# Fonctions de script

# FONCTIONS GENERIQUES

## GetGameVar

Retourne la valeur d'une variable générale du jeu.

**Syntaxe**

`GetGameVar(`*`name`*`);`

**Description**

Cette fonction retourne la valeur d'une variable générale du jeu portant le nom *`name`* si cette variable existe, sinon retourne une chaîne vide.

*`name`* – nom de la variable

## length

Détermine la longueur d'un tableau.

**Syntaxe**

`length(`*`array`*`);`

**Description**

Un tableau est un ensemble d'éléments ordonnés réunis dans une même table et indexés (c'est à dire numérotés). L'index, c'est-à-dire le numéro porté par chacun des éléments, est unique pour chaque élément du tableau. La numérotation commence à zéro pour le premier élément et se poursuit jusqu'au dernier élément sans qu'un élément soit oublié.

La fonction `length` permet de déterminer le nombre d'éléments contenus dans ce tableau. Si le tableau contient 5 éléments, ils seront donc numérotés de 0 à 4.

*`array`* – tableau

## GetDifficulty

Détermine le niveau de difficulté du jeu.

**Syntaxe**

`GetDifficulty();`

**Description**

Cette fonction permet de connaître le niveau de difficulté du jeu. Les constantes suivantes sont prédéfinies pour identifier les niveaux de difficulté possibles :

- `DIFFICULTY_EASY = 1`,
- `DIFFICULTY_NORMAL = 2`,
- `DIFFICULTY_HARD = 3`,
- `DIFFICULTY_HEROIC = 4`.

## Load

Charge le jeu depuis le fichier spécifié.

**Syntaxe**

`Load(`*`fileName`*`);`

**Description**

Cette fonction charge la partie sauvegardée dans un fichier. *`fileName`* est le nom du fichier texte de sauvegarde tel que spécifié dans les propriétés de la carte (à partir du menu de l'éditeur de cartes : `View->Map Properties Tree` ; puis dans l'arborescence de propriétés : `Resources->SaveFilenames`) et qui contient le nom localisé d'une partie sauvegardée.

## mod

Retourne le reste de la division d'un nombre par un autre (modulo).

**Syntaxe**

`mod(`*`x, y`*`);`

**Description**

Cette fonction retourne le reste de la division du premier argument par le second. Les deux arguments doivent être des nombres (pas forcément des entiers). Le second argument doit être différent de zéro.

**Exemple**

```
>mod(5, 2)
1
>mod(5.7, 1.3)
0.5
```

`mod(5.7, 1.3)` est égal à `0.5` puisque `5.7 = 1.3*4 + 0.5`

## print

Affiche la représentation textuelle de l'argument dans la console.

**Syntaxe**

`print(...);`

**Description**

Affiche la représentation textuelle de l'argument (ou des arguments) dans la console.

`print` fonctionne avec n'importe quel type de paramètre :

- les nombres, chaînes de caractères et la valeur *`nil`* sont présentés tels quel.
- les fonctions et les objets définis par l'utilisateur sont substitués par des marqueurs comme"`<function>`"
- le contenu des tableaux est affiché.

**Exemple**

```
> print(2, " sdf ", print)
2 sdf <C-function>
```

## random

Retourne un nombre au hasard.

**Syntaxe**

`random(top);`

**Description**

Cette fonction retourne un nombre au hasard entre 0 et le paramètre (positif) *top*.

## Save

Effectue une sauvegarde de la partie dans le fichier spécifié.

**Syntaxe**

`Save(fileName);`

**Description**

Cette fonction sauvegarde la partie dans le fichier spécifié. *fileName* est le nom du fichier texte de sauvegarde tel que spécifié dans les propriétés de la carte (à partir du menu de l'éditeur de cartes : `View->Map Properties Tree` ; puis dans l'arborescence de propriétés : `Resources->SaveFilenames`) et qui contient le nom localisé d'une partie sauvegardée.

## SetGameVar

Modifie la valeur d'une variable générale du jeu.

**Syntaxe**

`SetGameVar(name, value);`

**Description**

Cette fonction change la valeur d'une variable générale *name* en *value*. S'il n'existe pas de variable portant ce nom, cette dernière est créée.

*name* – nom de la variable

*value* – valeur à affecter à la variable

## Sleep

Suspend temporairement l'exécution du processus courant (thread).

**Syntaxe**

`Sleep(nombre-de-segments);`

**Description**

Cette fonction suspend temporairement l'activité du processus courant pour une durée spécifiée. Le temps est compté en nombre de segments de jeu. *nombre-de-segments* est un entier supérieur ou égal à 1.

La commande `sleep` est nécessaire pour créer des scripts qui exécutent des actions périodiques sur des périodes de temps relativement longues (comme les scénarios de carte) :

```
while 1 do
     sleep(100)
     print("100 segments de jeu viennent encore de se dérouler")
end
```

L'effet important de la commande `sleep` est que son exécution rend le contrôle au processus de planification du moteur de script en le transférant depuis le processus du script en cours. A partir de ce moment, le processus de planification est activé pour laisser d'autres processus de script fonctionner, puis finalement, rend la main au jeu.

Par exemple, si le script ci-dessus est exécuté en supprimant l'appel `sleep(100)`, le jeu se figera : le processus dans lequel s'exécutera le script ne rendra jamais le contrôle au moteur de script, qui ne pourra donc jamais rendre le contrôle au jeu.

## sqrt

Retourne la racine carrée du nombre.

**Syntaxe**

`sqrt(`*`x`*`);`

**Description**

Cette fonction retourne la racine carrée de son argument, qui doit être un nombre positif.

## startThread

Démarre une fonction dans un nouveau processus d'exécution (thread).

**Syntaxe**

`startThread(`*`func`*`);`

**Description**

Cette fonction démarre la fonction spécifiée dans un nouveau processus d'exécution.
Le paramètre *`func`* doit être une fonction de script : *`func`* ne peut pas être l'une des commandes décrites dans ce manuel (parce qu'elles ont été développées en C).

```
local immortal = "archer immortel";

function eternalLoop()
     while 1 do
            if not exist(%immortal) then
                  resurrect(%immortal)
            end
            sleep(1)
     end
end

print("Démarrage de la boucle de résurrection")
startThread(eternalLoop)
```

# SUR LA CARTE D'AVENTURE

## AddHeroCreatures

Ajoute un stack de créature à l'armée d'un héros.

**Syntaxe**

`AddHeroCreatures(`*`heroname`*`, `*`creatureID`*`, `*`quantity`*`);`

**Description**

Ajoute un certain nombre (*`quantity`*) de créatures de type *`creatureID`* à l'armée du héros *`heroname`* (*`heroname`* est le nom interne du héros, celui qui est défini lorsque la carte est créée). La commande fonctionnera même si le héros n'est pas sur la carte (dans une ville, en réserve, …).

Seules des créatures peuvent être ajoutées. Pour ajouter une machine de guerre (baliste, catapulte, etc.), il faut utiliser la fonction `GiveHeroWarMachine`.

## AddObjectCreatures

Ajoute un stack de créature à l'armée d'un objet

**Syntaxe**

`AddObjectCreatures(`*`objectName`*`, `*`creatureID`*`, `*`quantity`*`);`

**Description**

Cette fonction ajoute un certain nombre (*`quantity`*) de créatures de type *`creatureID`* à l'armée d'un objet. Des créatures peuvent être ajoutées à n'importe quel objet de la carte qui puisse accueillir une armée (y compris les monstres et les héros). Les monstres ne peuvent accueillir que des monstres du même type.

Seules des créatures peuvent être ajoutées, pas des machines de guerre (baliste, catapulte, etc.).

*`objectName`* – nom de l'objet.

*`creatureID`* – type de créature

*`quantity`* – nombre de créatures à ajouter

## BlockGame

Bloque l'interface utilisateur et le travail de l'IA

**Syntaxe**

`BlockGame();`

**Description**

Cette commande bloque l'interface utilisateur, le travail de l'IA, ainsi que la gestion de la caméra. De cette façon, tout est figé, et le script devient l'unique source d'événements dans le jeu. La commande fige aussi les héros en mouvement.
Pour débloquer le jeu, il faut utiliser la commande `UnblockGame`. L'utilisation de blocages successifs est possible et ces blocages sont alors accumulés, ce qui signifie que le jeu sera débloqué une fois que le nombre d'appels à `UnblockGame` sera égal à celui des appels à `BlockGame`.

## CalcHeroMoveCost

Calcule le coût de déplacement du héros vers le point spécifié.

