

```
U_MilitaryBandit_Melee_ME U_MilitaryBandit_Melee_SE
U_MilitaryBandit_Melee_NA U_MilitaryBandit_Melee_NE
U_MilitaryBandit_Ranged_ME U_MilitaryBandit_Ranged_NA
U_MilitaryBandit_Ranged_NE U_MilitaryBandit_Ranged_SE
```

Beispiel für ein **Banditenlager** (mit BanditsHQ) mit der Spieler ID 3, welches 2 Gruppen Bogenschützen und 3 Gruppen Schwertkämpfer als Söldner anbietet:

```
function Mission_InitMerchants()
    local traderId = Logic.GetStoreHouse(3)
    AddMercenaryOffer( traderId, 2,
                       Entities.U_MilitaryBandit_Melee_ME )
    AddMercenaryOffer( traderId, 3,
                      Entities.U_MilitaryBandit_Ranged_ME )
end
```

Abgesehen von Waren und Söldnern kann man auch Feuerschlucker für den Marktplatz anheuern. Dazu wird folgende Zeile verwendet:

```
AddEntertainerOffer ( traderId, Entities.U_FireEater )
```

3. Weitere Einstellungen für Handelspartner

Normalerweise kann mit jedem Spieler, für den dies vorgesehen ist, gehandelt werden sobald über Diplomatie Handelsbeziehungen erreicht wurden. In einigen Fällen möchte man aber den Handel dennoch zeitweise unterbinden, beispielsweise wenn die Handelsstraße gerade von gefährlichen Wölfen unsicher gemacht wird.

Sowohl der **An- als auch der Verkauf von Waren und Söldnern kann gesteuert werden**. Die entsprechenden Funktionen hierfür können an einer beliebigen Stelle im Script aufgerufen werden an der die Änderung im Handelsverhalten aktiviert werden soll. Damit die entsprechenden Zeilen funktionieren muss immer folgende Zeile vorangestellt werden:

```
local traderId = Logic.GetStoreHouse(AIPlayerID)
```

Warenangebot abschalten:

```
DeActivateMerchantForPlayer( traderId, PlayerID )
```

Warenangebot wieder für den Ritter aktivieren:

```
ActivateMerchantForPlayersKnight( traderId, PlayerID )
```

Warenangebot immer aktivieren:

```
ActivateMerchantPermanentlyForPlayer( traderId, PlayerID )
```

Ankauf von Waren steuern:

```
SetPlayerDoesNotBuyGoodsFlag( AIPlayerID, Active )
```

a) `AIPlayerID`: Nummer des KI Spielers dessen Handelseinstellungen verändert werden. sollen.
b) `PlayerID`: ID des Spielers. Diese ist normalerweise 1.
c) `Active:` Hiermit wird der Ankauf von Waren (de)aktiviert. Mögliche Werte sind `true` (kauft keine Waren) und `false` (kauft Waren).

Beispiel für eine Funktion, die den Handel mit Spieler 2 verhindert:

```
function DeactivateTradeWithPlayer2()
    local traderId = Logic.GetStoreHouse( 2 )
    DeActivateMerchantForPlayer( traderId, 1 )
    SetPlayerDoesNotBuyGoodsFlag( 2, true )
end
```

4. „KI Baureihenfolge“ von Gebäuden

Die **Reihenfolge in der die KI von Siedler 6 Gebäude aufbaut** definiert sich durch die im MapEditor in den Gebäudeeigenschaften festgelegte AI-Building-Order (Zahl von 1 bis X).

Mit Hilfe der folgenden Funktion wird definiert bis zu welcher AI-Building-Order die KI Gebäude aufbaut. Ergänzt diese Zeile unten in `Mission_FirstMapAction()`. Wird sie an weiteren Stellen hinzugefügt und beispielsweise nach Ablauf einer bestimmten Zeitspanne aufgerufen, so kann hierdurch ein stufenweiser Aufbau der KI erreicht werden.

Wird die Zeile nicht verwendet, baut die KI alle zuvor im MapEditor gesetzten Gebäude auf.

```
AICore.SetNumericalFact( AIPlayerID, "BPMX", AIBuildingOrder )
```

- *AIPlayerID*: ID des KI Spielers dessen Aufbau bestimmt werden soll.
- *AIBuildingOrder*: Die letzte Stufe von Gebäuden die von der KI gebaut werden sollen.

Beispiel für den KI Spieler mit der ID 3, welcher zunächst nur eine Holzfällerhütte und ein Jagdhaus aufbaut:

Im MapEditor müssen 3 Gebäude auf das Gebiet des KI Spielers gestellt werden und dessen ID zugewiesen bekommen – 2 Holzfäller und ein Jagdhaus.

Mit einem Doppelklick auf die gerade erstellten Häuser wird das Eigenschaftsmenü geöffnet. Hier wird für das Jagdhaus sowie eine der Holzfällerhütten unter AI-Building-Order der Wert 1 eingetragen. Die zweite Holzfällerhütte erhält hier eine 2.

```
function Mission Mission_InitPlayers()
    AICore.SetNumericalFact( 3, "BPMX", 1 )
end
```

Wird `SetNumericalFact` später noch mit der Nummer 2 aufgerufen, wird auch noch die zweite Holzfällerhütte aufgebaut.

Da **Mauern** keine AI-Building-Order im MapEditor bekommen können, kann diese per Script gesetzt werden. Fügt dazu folgende Funktion eurem Script hinzu:

```
function GameCallback_AIWallBuildingOrder(_PlayerID)
    if _PlayerID == AIPlayerID then
        return AIBuildingOrder
    end
end
```

So kann mit den oben schon beschriebenen Parametern festgelegt werden bei welcher Baustufe die Mauern aufgebaut werden sollen.

Sollen **zerstörte Gebäude nicht automatisch wiederaufgebaut werden**, muss folgende Zeile einmalig ausgeführt werden:

```
AICore.SetNumericalFact( AIPlayerID, "BARB", 0 )
```

5. Außenposten besetzen

Wenn der KI Spieler von Anfang an weitere Territorien mit Außenposten besitzt, so sind diese Außenposten nicht durch Truppen gesichert und können somit leicht vom Spieler eingenommen werden.

Um einen Außenposten mit einem Trupp Bogenschützen zu sichern muss dem Außenposten ein Name gegeben werden, beispielsweise „Outpost1“.

Dann wird für jeden Außenposten der besetzt werden soll folgender Aufruf in `Mission_FirstMapAction()` mit entsprechend angepasstem Namen eingefügt:

```
MountOutpostWithArchers( "Outpost1" )
```

Damit dieser Aufruf verwendet werden kann muss **einmalig** folgende Funktion an eine freie Stelle im Kartenscript (**nicht** innerhalb einer anderen Funktion) eingefügt werden:

```
function MountOutpostWithArchers( _Outpost )
    local outpostID = assert( Logic.GetEntityIDByName(_Outpost) )
    local AIPlayerID = Logic. EntityGetPlayer(outpostID)

    local ax, ay = Logic.GetBuildingApproachPosition(outpostID)
    local TroopID = Logic.CreateBattalionOnUnblockedLand(
                      Entities.U_MilitaryBow, ax, ay, 0, AIPlayerID, 0 )
    AICore.HideEntityFromAI( AIPlayerID, TroopID, true )
    Logic.CommandEntityToMountBuilding( TroopID, outpostID )
end
```

Die eingefügte Funktion braucht nicht verändert zu werden.

6. Wikingerangriff per Schiff

Wikinger werden von sich aus **keine speziellen Gebäude** des Spielers angreifen (Burg, Lagerhaus, Kathedrale, Marktplatz). **Rohstoffsammler oder Stadtgebäude** können allerdings angezündet werden.