**Syntaxe**

```
CalcHeroMoveCost(heroName, x, y, floorID = -1);
```

**Description**

Cette fonction calcule le coût de déplacement du héros vers le point spécifié, en utilisant les points de mouvements comme unité de calcul. Si l'étage (*floorID*) n'est pas spécifié, c'est celui où se trouve le héros qui est utilisé. Cette fonction ne peut être appelée que pour les héros contrôlés par l'IA. Seuls les objets statiques sont pris en compte lorsque le chemin est construit. Si le point spécifié ne peut être atteint, la fonction retourne la valeur -1.

*heroName* – id du héros.

*x, y* – coordonnées de la case de destination

*floorID* – numéro d'étage (-1 par défaut, qui indique l'étage où se trouve déjà le héros)

## CanMoveHero

Détermine si le héros peut être déplacé vers le point spécifié.

**Syntaxe**

```
CanMoveHero(heroName, x, y, floorID = -1);
```

**Description**

Cette fonction retourne *true* si le point spécifié peut être atteint par le héros. Sinon la valeur *false* est renvoyée. Si l'étage (*floorID*) n'est pas spécifié, c'est celui où se trouve le héros qui est utilisé. Cette fonction ne peut être appelée que pour les héros contrôlés par l'IA.

*heroName* – id du héros.

*x, y* – coordonnées de la case de destination

*floorID* – numéro d'étage (-1 par défaut, qui indique l'étage où se trouve déjà le héros)

## ChangeHeroStat

Change les statistiques du héros

**Syntaxe**

```
ChangeHeroStat(heroName, statID, delta);
```

**Description**

Cette fonction modifie la statistique d'un héros d'une valeur *delta* (positive ou négative). Les valeurs des statistiques ne peuvent pas être négatives. Les points de mouvement et de mana sont limités à leurs valeurs maximales respectives. Si la modification entraîne le dépassement des valeurs limites, la valeur résultante de la statistique est tronquée à la valeur limite.

*heroName* – id du héros.

*statID* – identifiant de la statistique, qui peut prendre les valeurs suivantes :

- `0 = STAT_EXPERIENCE` – expérience du héros (*delta* ne peut être que positif)
- `1 = STAT_ATTACK` – attaque

- `2 = STAT_DEFENCE` – défense
- `3 = STAT_SPELL_POWER` – puissance
- `4 = STAT_KNOWLEDGE` – savoir
- `5 = STAT_LUCK` – chance
- `6 = STAT_MORALE` – moral
- `7 = STAT_MOVE_POINTS` – nombre de points de mouvement restants
- `8 = STAT_MANA_POINTS` – nombre de points de mana (magie) restants

*`delta`* – valeur de modification de la statistique

## CreateMonster

Crée un monstre sur la carte

**Syntaxe**

```
CreateMonster(monsterName, creatureType, creaturesCount, x, y, floorID,
 mood = MONSTER_MOOD_AGGRESSIVE,
 courage = MONSTER_COURAGE_CAN_FLEE_JOIN,
 rotation = 0);
```

**Description**

Crée un monstre avec *`creaturesCount`* créatures de type *`monsterType`* type sur la case (*`x`*, *`y`*) à l'étage *`floorID`*, lui affectant le nom *`monsterName`*. Si la case spécifiée est occupée, le monstre est placé sur l'une des cases adjacentes disponibles. Les deux derniers paramètres affectent le comportement du monstre lorsqu'il rencontre un héros.

*`monsterName`* – nom du monstre, qui pourra être utilisé avec d'autres commandes de script.

*`creatureType`* – type de créatures.

*`creaturesCount`* – nombre de créatures.

*`x, y`* – coordonnées de la case sur laquelle le monstre doit être placé.

*`floorID`* – numéro d'étage sur lequel le monstre doit être placé.

*`mood`* – attitude du monstre, caractérise son agressivité :

- `MONSTER_MOOD_FRIENDLY` – amical
- `MONSTER_MOOD_AGGRESSIVE` – agressif
- `MONSTER_MOOD_HOSTILE` – hostile
- `MONSTER_MOOD_WILD` – sauvage

*`courage`* – intentions du monstre :

- `MONSTER_COURAGE_ALWAYS_JOIN` – rejoindra toujours l'armée du héros
- `MONSTER_COURAGE_ALWAYS_FIGHT` – entamera toujours le combat
- `MONSTER_COURAGE_CAN_FLEE_JOIN` – peut s'enfuir ou rejoindre l'armée du héros

*`rotation`* – l'angle de rotation du monstre (0 par défaut).

## DeployReserveHero

Place le héros, qui a été mis dans la réserve du joueur, sur la carte.

**Syntaxe**

```
DeployReserveHero(heroName, x, y, floor);
```

**Description**

Lorsqu'une carte est créée, des héros peuvent être mis en réserve pour tel ou tel joueur. Ces héros n'apparaîtront pas dans la taverne et ne peuvent être engagés par les moyens usuels. Pour placer le héros mis en réserve sur la carte, utilisez la commande `DeployReserveHero`, en indiquant en paramètres l'identifiant du héros et la position voulue sur la carte. Si la case spécifiée est inaccessible pour une quelconque raison, le héros sera placé sur une case adjacente.

Les héros réservés qui sont tués, ont fuit ou disparaissent de la liste des héros possédés par le joueur pour d'autres raisons, sont à nouveau en réserve et peuvent être replacés sur la carte avec la même commande. Leurs armées sont réinitialisées à celles spécifiées dans l'éditeur. Pour enlever un héros de la réserve d'un joueur et le rendre disponible dans la taverne des autres joueurs, utilisez la fonction `UnreserveHero`.

*`heroName`* – id du héros mis en réserve pour un joueur.

*`x, y`* – coordonnées de la case sur laquelle doit être placé le héros.

*`floor`* – Le numéro d'étage sur lequel le héros doit être placé.

## EnableAIHeroHiring

Autorise/Interdit le recrutement de héros par l'IA (Intelligence Artificielle) dans une ville

**Syntaxe**

`EnableAIHeroHiring(`*`playerID, townName, enable`*`);`

**Description**

Cette fonction vous permet d'autoriser ou d'interdire à l'IA de recruter des héros dans la taverne d'une ville (par défaut, ceci lui est autorisé).

*`playerID`* – ID du joueur

*`townName`* – Nom de la ville

*`enable`* – *`true`* pour autoriser, *`false`* pour interdire le recrutement.

## EnableHeroAI

Active ou désactive le contrôle de l'IA sur le héros spécifié.

**Syntaxe**

`EnableHeroAI(`*`heroName, enable`*`);`

**Description**

Cette fonction permet d'activer ou de désactiver le contrôle de l'IA sur le héros spécifié. Cette commande ne peut être utilisée qu'en mode Solo. Lorsque cette fonction est appelée pour un héros normalement contrôlé par un joueur humain, un message d'erreur est généré. Si le statut de contrôle d'un héros est le même que celui que la fonction essaie de mettre en place, rien ne se passe.

*`heroName`* – id du héros

*`enable`* – *`true`* pour activer l'IA, et *`false`* pour la désactiver.

## IsObjectExists

Détermine si l'objet spécifié existe sur la carte

**Syntaxe**

```
IsObjectExists(objectName);
```

**Description**

Cette fonction retourne *true* si l'objet portant le nom indiqué en paramètre est présent sur la carte, et *false* s'il n'y est pas. Cette fonction peut être utilisée pour la Larme d'Asha, en passant comme nom d'objet la valeur OBJECT_GRAIL.

*objectName* – le nom de l'objet.

## GenerateMonsters

Génère des monstres sur la carte.

**Syntaxe**

```
GenerateMonsters(monsterTypeID, countGroupsMin, countGroupsMax,
 countInGroupMin, countInGroupMax);
```

**Description**

Cette fonction génère des monstres sur des emplacements aléatoires de la carte d'aventure. Il y aura entre *countGroupsMin* et *countGroupsMax* groupes de créatures générés, chaque groupe étant constitué de *countInGroupMin* à *countInGroupMax* créatures.

*monsterTypeID* – Identifiant du type de créature

*countGroupsMax* ≥ *countGroupsMin* ≥ *0* – nombres maxi et mini de groupes générés

*countInGroupMax* ≥ *countInGroupMin* ≥ *0* – nombres maxi et mini de créatures par groupe

## GetCurrentPlayer

Détermine le joueur en cours

**Syntaxe**

```
GetCurrentPlayer();
```

**Description**

Cette fonction retourne l'identifiant du joueur dont c'est le tour de jeu.

## GetDate

Retourne la date en cours dans le jeu (jour, semaine, mois ou jour de la semaine)

**Syntaxe**

```
GetDate(dateTypeID = DAY);
```

**Description**

Cette fonction retourne la date du jeu en cours. Le paramètre indique la valeur qui doit être déterminée : le jour en cours, la semaine, le mois, ou bien le jour de la semaine. Par défaut, lorsque cette fonction est appelée sans paramètres, c'est le jour en cours qui est renvoyé.

*dateTypeID* – type d'information temporelle recherchée. Valeurs possibles :

- `DAY ou ABSOLUTE_DAY` – jour
- `WEEK` – semaine
- `MONTH` – mois
- `DAY_OF_WEEK` – jour de la semaine

## GetHeroCreatures

Retourne le nombre de créatures du type spécifié placées sous le commandement d'un héros.

**Syntaxe**

`GetHeroCreatures(`*heroName, creatureID*`);`

**Description**

*heroName* – id du héros

*creatureID* – identifiant du type de créature

## GetHeroLevel

Retourne le niveau d'un héros

**Syntaxe**

`GetHeroLevel(`*heroname*`);`

**Description**

Cette fonction retourne le niveau en cours du héros *heroname*. Une erreur est renvoyée s'il n'existe pas de héros portant le nom *heroname* : `IsHeroAlive(`*heroname*`)` retourne *nil/false*

## GetHeroStat

Retourne les statistiques d'un héros

**Syntaxe**

`GetHeroStat(`*heroName, statID*`);`

**Description**

Cette fonction retourne la valeur de la statistique d'un héros. Les valeurs retournées prennent en compte toutes les influences éventuelles (artéfacts, etc.)

*heroName* – id du héros.

*statID* – identifiant de la statistique, qui peut prendre les valeurs suivantes :

- `0 = STAT_EXPERIENCE` – expérience du héros
- `1 = STAT_ATTACK` – attaque
- `2 = STAT_DEFENCE` – défense
- `3 = STAT_SPELL_POWER` – puissance
- `4 = STAT_KNOWLEDGE` – savoir
- `5 = STAT_LUCK` – chance
- `6 = STAT_MORALE` – moral
- `7 = STAT_MOVE_POINTS` – nombre de points de mouvement restants
- `8 = STAT_MANA_POINTS` – nombre de points de mana (magie) restants

- `9` – nombre de points de mouvement en début de tour

## GetObjectCreatures

Retourne le nombre de créatures du type spécifié dans l'armée d'un objet

**Syntaxe**

```
GetObjectCreature(objectName, creatureID);
```

**Description**

Cette fonction retourne le nombre de créatures du type spécifié dans l'armée d'un objet. On entend ici par « Objet » tous les objets de la carte (bâtiments, ainsi que monstres et héros).

*`objectName`* – le nom de l'objet

*`creatureID`* – le type de créature

## GetObjectiveProgress

Détermine la progression parcourue dans l'accomplissement d'un objectif.

**Syntaxe**

```
GetObjectiveProgress(objectiveName, playerID = PLAYER_1);
```

**Description**

Cette fonction retourne la progression réalisée dans l'accomplissement de l'objectif *`objectiveName`*. Pour les objectifs communs à tous les joueurs, le paramètre *`playerID`* précise le joueur pour lequel la progression doit être déterminée. Pour les objectifs spécifiques à un joueur, le paramètre *`playerID`* est ignoré.

Le nombre d'étapes franchies dans un objectif contrôlé manuellement est déterminé par le nombre de commentaires de progression définis dans l'éditeur. Les objectifs « capturez les objets » et « détruisez les armées neutres » possèdent un nombre d'étapes de progression respectivement égal au nombre d'objet à capturer sur la carte ou au nombre d'armées à éliminer. Les autres types d'objectifs classiques soit sont atteints, soit non.

*`objectiveName`* – nom de l'objectif

*`playerID`* – identifiant du joueur pour lequel la progression doit être évaluée

## GetObjectiveState

Détermine l'état d'un objectif

**Syntaxe**

```
GetObjectiveState(objectiveName, playerID = PLAYER_1);
```

**Description**

Cette fonction retourne l'état de l'objectif *`objectiveName`* pour le joueur indiqué. Pour les objectifs spécifiques à un joueur, le paramètre *`playerID`* est ignoré. Pour les objectifs communs à tous les joueurs, *`playerID`* précise le joueur pour lequel la progression doit être déterminée.

*`objectiveName`* – nom de l'objectif

*`playerID`* – identifiant du joueur pour lequel l'état de l'objectif doit être renvoyé

Liste des états possibles des objectifs :

- `OBJECTIVE_SCENARIO_INFO` – statut spécial désignant la description du scénario
- `OBJECTIVE_UNKNOWN` – l'objectif est encore inconnu du joueur. S'il est visible (voir les fonctions `GetObjectiveState` / `SetObjectiveState`), seule une vague description est affichée dans le panneau des objectifs.
- `OBJECTIVE_ACTIVE` – l'objectif est actif
- `OBJECTIVE_COMPLETED` – l'objectif a été atteint
- `OBJECTIVE_FAILED` – la mission a échouée

## GetObjectOwner

Retourne le propriétaire d'un objet.

**Syntaxe**

`GetObjectOwner(`*`objectName`*`);`

**Description**

Cette fonction retourne l'identifiant du joueur propriétaire de l'objet *`objectName`*, et `PLAYER_NONE` s'il n'appartient à aucun joueur.

*`objectName`* – nom de l'objet

## GetObjectsInRegion

Liste les objets présents dans une région.

**Syntaxe**

`GetObjectsInRegion(`*`regionName, objectType`*`);`

**Description**

Cette fonction retourne les noms des objets du type spécifié présents dans la région *`regionName`*. Les objets sans nom sont ignorés.

*`regionName`* – nom de la région

*`objectType`* – type des objets à lister. Peut prendre les valeurs :

- `OBJECT_HERO` – héros

## GetObjectPosition

Détermine la position d'un objet sur la carte.

**Syntaxe**

`GetObjectPosition(`*`objectName`*`);`

**Description**

Cette fonction retourne 3 valeurs : les coordonnées *`x`* et *`y`* de la case (le centre de l'objet si celui occupe plusieurs cases), et son étage. En utilisant `OBJECT_GRAIL` comme nom, la fonction renvoie la position de la Larme d'Asha.

*`objectName`* – nom de l'objet

## GetPlayerHeroes

Retourne les héros d'un joueur.

**Syntaxe**

`GetPlayerHeroes(`*`playerID`*`);`

**Description**

Cette fonction retourne dans un tableau les noms des héros du joueur spécifié.

*`playerID`* – id du joueur

## GetPlayerResource

Retourne la quantité de ressource possédée par un joueur.

**Syntaxe**

`GetPlayerResource(`*`player`*`, `*`resourceKind`*`);`

**Description**

La commande fonctionne pour les joueurs actifs et joueurs ayant déjà perdu. Elle génère une erreur si le joueur n'a pas du tout participé à la partie.

*`player`* – id du joueur

*`resourceKind`* – id de la ressource (`WOOD`, `ORE`, `MERCURY`, `CRYSTAL`, `SULFUR`, `GEM`, `GOLD`).

## GetTownBuildingLevel

Retourne le niveau d'amélioration d'un bâtiment dans une ville.

**Syntaxe**

`GetTownBuildingLevel(`*`townName`*`, `*`buildingID`*`);`

**Description**

Les bâtiments non construits ont un niveau de 0.

*`townName`* – nom de la ville

*`buildingID`* – id du bâtiment (voir la liste des identifiants de scripts)

## GetTownBuildingLimitLevel

Retourne le niveau limite d'amélioration d'un bâtiment dans une ville.

**Syntaxe**

`GetTownBuildingLimitLevel(`*`townName`*`, `*`buildingID`*`);`

**Description**

Les bâtiments non construits ont un niveau de 0. La limite peut être fixée par la fonction `SetTownBuildingLimitLevel`.

*`townName`* – nom de la ville

*`buildingID`* – id du bâtiment (voir la liste des identifiants de scripts)

## GetTownBuildingMaxLevel

Retourne le niveau maximal d'amélioration d'un bâtiment dans une ville.

**Syntaxe**

```
GetTownBuildingMaxLevel(townName, buildingID);
```

**Description**

Les bâtiments non construits ont un niveau de 0.

*townName* – nom de la ville

*buildingID* – id du bâtiment (voir la liste des identifiants de scripts)

## GetTownHero

Retourne l'identifiant du héros en garnison dans une ville.