Damit alles funktioniert, müssen für die Spieler ID des Wikinger KI Spielers im MapEditor einige **ScriptEntities** platziert werden (über „Entitäten Platzieren“ > „Script“)

- XD_ScriptEntity mit dem Scriptnamen „VikingsShipStart“. Hier wird das Wikingerschiff vor dem Angriff erscheinen.
- XD_ScriptEntity mit dem Scriptnamen „VikingsShipEnd“ in Strandnähe. Bis hier hin wird das Wikingerschiff vor dem Angriff fahren.
- S_AIHomePosition in der Nähe des Strandes an dem die Wikinger landen sollen.
- XD_ScriptEntity mit dem Scriptnamen „VikingsSpawnPoint“ auf einem begehbaren Strandbereich. Hier werden die Wikinger aus dem Schiff aussteigen.
- Optional eine XD_ScriptEntity mit dem Scriptnamen „VikingsTarget“. Wenn kein solches Ziel gesetzt ist, werden die Wikinger sich selbst einen Angriffsbereich aussuchen.

Sucht nach der Funktion `Mission_InitPlayers()` in eurem MapScript. Ergänzt die Funktion um den folgenden Scriptblock:

```
Viking = AIPlayer:new( AIPlayerID, AIPlayerProfile_Viking )
Viking.m_ShipSpawnID      = Logic.GetEntityIDByName("VikingsShipStart")
Viking.m_ShipEndID        = Logic.GetEntityIDByName("VikingsShipEnd")
Viking.m_VikingSpawnID    = Logic.GetEntityIDByName("VikingsSpawnPoint")
Viking.m_NumberOfVikings  = NumberOfAttackers
Viking.m_StartAttackMonth = Month
-- Die folgende Zeile wird nur benötigt wenn ein Ziel gesetzt wurde
Viking.m_TargetID         = Logic.GetEntityIDByName("VikingsTarget")
```

- *`AIPlayerID`*: Eine freie PlayerID. Es sollten keine Gebäude oder Einheiten mit dieser PlayerID existieren.
- *`NumberOfAttackers`*: Anzahl der nach der Schiffslandung erstellten Wikinger Einheiten. 8 Einheiten reichen meistens aus um eine Holzfällerhütte zu zerstören.
- *Month*: entspricht dem Monat in dem die Wikinger-Attacke ausgelöst wird.

Hinweis: Bis zum Patch 1.4 muss **immer ein Ziel** gesetzt werden damit die Wikinger zuverlässig angreifen nachdem sie aus dem Schiff gestiegen sind.

7. Interaktive Objekte, Ruinen

Ein **interaktives Objekt** mit dem **Scriptnamen** „IO1“ kann mit folgenden Zeilen, die beispielsweise in `Mission_FirstMapAction()` eingefügt werden **konfiguriert** werden.

```
IOID = Logic.GetEntityIDByName("IO1")
Logic.InteractiveObjectSetPlayerState(IOID, PlayerID, Use1 )
Logic.InteractiveObjectSetAvailability(IOID, Use2 )
Logic.InteractiveObjectSetInteractionDistance(IOID, Distance )
Logic.InteractiveObjectSetTimeToOpen(IOID, Time )
Logic.InteractiveObjectClearCosts(IOID)
Logic.InteractiveObjectClearRewards(IOID)
Logic.InteractiveObjectAddCosts(IOID, CostGood, CostAmount )
Logic.InteractiveObjectSetCostGoldCartType( ID, GoldCart )
Logic.InteractiveObjectSetCostResourceCartType( ID, GoodCart )
```

Die folgenden drei Zeilen werden nur benötigt, wenn der Spieler durch
das interaktive Objekt **Ressourcen** erhalten soll

```
Logic.InteractiveObjectAddRewards(IOID, RewardGood, RewardAmount )
Logic.InteractiveObjectSetRewardGoldCartType( ID, GoldCart )
Logic.InteractiveObjectSetRewardResourceCartType( ID, GoodCart )
```