**Syntaxe**

```
GetTownHero(townName);
```

**Description**

La fonction retourne *nil* s'il n'y a pas de héros en garnison dans la ville *townName*.

*townName* – nom de la ville

## GiveArtefact

Ajoute un artéfact à l'inventaire un héros.

**Syntaxe**

```
GiveArtefact(heroname, artefactID, bindToHero = 0);
```

**Description**

*heroname* – id du héros.

*artefactID* – id de l'artéfact.

*bindToHero* – passer la valeur `1` pour attacher l'artéfact au héros (il ne peut plus être transféré).

## LevelUpHero

Fait gagner un niveau à un héros.

**Syntaxe**

```
LevelUpHero(heroName);
```

**Description**

Cette fonction donne au héros *heroName* exactement le montant d'expérience qui lui permet de passer au niveau supérieur. La fonction renvoie *true* en cas de succès, et *nil* sinon (si le héros a déjà atteint le niveau maximal).

*heroName* – id du héros

## GiveHeroSkill

Apprend une compétence/capacité à un héros.

**Syntaxe**

`GiveHeroSkill(heroName, skillID);`

**Description**

Le héros doit être capable d'apprendre cette compétence/capacité (emplacement disponible et prérequis validés). La fonction renvoie `true` en cas de succès et `false` sinon.

Plusieurs appels successifs avec la même compétence font augmenter le niveau du héros dans cette compétence (Notions, Pratique, Avancée).

`heroName` – id du héros

`skillID` – id de la compétence/capacité

## GiveHeroWarMachine

Ajoute une machine de guerre à un héros.

**Syntaxe**

`GiveHeroWarMachine(heroName, warMachineType);`

**Description**

La fonction renvoie `nil` si le héros possède déjà cette machine de guerre.

`heroName` – id du héros

`warMachineID` – id de la machine de guerre

## HasArtefact

Indique si le héros possède l'artéfact spécifié.

**Syntaxe**

`HasArtefact(heroname, artefactID);`

**Description**

`heroName` – id du héros

`artefactID` – id de l'artéfact

## HasBorderguardKey

Indique si le joueur possède la clé de la couleur spécifiée.

**Syntaxe**

`HasBorderguardKey(player, color);`

**Description**

La fonction renvoie `not nil` si le joueur possède la clé, et `nil` sinon.

`player` – id du joueur

*color* – couleur de la clé. Peut prendre les valeurs suivantes :

- RED_KEY = 1
- BLUE_KEY = 2
- GREEN_KEY = 3
- YELLOW_KEY = 4
- ORANGE_KEY = 5
- TEAL_KEY = 6
- PURPLE_KEY = 7
- TAN_KEY = 8

## HasHeroSkill

Indique si le héros posséde la compétence/capacité spécifiée.

**Syntaxe**

`HasHeroSkill(`*heroName*`, `*skillID*`);`

**Description**

La fonction renvoie *true* si le héros possède la compétence/capacité (éventuellement grâce à un artéfact), et *false* sinon.

*heroName* – id du héros

*skillID* – id de la compétence/capacité

## HasHeroWarMachine

Indique si le héros possède la machine de guerre spécifiée.

**Syntaxe**

`HasHeroWarMachine(`*heroName*`, `*warMachineType*`);`

**Description**

La fonction renvoie *not nil* si le héros possède déjà cette machine de guerre, et *nil* sinon.

*heroName* – id du héros

*warMachineID* – id de la machine de guerre

## IsHeroAlive

Indique si le héros est en vie.

**Syntaxe**

`IsHeroAlive(`*heroname*`);`

**Description**

La fonction renvoie *not nil* si le héros existe est en vie. Un héros est considéré « en vie » s'il appartient à un joueur actif (pour qui `GetPlayerState()` renvoie *true*).

*heroName* – id du héros

## IsHeroLootable

Indique si les artéfacts du héros sont transmis au vainqueur lorsque le héros est vaincu en combat.

**Syntaxe**

`IsHeroLootable(heroName);`

**Description**

La fonction renvoie `not nil` si le transfert peut avoir lieu, et `nil` si les artéfacts ne sont pas transférés. Ce comportement peut être modifié par la fonction `SetHeroLootable`.

`heroName` – id du héros

## IsObjectEnabled

Indique si l'objet interactif peut être activé par un héros.

**Syntaxe**

`IsObjectEnabled(objectName);`

**Description**

La fonction renvoie `true` si l'objet interagit de manière standard, et `false` s'il est désactivé. Dans ce cas, il peut toujours interagir par l'intermédiaire d'un trigger associé.

`objectName` – nom de l'objet

## IsObjectInRegion

Indique si l'objet se trouve dans la région spécifiée.

**Syntaxe**

`IsObjectInRegion(objectName, regionName);`

**Description**

La fonction renvoie `true` si l'objet se trouve dans la région, et `false` sinon. Si l'objet occupe plus d'une case, il se peut que la réponse soit incorrecte.

`objectName` – nom de l'objet

`regionName` – nom de la région

## IsObjectiveVisible

Indique si l'objectif est affiché dans le panneau d'objectifs du joueur spécifié.

**Syntaxe**

`IsObjectiveVisible(objectiveName, playerID = PLAYER_1);`

**Description**

La fonction renvoie `true` si l'objectif est affiché, et `false` sinon. Le paramètre `playerID` est ignoré si l'objectif est spécifique à un joueur.

`objectiveName` – nom de l'objectif

`playerID` – id du joueur

## IsObjectVisible

Indique si l'objet est visible par le joueur spécifié.

**Syntaxe**

```
IsObjectVisible(playerID, objectName);
```

**Description**

La fonction renvoie *true* si l'objet est visible, et *false* sinon. Si l'objet occupe plus d'une case, il se peut que la réponse soit incorrecte.

*playerID* – id du joueur

*objectName* – nom de l'objet

## IsRegionBlocked

Indique si les héros du joueur peuvent pénétrer dans la région spécifiée.

**Syntaxe**

```
IsRegionBlocked(regionName, playerID);
```

**Description**

La fonction renvoie *not nil* si la région est interdite aux héros du joueur.

*regionName* – nom de la région

*playerID* – id du joueur

## KnowHeroSpell

Indique si le héros connait le sort spécifié.

**Syntaxe**

```
KnowHeroSpell(heroName, spell);
```

**Description**

La fonction renvoie *true* si le héros connait le sort (éventuellement grâce à un artéfact), et *false* sinon.

*heroName* – id du héros

*spell* – id du sort (voir les identifiants de scripts pour la liste des valeurs possibles)

## Loose

Le joueur humain perd la partie.

**Syntaxe**

```
Loose();
```

**Description**

Lorsque la fonction est invoquée, le joueur humain perd la partie instantanément.

## MarkObjectAsVisited

Marque l'objet comme visité par le héros spécifié.

**Syntaxe**

```
MarkObjectAsVisited(objectName, heroName);
```

**Description**

Cette fonction permet de marquer comme visité un objet désactivé par la commande `SetObjectEnabled`, après le déclenchement de l'événement associé.

*`objectName`* – nom de l'objet

*`heroName`* – id du héros

## MessageBox

Affiche un message dans une boîte de dialogue.

**Syntaxe**

```
MessageBox(messageName, callback = "");
```

**Description**

La boîte de dialogue possède uniquement un bouton 'OK'. Lorsque l'utilisateur clique sur ce bouton, la fonction *`callback`* est invoquée.

*`messageName`* – identifiant du message dans la base de ressources.

*`callback`* – nom de la fonction à invoquer au clic sur le bouton 'OK'.

## MoveCamera

Déplace la caméra à une case spécifique de la carte.

**Syntaxe**

```
MoveCamera(x, y, floorID, zoom = 50, pitch = pi/2, yaw = 0, noZoom = 0,
 noRotate = 0);
```

**Description**

Cette fonction déplace la caméra à une position spécifique de la carte et initialise les valeurs spécifiques de zoom et d'angle.

*`x, y, floorId`* - case de la carte où la caméra doit se positionner.

*`zoom`* – distance à laquelle la caméra doit être placée par rapport à la case.

*`pitch`* – angle de la caméra (0 – la caméra est à l'horizontale, pi/2 – la caméra est à la verticale)

*`yaw`* – angle de rotation de la caméra (0 – le nord)

*`noZoom`* – mettre à 1 pour que la caméra ne change pas son zoom pendant le déplacement

*`noRotate`* – mettre à 1 pour que la caméra ne fasse pas de rotation pendant le déplacement.

## MoveHero

Oblige le héros à se déplacer jusqu'à une case spécifique.

**Syntaxe**

```
MoveHero(heroName, x, y, floorID = -1);
```

**Description**

Cette fonction oblige le héros à se déplacer jusqu'à la case voulue. Si *floorID* n'est pas spécifié, le héros est supposé aller jusqu'à la case précisée à l'étage (sous-terrain ou surface) où il se trouve. Cette fonction n'est disponible que pour les héros contrôlés par l'IA, avec leur IA désactivée. Si la destination est inaccessible, cette fonction génère une erreur.
Si le héros n'a pas assez de point de déplacement pour arriver à destination, le déplacement va se continuer au prochain tour.

*heroName* – id du héros

*x, y* – coordonnées de la case cible

*floorID* – numéro de l'étage (-1 par défaut, le niveau de l'étage où se trouve le héros)

## MoveHeroRealTime

Déplace le héros jusqu'à la case spécifiée.

**Syntaxe**

```
MoveHeroRealTime(heroName, x, y, floorID = -1);
```

**Description**

Cette fonction oblige le héros à se déplacer jusqu'à la case voulue. Si *floorID* n'est pas spécifié, le héros est supposé aller jusqu'à la case précisée à l'étage (sous-terrain ou surface) où il se trouve. Si la destination est inaccessible, cette fonction génère une erreur. Le déplacement commence immédiatement après l'exécution de la commande, quel que soit le tour actuel. Le script se poursuit dès que la commande est exécutée, sans attendre que le héros atteigne sa destination. Les actions du joueur et de l'IA sont bloquées tant que le héros se déplace. Si le héros n'a plus de point de mouvement ou si le héros est rencontre un obstacle, il arrête son mouvement sans interagir avec l'objet (ce qui veut dire qu'il ne pourra pas utiliser un bateau, utiliser porte de téléportation, …)

*heroName* – id du héros

*x, y* – coordonnées de la case cible

*floorID* – numéro de l'étage (-1 par défaut, le niveau de l'étage où se trouve le héros)

## GetAllNames

Donne la liste des noms des objets présents sur la carte.

**Syntaxe**

```
GetAllNames(filter = 0);
```

**Description**

Cette fonction retourne une chaîne de caractère de la liste d'objet de la carte. Les noms d'unité sont séparés par un espace. Le filtre permet de spécifier le type d'objet à lister. Il n'y a pour l'instant qu'une valeur possible :

- 0 : nom des héros du joueur actif

## OpenCircleFog

Enlève le brouillard de guerre sur une zone circulaire spécifique.

**Syntaxe**

`OpenCircleFog(x, y, floorID, range, playerID);`

**Description**

Cette fonction enlève le brouillard de guerre à l'étage *floorID*, sur une zone circulaire dont le centre est aux coordonnées (*x, y*), de diamètre *range*.

*playerID* – id du joueur

*x, y, floorID* – coordonnées du centre du cercle

*range* – diamètre du cercle

## OpenRegionFog

Enlève le brouillard de guerre sur une région spécifique.

**Syntaxe**

`OpenRegionFog(player, regionName);`

**Description**

Cette fonction enlève le brouillard de guerre du joueur sur une région spécifique.

*player* – id du joueur

*regionName* – nom de la région

## Play2DSound

Joue un son 2D.

**Syntaxe**

`Play2Dsound(soundName);`

**Description**

Joue le son 2D spécifié. Si le son est cyclique, la fonction retourne l'ID du son qui peut être utilisé par la fonction `StopPlaySound` pour arrêter le son. Si le son n'est pas cyclique, la fonction retourne –1.

*soundName* – nom du son

## Play3DSound

Joue un son 3D.

**Syntaxe**

`Play3DSound(soundName, x, y, floorID);`

**Description**

Joue un son 3D du nom spécifié. Le son est joué à la case qui a pour coordonnées *x, y, floorID*.

Si le son est cyclique, la fonction retourne l'ID du son qui peut être utilisé par la fonction `StopPlaySound` pour arrêter le son. Si le son n'est pas cyclique, la fonction retourne –1.

*`soundName`* – nom du son

*`x, y, floorID`* – coordonnées d'origine du son

## PlayObjectAnimation

Lance une animation de l'objet sur la carte.

**Syntaxe**

`PlayObjectAnimation(`*`objectName, animName, action`*`);`

**Description**

Lance l'animation *`animName`* de l'objet spécifié sur la carte.

*`objectName`* – nom de l'objet à animer

*`animName`* – nom de l'animation ("idle00", "attack00", "hit", etc.). Si l'objet n'a pas d'animation de ce nom, la fonction n'a pas d'effet.

*`action`* – détermine comment l'animation est jouée, parmi les types suivants :

- `INVISIBLE` – (spécial) l'objet devient invisible.
- `IDLE` – l'animation est jouée en boucle, jusqu'à ce qu'elle soit remplacée.
- `ONESHOT_STILL` – l'animation est jouée une fois, et l'objet conserve sa position finale.
- `ONESHOT` – l'animation est jouée une fois, puis l'objet reprend son animation `IDLE`.
- `NON_ESSENTIAL` – même effet que `ONESHOT` lorsque l'objet est disponible (`IDLE`), sinon rien ne se produit.

## RazeTown

Rase une ville de la carte.

**Syntaxe**

`RazeTown(`*`townName`*`);`

**Description**

Enlève la ville spécifiée de la carte et la remplace par l'objet statique « ville rasée », déterminé par le paramètre *`razed`* de *`AdvMapTownShared`*. Si ce champ est vide, la ville ne peut être rasée.

L'objet statique spécifié comme ville rasée doit bloquer exactement les mêmes cases que la ville originale, et ne pas avoir de cases d'interaction.

*`townName`* – nom de la ville

## RegionToPoint

Donne les coordonnées et le numéro du terrain de la région.

**Syntaxe**

`RegionToPoint(`*`regionName`*`);`

**Description**

Cette fonction retourne les coordonnées *x*, *y*, et le numéro d'étage de la région à une case. Si la région est composée de plus d'une case, une erreur est générée.

*regionName* – nom de la région

## RemoveArtefact

Enlève un artéfact du héros.

**Syntaxe**

`RemoveArtefact(`*heroName*`, `*artefactID*`);`

**Description**

Enlève l'artéfact *artefactID* du héros *heroName*.

Une erreur est retournée lorsque :

- il n'y a pas de héros de nom *heroName* : `IsHeroAlive(`*heroName*`)` retourne *nil/false*
- le héros n'a pas l'artéfact : `HasArtefact(`*heroName*`, `*artefactID*`)` retourne *nil/false*

*heroName* - id du héros

*artefactID* – numéro de l'artéfact (les constantes d'artéfacts peuvent être utilisées)

## RemoveHeroCreature

Enlève les créatures de l'armée du héros.

**Syntaxe**

`RemoveHeroCreature(`*heroName*`, `*creatureID*`, `*quantity*`);`

**Description**

Enlève *quantity* créatures de type *creatureID* de l'armée du héros *heroName*.

Si le nombre de créatures du type spécifié de l'armée du héros est inférieur à *quantity*, toutes ces créatures sont enlevées. Si toutes les créatures du héros sont sur le point d'être supprimées, alors le héros conservera un peloton d'une créature.

La commande fonctionne aussi avec les héros inactifs (en prison, dans la taverne, etc.). Seules les créatures peuvent être enlevées par cette fonction. Pour les machines de guerre, il faut utiliser `RemoveHeroWarMachine`.

*heroName* – id du héros

*creatureID* – type de la créature

*quantity* – quantité de créatures

## RemoveHeroWarMachine

Enlève la machine de guerre spécifiée du héros.

**Syntaxe**

`RemoveHeroWarMachine(`*heroName*`, `*warMachineType*`);`

**Description**

Cette fonction retire la machine de guerre spécifiée au héros. Si le héros n'a pas la machine de guerre ou s'il est impossible de la lui enlever (catapulte), la fonction retourne `nil`.

*`heroName`* – id du héros

*`warMachineID`* – type de machine de guerre

Note : Un héros a toujours une catapulte, elle ne peut être enlevée.

## RemoveObject

Enlève un objet de carte d'aventure.

**Syntaxe**

`RemoveObject(objectName);`

**Description**

Cette fonction enlève l'objet spécifié de la carte d'aventure.

*`objectName`* – nom de l'objet

## RemoveObjectCreature

Enlève les créatures de l'armée d'un objet.

**Syntaxe**

`RemoveObjectCreature(objectName, creatureID, quantity);`

**Description**

Enlève *`quantity`* créatures de type *`creatureID`* de l'armée de l'objet *`objectName`*.