Die folgende Zeile wird nur benötigt, wenn das interaktive Objekt
durch ein anderes Objekt ersetzt werden soll wenn es aktiviert wird.
Dies kann **nur dann** gemacht werden wenn der Spieler **keine Ressourcen** durch das Objekt erhält.

```
Logic.InteractiveObjectSetReplacingEntityType( IOID, Replace )
```

- *PlayerID*: ID des Spielers für den das interaktive Objekt (nicht) benutzbar sein soll. Diese ist normalerweise 1.
- *Use1*: Art der Benutzungsmöglichkeit: 0 - kann nur durch einen Ritter benutzt werden. 1 – kann immer benutzt werden. 2 – kann nie benutzt werden.
- *Use2*: Hiermit wird global die Möglichkeit zur Benutzung (de)aktiviert. Mögliche Werte sind `true` (aktiv) und `false` (nicht aktiv).
- *Distance*: Entfernung aus der ein Ritter das Objekt aktivieren kann. Normalerweise werden hier Werte zwischen 1000 und 2000 verwendet.
- *Time*: Zeit in Sekunden bis das Objekt aktiviert werden kann nachdem ein Ritter in Reichweite ist.
- *CostGood* und *RewardGood*: Ressource die für die Aktivierung benötigt oder nach der Aktivierung übergeben wird. Eine entsprechende Auflistung ist unter Punkt 2 zu finden, wobei nicht alle Ressourcen fehlerfrei verwendet werden können. Oft wird hier Holz oder Gold eingetragen.
- *CostAmount und RewardAmount:* Menge der Ressource die ein Spieler aufbringen muss, beziehungsweise nach der Aktivierung erhält.
- *GoldCart und GoodCart:* Wagen der zum Transport der entsprechenden Ressource verwendet werden soll („Entities.“ voranstellen nicht vergessen). Möglich sind hier: `U_GoldCart U_Noblemen_Cart U_RegaliaCart U_ThiefCart U_ResourceMerchant`
- *Replace:* Gebäude durch das das interaktive Objekt nach der Aktivierung ersetzt werden soll. Dies wird normalerweise nur für Signalfeuer verwendet (siehe Beispiel auf der nächsten Seite).

Es folgt ein **Beispiel** für ein entzündbares Signalfeuer, welches zunächst nicht benutzbar ist. Später kann es dann durch ein Ereignis aktiviert und schließlich von dem Ritter von Spieler 1 nach 10 Sekunden Vorbereitungszeit entzündet werden.

Es kostet 20 Holz und 5 Gold (der Holzfahrer möchte schließlich bezahlt werden) das Signalfeuer zu entzünden.

Nachdem das Signalfeuer entzündet ist wird ein Stein von der Feuerstelle sowie übrig gebliebenes Gold mit zurück genommen.