Si le nombre de créatures du type spécifié de l'armée de l'objet est inférieur à *`quantity`*, toutes ces créatures sont enlevées. La commande peut être appliquée à tous les objets de la carte d'aventure qui peut avoir une armée (y compris les héros et les monstres neutres). Si l'objet est un héros et que toutes son armée est sur le point d'être supprimées, alors le héros conservera un peloton d'une créature. Si l'objet est un monstre neutre et que toutes les créatures sont enlevées, les monstres sont retirés de la carte.

Seules les créatures peuvent être enlevées (et non les machines de guerre).

*`objectName`* – nom de l'objet

*`creatureID`* – type de créature

*`quantity`* – quantité

## ResetHeroCombatScript

Réinitialise le script qui aurait pu être utilisé en combattant ce héros.

**Syntaxe**

`ResetHeroCombatScript(heroName);`

**Description**

Cette fonction permet qu'aucun script ne se lance quand on combat ce héros. Si le combat prend place dans un endroit où un autre script de combat est spécifié (ville, garnison etc.), ce script se lancera.

*HeroName* – id du héros.

## ResetObjectFlashlight

Retire la source lumineuse ponctuelle de l'objet de la carte d'aventure.

**Syntaxe**

`ResetObjectFlashlight(`*objectName*`);`

**Description**

La commande retire la source de lumière supplémentaire, ajoutée par la commande `SetObjectFlashlight`, de l'objet sélectionné.

*objectName* – nom de l'objet.

## SetAIHeroAttractor

Définit un attracteur pour le héros spécifié.

**Syntaxe**

`SetAIHeroAttractor(`*objectName*`, `*heroName*`, `*priority*`);`

**Description**

Cette fonction modifie la priorité de l'objet *objectName* comme attracteur pour le héros *heroName*.

*objectName* – nom de l'objet. Cet objet doit être immobile.

*heroName* – id du héros

*priority* – priorité de l'attracteur (entre –1 et 2)

- -1 : la priorité baisse énormément
- 0 : la priorité est inchangée (l'appel à la fonction est donc inutile)
- 1 : la priorité augmente beaucoup, mais l'IA ne risquera pas son héros pour prendre le contrôle de l'objet.
- 2 : prendre le contrôle de l'objet devient prioritaire (comme un objectif de victoire).

## SetAIPlayerAttractor

Définit un attracteur pour tous les héros du joueur IA spécifié.

**Syntaxe**

`SetAIPlayerAttractor(`*objectName*`, `*playerID*`, `*priority*`);`

**Description**

Cette fonction modifie la priorité de l'objet *objectName* comme attracteur pour tous les héros du joueur *playerID* contrôlé par l'IA.

*objectName* – nom de l'objet. Cet objet doit être immobile.

*playerID* – l'id du joueur contrôlé par l'IA

*priority* – priorité de l'attracteur (entre –1 et 2)

- -1 : la priorité baisse énormément
- 0 : la priorité est inchangée (l'appel à la fonction est donc inutile)
- 1 : la priorité augmente beaucoup, mais l'IA ne risquera pas ses héros pour prendre le contrôle de l'objet.
- 2 : prendre le contrôle de l'objet devient prioritaire (comme un objectif de victoire).

## SetCombatLight

Modifie l'illumination de toutes les arènes de combat de la carte en cours.

**Syntaxe**

`SetCombatLight(lightName);`

**Description**

Cette fonction change la luminosité ambiante, avec le paramètre spécifié dans la commande, de toutes les arènes de combat de carte en cours.

*lightName* – référence de la nouvelle source lumineuse dans la base de ressources.

## SetHeroCombatScript

Détermine le script de combat d'un héros.

**Syntaxe**

`SetHeroCombatScript(heroName, scriptName);`

**Description**

Cette fonction permet de spécifier un script qui sera lancé lorsque le héros *heroName* entre en combat. Ce script se lancera uniquement si le héros est sous le contrôle d'un joueur de l'IA et qu'aucun script n'est défini pour le lieu où le combat prend place (château, garnison, etc.) Ce script est réinitialisé lorsque le joueur perd le héros.

*heroName* – id du héros.

*scriptName* – nom du script.

## SetHeroLootable

Détermine s'il est possible de prendre les artéfacts du héros en le battant.

**Syntaxe**

`SetHeroLootable(heroName, enable);`

**Description**

Le paramètre *enable* détermine si les artéfacts du héros sont donnés au héros qui l'a vaincu (*not nil*) ou s'ils sont gardés par le héros vaincu (*nil*).

*heroName* – id du héros

*enable* – détermine si les artéfacts du héros lui sont pris après une défaite.

## SetAmbientLight

Change la luminosité ambiante du terrain spécifié de la carte.

**Syntaxe**

```
SetAmbientLight(floorID, lightName, fade = false, time = 1);
```

**Description**

Cette fonction initialise une nouvelle source de lumière ambiante à l'étage *floorID* de la carte. La liste des sources de lumière disponible est spécifiée dans la description de la carte.

*floorID* – numéro de l'étage où la luminosité va être modifiée.

*lightName* – nom de la source de lumière ambiante

*fade* – détermine si l'éclairage doit changer progressivement (*fade = true*) ou immédiatement (*fade = false*) (l'ancienne et la nouvelle sources de lumière ne doivent avoir qu'une différence de couleur).

*time* – la durée de la transition (si *fade = true*).

## SetObjectEnabled

Active/désactive l'interaction standard de l'objet avec les héros.

**Syntaxe**

```
SetObjectEnabled(objectName, enable);
```

**Description**

Cette commande permet d'activer (*enable = true*) ou désactiver (*enable = false*) l'interaction standard de l'objet avec les héros. Un objet désactivé ne pourra plus interagir, sauf si une fonction spécifique lui a été associée par `Trigger`.

*objectName* – nom de l'objet

*enable* – *true* pour activer l'interaction standard, *false* pour la désactiver.

## SetObjectiveProgress

Fixe la progression de l'objectif.

**Syntaxe**

```
SetObjectiveProgress(objectiveName, step, playerID = PLAYER_1);
```

**Description**

Cette fonction permet de spécifier la progression de l'objectif *objectiveName*. Pour les objectifs communs à tous les joueurs, le paramètre *playerID* spécifie à quel joueur s'applique la fonction (par défaut le joueur 1). Pour les objectifs spécifiques à un joueur, *playerID* est ignoré.

`SetObjectiveProgress` permet de gérer la progression des objectifs déclarés comme manuels. Le nombre d'étapes des objectifs manuels est défini dans l'éditeur à la création de la carte.

*objectiveName* – nom de l'objectif

*step* – étape de la progression

*playerID* – id du joueur pour qui la progression doit être modifiée

## SetObjectiveState

Change le statut de l'objectif.

**Syntaxe**

```
SetObjectiveState(objectiveName, state, playerID = PLAYER_1);
```

**Description**

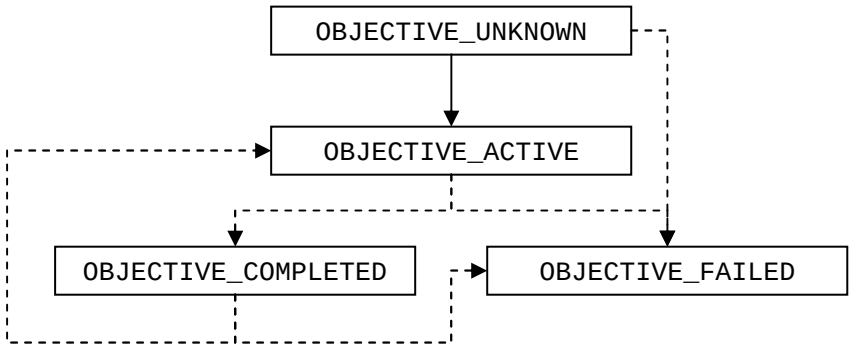
Cette fonction change le statut de l'objectif *objectiveName* pour le joueur spécifié. Pour un objectif spécifique à un joueur, le paramètre *playerID* est ignoré.

*objectiveName* – nom de l'objectif

*state* – nouveau statut de l'objectif.

*playerID* – id du joueur pour qui le statut de l'objectif est modifié.

Transitions possibles :




Les transitions en pointillés ne sont possibles que pour les objectifs contrôlés manuellement.

Note : si l'objectif a le statut `OBJECTIVE_ACTIVE`, il devient automatiquement visible par le joueur.

## SetObjectiveVisible

Affiche/cache un objectif dans l'interface du joueur spécifié.

**Syntaxe**

```
SetObjectiveVisible(objectiveName, enable, playerID = PLAYER_1);
```

**Description**

L'objectif *objectiveName* sera affiché (*enable = true*) ou caché (*enable = false*) dans l'interface du joueur. Pour un objectif spécifique à un joueur, *playerID* est ignoré.

*objectiveName* – nom de l'objectif

*enable* – *true* pour afficher l'objectif, *false* pour le cacher.

*playerID* – id du joueur

## SetObjectFlashLight

Attache une source de lumière sur un objet de la carte d'aventure.

**Syntaxe**

```
SetObjectFlashLight(objectName, lightName);
```

**Description**

La commande attache une source de lumière supplémentaire sur l'objet sélectionné de la carte d'aventure. La liste des sources de lumière qui peuvent être utilisées est spécifiée dans la description de carte (dans le champ *resources->pointLights*).

*objectName* – nom de l'objet

*lightName* – nom de la source de lumière

## SetObjectOwner

Change le propriétaire de l'objet spécifié de la carte d'aventure.

**Syntaxe**

```
SetObjectOwner(objectName, playerID);
```

**Description**

La commande change le propriétaire de l'objet de la carte d'aventure. Elle ne peut être utilisée que pour des objets statiques qui peuvent avoir un propriétaire, ainsi que les héros. Si le héros est dans la réserve d'un château, il en sera retiré. Les héros ne peuvent pas être neutres.

*objectName* – nom de l'objet

*playerID* – id du joueur qui va être le nouveau propriétaire de l'objet.

## SetObjectPosition

Déplace un objet

**Syntaxe**

```
SetObjectPosition(objectName, x, y, floor = -1);
```

**Description**

Cette fonction déplace objet à la position spécifiée par les coordonnées *x*, *y* et l'étage *floor*. Si le paramètre *floor* n'est pas spécifié, c'est l'étage où se trouve l'objet qui est utilisé par défaut. Seuls les objets mobiles peuvent être déplacés. Si la position cible est inaccessible, occupé par un autre objet, la fonction génère une erreur.

*objectName* – nom de l'objet

*x, y* – les coordonnées de la position cible

*floor* – le numéro de l'étage (par défaut celui où se trouve l'objet).

## SetPlayerResource

Fixe la quantité d'une ressource d'un joueur spécifique.

**Syntaxe**

```
SetPlayerResource(player, resourceKind, quantity);
```

**Description**

Fixe à *quantity* la valeur la ressource *resourceKind* pour le joueur *player*.

La commande fonctionne pour les joueurs actifs et joueurs ayant déjà perdu. Par contre, modifier les ressources d'un joueur n'ayant pas du tout participé à la partie génère une erreur.

*player* – id du joueur

*resourceKind* – id de la ressource (WOOD, ORE, MERCURY, CRYSTAL, SULFUR, GEM, GOLD).

*quantity* – quantité (entier positif).

## SetPlayerStartResources

Fixe les quantités initiales des ressources du joueur spécifié.

**Syntaxe**

```
SetPlayerStartResources(player, wood, ore, mercury, crystal, sulfur, gem, gold);
```

**Description**

Fixe les quantités initiales des ressources pour le joueur *player*. La commande est identique à SetPlayerResource pour chaque type de ressource, mais elle n'est pas utilisable en cours de jeu.

*player* – id du joueur

*wood*, *ore*, *mercury*, *crystal*, *sulfur*, *gem* et *gold* – valeur initiale de la ressource (entier positif)

## SetRegionBlocked

Bloque/débloque l'accès à une région pour les déplacements des héros.

**Syntaxe**

```
SetRegionBlocked(regionName, status, playerID = -1);
```

**Description**

Cette fonction bloque (*status = true*) ou débloque (*status = false*) la région pour les déplacements des héros. Si le paramètre *playerID* est indiqué, la région est bloquée uniquement pour les héros de ce joueur, sinon la région est bloquée pour tous les joueurs.

*regionName* – nom de la région

*status* – *true* pour bloquer, *false* pour débloquer la région

*playerID* – id du joueur (par défaut à –1, qui correspond à tous les héros)

## SetTownBuildingLimitLevel

Fixe la limitation du niveau de construction d'un bâtiment de la ville.

**Syntaxe**

```
SetTownBuildingLimitLevel(townName, buildingID, limit);
```

**Description**

La fonction fixe le niveau limite que pourra atteindre le bâtiment *buildingID* dans la ville *townName*. Le niveau 0 empêche complètement la construction du bâtiment. *limit* ne doit pas être inférieur au niveau actuel du bâtiment.

*townName* – nom de la ville

*buildingID* – id du type de bâtiment

*limit* – niveau maximal

## SetWarfogBehaviour

Active ou désactive la dissipation du brouillard de guerre par les héros.

**Syntaxe**

```
SetWarfogBehaviour(onLand, onSea);
```

**Description**

Si le paramètre *onLand/onSea* vaut 1, le brouillard de guerre se dissipe lorsque les héros se déplacent sur terre (*onLand*) et sur mer (*onSea*), sinon le brouillard de guerre ne se dissipe pas. Il est impossible de pénétrer dans le brouillard de guerre.

## ShowFlyingSign

Affiche un message qui s'envole au-dessus de l'objet.

**Syntaxe**

```
ShowFlyingSign(messageName, objectName, targetPlayerID = -1, time = 1.0);
```

**Description**

Cette fonction affiche le message *messageName* flottant au-dessus de l'objet *objectName*. Le message peut être vu par le joueur *targetPlayerID* seulement, ou par tous si *targetPlayerID* vaut -1. Le temps d'affichage est donné par *time* (par défaut à 1 seconde).

*messageName* – nom du message dans la base des ressources

*objectName* – nom de l'objet au-dessus duquel le message doit s'afficher

*targetPlayerID* – id du joueur qui doit voir le message (-1 pour tous les joueurs)

*time* – temps d'affichage du message (par défaut à 1 seconde)

## SiegeTown

Démarre le siège d'une ville

**Syntaxe**

```
SiegeTown(heroName, townName, areaName="");
```

**Description**

Cette fonction démarre le siège de la ville *townName* par le héros *heroName*. La ville *townName* peut être désignée par son nom sur la carte ou son identifiant. Le combat se déroule comme un siège normal, en tenant compte des paramètres actuels de la ville, comme la garnison, le héros positionné dans la ville, le script de combat, etc. Le paramètre *arenaName* permet de remplacer l'arène de combat standard par n'importe quelle autre.

*heroName* – id du héros qui attaque la ville

*townName* – ville assiégée (son nom ou son identifiant)

*arenaName* – référence de l'arène où se déroulera le combat. Par défaut c'est l'arène standard.

## StartCombat

Démarre un combat avec des paramètres spécifiques.

**Syntaxe**

```
StartCombat(heroName, enemyHeroName, creatureCount, creatureType[1],
 creatureAmount[1], ..., combatScriptName = nil, combatFinishTrigger = nil,
 arenaName = nil, allowQuickCombat = nil);
```

**Description**

Commence un combat avec le héros *heroName*, contre le héros *enemyHeroName* (indiquer *nil* pour une armée sans héros). L'armée du héros ennemi est ignorée, et remplacée par les créatures spécifiées. Il est possible d'indiquer un script de combat *combatScriptName*, ainsi qu'une fonction *combatFinishTrigger* à lancer à la fin du combat.

*heroName* – id du héros qui va combattre

*enemyHeroName* – id du héros ennemi (*nil* pour aucun héros)

*creatureCount* – nombre de pelotons de créatures à combattre. Indique combien de couples *creatureType*, *creatureAmount* le suivent dans la liste des paramètres.

*creatureType* – type de la créature

*creatureAmount* – nombre de créatures du peloton

*combatScriptName* – nom du script qui sera lancé au début du combat (*nil* pour aucun).

*combatFinishTrigger* – nom de la fonction qui sera lancée à la fin du combat (*nil* pour aucun). Cette fonction doit accepter deux paramètres : l'identifiant du héros attaquant et le résultat du combat (*nil* si l'attaquant a perdu, *not nil* s'il a gagné).

*areaName* – référence de l'arène où le combat doit se dérouler (*nil* pour conserver l'arène par défaut du terrain où se trouve le héros).

*allowQuickCombat* – Si égal à *nil*, alors le combat ne pourra pas être résolu automatiquement (combat rapide). Sinon, le déroulement du combat dépendra des options sélectionnées.

## StartCutScene

Lance une cinématique

**Syntaxe**

```
StartCutScene(cutSceneName, callBack = "",  saveName = "");
```

**Description**

Cette fonction lance une cinématique. Elle peut indiquer une fonction à appeler juste après la cinématique, et déclencher une sauvegarde juste avant le début de la cinématique.