Als Vorbereitung muss im MapEditor unter „Entitäten Platzieren“ > „Interactive Objects“ ein Signalfeuer (I_X_SignalFire_Base) mit dem Scriptnamen „SignalFire“ gesetzt werden.

```
function Mission_FirstMapAction()
    SignalFireID = Logic.GetEntityIDByName("SignalFire")
    Logic.InteractiveObjectSetPlayerState( SignalFireID, 1, 2 )
    Logic.InteractiveObjectSetAvailability( SignalFireID, false )
end

function Activate_SignalFireID()
    SignalFireID = Logic.GetEntityIDByName("SignalFire")

    Logic.InteractiveObjectSetPlayerState( SignalFireID, 1, 0 )
    Logic.InteractiveObjectSetAvailability( SignalFireID, true )
    Logic.InteractiveObjectSetInteractionDistance( SignalFireID, 1500 )
    Logic.InteractiveObjectSetTimeToOpen( SignalFireID, 10 )

    Logic.InteractiveObjectClearCosts(SignalFireID)
    Logic.InteractiveObjectAddCosts( SignalFireID, Goods.G_Wood, 20 )
    Logic.InteractiveObjectAddCosts( SignalFireID, Goods.G_Gold,  5 )

    Logic.InteractiveObjectSetCostGoldCartType( ID, Entities.U_GoldCart)
    Logic.InteractiveObjectSetCostResourceCartType( ID,
                                                    Entities.U_ThiefCart )

    Logic.InteractiveObjectClearRewards(SignalFireID )
    Logic.InteractiveObjectAddRewards( ID, Goods.G_Stone, 1 )
    Logic.InteractiveObjectAddRewards( ID, Goods.G_Gold,  1 )

    Logic.InteractiveObjectSetRewardGoldCartType( ID,
                                                   Entities.U_GoldCart )
    Logic.InteractiveObjectSetRewardResourceCartType( ID,
                                                    Entities.U_ThiefCart )

    Logic.InteractiveObjectSetReplacingEntityType( SignalFireID,
                                          Entities.D_X_SignalFire_Fire )
end
```

Wer kein Ereignis, welches `Activate_SignalFireID()` aufruft, hinzufügen möchte, kann dies auch von der letzten Zeile in `Mission_FirstMapAction()` aus aufrufen.

8. Titel und Technologien einschränken

In Karten macht es Sinn den maximal erreichbaren **Rittertitel** einzuschränken um die Karte nicht zu leicht werden zu lassen. Auch können einige **Technologien** zu Anfang gesperrt und dann nach Erfüllung eines Quests freigeschaltet werden.

Mit dieser Zeile wird der Rittertitel festgelegt, den der Spieler nicht mehr erreichen kann. Die Zeile wird normalerweise in `Mission_InitPlayers()` eingefügt.

```
LockTitleForPlayer( PlayerID, Title )
```

- *PlayerID*: ID des Spielers. Diese ist normalerweise 1.
- *Title*: Der Titel den der Spieler nicht mehr erreichen kann. Jedem hier eingetragenen Titel muss ein „`KnightTitles.`“ vorangestellt werden. Es können folgende Titel verwendet werden: `Knight Mayor Baron Earl Marquees Duke Archduke`

Beispiel bei dem der Spieler nur die Grundgebäude bauen und keinen zusätzlichen Rittertitel erlangen kann:

```
LockTitleForPlayer( 1, KnightTitles.Mayor )
```

Während die Sperrung eines Rittertitels nicht mit bestehenden Funktionen rückgängig gemacht werden kann, können Technologien beliebig gesperrt und wieder freigeschaltet werden. Daher kann der entsprechende Aufruf an allen entsprechenden Stellen eingefügt werden an denen die Nutzung der Technologien verändert werden soll:

```
local TechologiesToChange = {
    Tech,
    Tech,
}
```

Die folgende Zeile wird nur benötigt, wenn die **Technologien gesperrt werden** sollen:

```
LockFeaturesForPlayer( PlayerID, TechologiesToChange )
```

Die folgende Zeile wird nur benötigt, wenn die **Technologien wieder freigeschaltet werden** sollen:

```
UnLockFeaturesForPlayer( PlayerID, TechologiesToChange)
```

- *PlayerID*: ID des Spielers. Diese ist normalerweise 1.
- *Tech*: Zu ändernde Technologie. Jeder hier eingetragenen Technologie muss ein „Technologies.“ vorangestellt werden. Es können folgende Technologien verwendet werden:

```
R_AmmunitionCart R_Bakery R_Ballista R_BannerMaker R_Barracks
R_BarracksArchers R_Baths R_BatteringRam R_Beekeeper R_Blacksmith
R_BowMaker R_BroomMaker R_BuildingUpgrade R_Butcher R_CandleMaker
R_Carpenter R_Castle_Upgrade_1 R_Castle_Upgrade_2 R_Castle_Upgrade_3
R_Catapult R_Cathedral_Upgrade_1 R_Cathedral_Upgrade_2
R_Cathedral_Upgrade_3 R_CattleFarm R_Clothes R_Construction R_Dairy
R_Entertainment R_Festival R_FishingHut R_Gathering R_GrainFarm
R_HerbGatherer R_HuntersHut R_Hygiene R_IronMine R_KnockDown
R_Medicine R_Military R_Nutrition R_Pallisade R_Prosperity R_Sermon
R_SheepFarm R_SiegeEngineWorkshop R_SiegeTower R_SmokeHouse
R_Soapmaker R_StoneQuarry R_Storehouse_Upgrade_1
```

```
R_Storehouse_Upgrade_2 R_Storehouse_Upgrade_3 R_Street R_SwordSmith
R_Tanner R_Tavern R_Taxes R_Theater R_Thieves R_Trade R_Trail
R_Victory R_Wall R_Wealth R_Weaver R_Woodcutter
```

Beispiel in dem der Jäger und der Fischer für den Spieler gesperrt werden:

```
local TechologiesToLock = {
    Technologies.R_HuntersHut,
    Technologies.R_FishingHut,
}

LockFeaturesForPlayer( 1, TechologiesToLock )
```

Beispiel in dem die zuvor gesperrten Technologien später durch den Aufruf der Funktion wieder entsperrt werden können:

```
function UnlockTechs()
    local TechologiesToLock = {
        Technologies.R_HuntersHut,
        Technologies.R_FishingHut,
    }

    UnLockFeaturesForPlayer( 1, TechologiesToLock )
end
```

9. Siegesfeier

Es gibt die Möglichkeit einige **besondere Aktionen** durchzuführen **nachdem die Karte gewonnen wurde**.

Eine davon ist eine **Siegesfeier** bei der auch der Ritter des Spielers anwesend sein kann.

Um die Position des Ritters bei der Feier festzulegen muss eine Script-Entity mit dem Namen „VictoryKnightPos“ in unmittelbarer Nähe zum Marktplatz platziert werden.

Danach muss eine neue Funktion in das **Kartenscript** eingefügt und beim Sieg aufgerufen werden:

```
function Mission_Victory()
    local PossibleSettlerTypes = {
        Entity,
        Entity,
    }
    VictoryGenerateFestivalAtPlayer( PlayerID, PossibleSettlerTypes )

    local VictoryKnightPos = Logic.GetEntityIDByName("VictoryKnightPos")
    local x,y = Logic.GetEntityPosition(VictoryKnightPos)
    local Orientation = Logic.GetEntityOrientation(VictoryKnightPos)
    local KnightID = Logic.GetKnightID(1)
    VictorySetEntityToPosition( KnightID, x, y, Orientation )
end
```

- *PlayerID*: ID des Spielers der gewonnen hat. Diese ist normalerweise 1.
- *Entity*: Ein Siedler der beim Fest anwesend sein kann. Hier kann eine Liste verschiedener Siedler angegeben werden. Jedem hier eingetragenen Wert muss ein „Entities.“ vorangestellt werden. Beispielsweise können folgende Werte verwendet werden:

```
U_NPC_Monk_ME U_NPC_Monk_NE U_NPC_Monk_NA U_NPC_Monk_SE
U_NPC_Villager01_ME U_NPC_Villager01_NE U_NPC_Villager01_SE
U_NPC_Villager01_NA  U_Baker U_BathWorker U_Soapmaker U_DairyWorker
U_HerbGatherer U_GrainFarmer U_CattleFarmer U_Woodcutter
U_SheepFarmer U_Weaver U_BannerMaker U_Beekeeper U_Barkeeper
U_IronMiner U_Stonecutter U_Fisher U_Hunter U_SmokeHouseWorker
U_Butcher U_Pharmacist U_BroomMaker U_Tanner U_Priest U_Blacksmith
U_CandleMaker U_Carpenter U_Actor_Nobleman U_Actor_Bandit
U_Actor_Princess U_TheatreWorker U_SpouseS01 U_SpouseS02 U_SpouseS03
U_SpouseF01 U_SpouseF02 U_SpouseF03
```

Beispiel für eine kleine Siegesfeier auf der nur Fischer und Holzfäller sowie deren Ehefrauen anwesend sind, nicht aber der Ritter des Spielers:

```
function Mission_Victory()
    local PossibleSettlerTypes = {
        Entities.U_Fisher,
        Entities.U_Woodcutter,
        Entities.U_SpouseS01,
        Entities.U_SpouseF01,
     }
    VictoryGenerateFestivalAtPlayer( 1, PossibleSettlerTypes )
end
```

10. Wegpunkte

NPCs können eine Reihe vorgegebener Wegpunkte auf verschiedene Arten ablaufen. Hierzu werden zunächst ein NPC (Units -> NPCs) sowie dessen Wegpunkte in Form von Script-Entities im MapEditor gesetzt und benannt.

Es ist auch möglich benannte Gebäude als Wegpunkte zu nutzen, allerdings können dabei in Kombination mit weiteren eigenen Einstellungen Problemen auftreten.
Der Code für die Wegpunkte wird an eine geeignete Stelle im Script kopiert,
dies ist normalerweise `Mission_FirstMapAction()`.

```
local Entity = Logic.GetEntityIDByName(NPCName)
local ControlPointsList = { WPName1, WPName2 }

Path:new( Entity, ControlPointsList, Loop, Back, OnArrival, OnReturn,
          nil, LoopCallback, nil, Distance )
```

- *NPCName*: Scriptname des NPCs.
- *WPName*: Name der gesetzten Wegpunkte.

Alle folgenden Werte sind **optional**:

- *Loop:* Wird dies auf true gesetzt, wird der NPC wieder zum ersten Wegpunkt laufen sobald der letzte Wegpunkt erreicht wurde. Die Wegpunkte werden dann erneut abgelaufen.
- *Back:* Wird dies auf true gesetzt, geht der NPC die Strecke rückwärts zurück zum Startpunkt nachdem der letzte Wegpunkt erreicht wurde.
- *OnArrival:* Eine eigene Funktion die aufgerufen wird sobald der NPC sein Ziel (den letzten Wegpunkt) erreicht hat.
- *OnReturn:* Eine eigene Funktion die aufgerufen wird sobald der NPC (mit Back = true) zurück zum ersten Wegpunkt gelaufen ist.
- *LoopCallback:* Eigene Funktion die jede Sekunde aufgerufen wird solange der NPC dem Pfad folgt.
- *Distance:* Entfernung die der NPC zu einem Wegpunkt haben muss um diesen Wegpunkt als „erreicht“ zu betrachten.

Beispiel für einen NPC mit dem Namen „NPC1“, der einmalig von „NPC1_WP0“ nach „NPC1_WP1“ läuft:

```
local Entity = Logic.GetEntityIDByName("NPC1")
local ControlPointsList = { "NPC1_WP0", "NPC1_WP1" }

Path:new( Entity, ControlPointsList )
```

Beispiel für einen NPC mit dem Namen “NPC2”, der von “NPC2_WP0“ über “NPC2_WP1“ nach “NPC2_WP2“ läuft, dort umdreht und wieder zum Startpunkt zurück läuft, hierbei muss sich der NPC jedem Wegpunkt auf 99cm nähern:

```
local Entity = Logic.GetEntityIDByName("NPC2")
local ControlPointsList = { "NPC1_WP0", "NPC1_WP1", "NPC1_WP2" }

Path:new( Entity, ControlPointsList, nil, true, nil, nil, nil, nil, nil,99)
```