*cutSceneName* – référence à la cinématique dans la base des ressources.

*callBack* – nom de la fonction à lancer après la cinématique.

*saveName* – nom du fichier de sauvegarde (voir la fonction `save`).

## StartDialogScene

Lance une scène de dialogue

**Syntaxe**

`StartDialogScene(`*`dialogSceneName, callBack = "",  saveName = ""`*`);`

**Description**

Cette fonction lance la scène de dialogue spécifiée. Elle peut indiquer une fonction à appeler juste après la scène, et déclencher une sauvegarde juste avant le début de la scène.

*`dialogSceneName`* – référence à la scène dans la base des ressources

*`callBack`* – nom de la fonction à lancer à la fin de la scène.

*`saveName`* – nom du fichier de sauvegarde (voir la fonction `save`).

**Exemple :**

```
StartDialogScene("/DialogScenes/C6/M4/C1/DialogScene.xdb#xpointer(/DialogScene)");
```

## StopPlaySound

Arrête la répétition du son

**Syntaxe**

`StopPlaySound(`*`loopingSoundID`*`);`

**Description**

Cette fonction arrête la répétition du son spécifié.

*`loopingSoundID`* – ID du son, comme celui retourné par les fonctions `Play2DSound` et `Play3DSound` pour les sons en boucle.

## TeachHeroSpell

Donne le sort spécifié au héros

**Syntaxe**

`TeachHeroSpell(`*`heroName, spell`*`);`

**Description**

Cette fonction essaie de donner un sort au héros. Si le héros connaît déjà ce sort, la fonction retourne *`nil`*, sinon *`not nil`*.

*`heroName`* – id du héros

*`spell`* – ID du sort

## TransformTown

Change le type de la ville

**Syntaxe**

`TransformTown(`*`townName, type`*`);`

**Description**

Cette fonction change le type de la ville. Toutes les constructions dans la ville sont perdues.

*`townName`* – nom de la ville

*type* – nouveau type de la ville. Peut prendre les valeurs suivantes :

- TOWN_HEAVEN,
- TOWN_PRESERVE,
- TOWN_ACADEMY,
- TOWN_DUNGEON,
- TOWN_NECROMANCY,
- TOWN_INFERNO

## Trigger

Active/désactive la fonction déclenchée dépendant d'un événement.

**Syntaxe**

`Trigger(`*`triggerType, ..., functionName`*`);`

**Description**

Si le paramètre *functionName* n'est pas égal à *nil*, la fonction indiquée est liée avec l'événement spécifié. Sinon, l'association est supprimée, et l'événement ne déclenchera pas de fonction particulière. Pour chaque événement, il ne peut y avoir qu'une seule fonction déclenchée. Associer une autre fonction à l'événement remplace la précédente.

*triggerType* – type d'événement. Peut prendre une des valeurs suivantes :

- NEW_DAY_TRIGGER – début d'un nouveau jour
- WAR_FOG_ENTER_TRIGGER – un héros entre dans le brouillard de guerre (le comportement du brouillard de guerre est géré par la fonction SetWarfogBehaviour)
- PLAYER_ADD_HERO_TRIGGER – le joueur a un nouveau héros
- PLAYER_REMOVE_HERO_TRIGGER – le joueur perd un héros
- OBJECTIVE_STATE_CHANGE_TRIGGER – le statut d'un objectif est changé
- OBJECT_CAPTURE_TRIGGER – un objet a été pris
- OBJECT_TOUCH_TRIGGER – l'objet interactif a été activé
- TOWN_HERO_DEPLOY_TRIGGER – un héros sort de la ville
- REGION_ENTER_AND_STOP_TRIGGER – un héros entre et s'arrête dans la région
- REGION_ENTER_WITHOUT_STOP_TRIGGER – un héros entre dans la région
- HERO_LEVELUP_TRIGGER – le héros gagne un niveau

Certains événements acceptent des paramètres supplémentaires (avant *functionName*), permettant de préciser les conditions du déclenchement :

- NEW_DAY_TRIGGER et WAR_FOG_ENTER_TRIGGER – aucun
- PLAYER_ADD_HERO_TRIGGER et PLAYER_REMOVE_HERO_TRIGGER – ID du joueur
- OBJECTIVE_STATE_CHANGE_TRIGGER – nom de l'objectif
- OBJECT_CAPTURE_TRIGGER et OBJECT_TOUCH_TRIGGER – nom de l'objet
- TOWN_HERO_DEPLOY_TRIGGER – nom de la ville
- REGION_ENTER_AND_STOP_TRIGGER et REGION_ENTER_WITHOUT_STOP_TRIGGER – nom de la région
- HERO_LEVELUP_TRIGGER – id du héros

*functionName* – le nom de la fonction déclenchée. Lorsqu'elle est appelée, elle reçoit les paramètres suivants :

- NEW_DAY_TRIGGER – aucun
- WAR_FOG_ENTER_TRIGGER – id du héros
- PLAYER_ADD_HERO_TRIGGER – id du héros

- `PLAYER_REMOVE_HERO_TRIGGER` – id du héros
- `OBJECTIVE_STATE_CHANGE_TRIGGER` – ID du joueur dont l'objectif change de statut
- `OBJECT_CAPTURE_TRIGGER` – ancien propriétaire de l'objet, nouveau propriétaire de l'objet, id du héros qui l'a capturé (*`nil`* si aucun), et nom de l'objet lui-même
- `OBJECT_TOUCH_TRIGGER` – id du héros et nom de l'objet
- `TOWN_HERO_DEPLOY_TRIGGER` – id du héros
- `REGION_ENTER_AND_STOP_TRIGGER et REGION_ENTER_WITHOUT_STOP_TRIGGER` – id du héros
- `HERO_LEVELUP_TRIGGER` – aucun

## UnblockGame

Débloque l'interface du joueur et le fonctionnement de l'IA.

**Syntaxe**

```
UnblockGame();
```

**Description**

La commande débloque l'interface du joueur, le fonctionnement de l'IA, et aussi la gestion de la caméra qui a été bloquée par la fonction `BlockGame`.

## UnreserveHero

Libère un héros réservé.

**Syntaxe**

```
UnreserveHero(heroName);
```

**Description**

Le héros n'est plus réservé à un joueur et peut maintenant apparaitre dans les tavernes des autres joueurs au début de chaque début de semaine.

Si le héros est vivant lorsque cette fonction est appelée, le héros n'est plus réservé au joueur mais le joueur ne perd pas le héros.
Si le héros est mort quand la fonction est appelée, le héros devient immédiatement disponible pour tous les joueurs sauf pour l'ancien propriétaire, parce qu'il est considéré comme ayant perdu une bataille.

*`heroName`* – id du héros

## Win

Le joueur humain gagne la partie lorsque cette fonction est lancée

**Syntaxe**

```
Win();
```

**Description**

Cette fonction ne peut être lancée qu'en mode 1 joueur. Elle génère une erreur sinon. Lorsqu'elle est appelée le joueur humain gagne la partie immédiatement, et les joueurs de l'IA perdent.

# EN COMBAT

## Prepare

Préparation pour le combat, avant la phase tactique.

**Syntaxe**

```
Prepare();
```

**Description**

Cette fonction est appelée avant que les joueurs commencent à placer leurs troupes sur-le-champ de bataille. L'organisation des troupes ne commencera après qu'à la fin cette fonction.

## Start

Début d'une bataille, après la phase tactique.

**Syntaxe**

```
Start();
```

**Description**

Cette fonction est appelée après que les joueurs ont placé leurs troupes sur le champ de bataille et immédiatement avant que la bataille commence. La bataille ne commencera qu'à la fin de cette fonction.

## IsHuman

Détermine si un joueur humain contrôle le camp spécifique.

**Syntaxe**

```
IsHuman(side);
```

**Description**

Cette fonction retourne `not nil` si le camp spécifié appartient à un joueur humain (sans prendre en compte le fait que le joueur contrôle les actions ou non), sinon elle retourne `nil`.

*side* – id du camp (`ATTACKER=0 ou DEFENDER=1`)

## IsComputer

Détermine si un joueur IA contrôle le camp spécifique.

**Syntaxe**

```
IsComputer(side);
```

**Description**

Cette fonction retourne `not nil` si le camp spécifié appartient à un joueur de l'IA sinon elle retourne `nil`.

*side* – id du camp (`ATTACKER=0 ou DEFENDER=1`)

## SetControlMode

Initialise le mode de contrôle du combat pour un joueur humain

**Syntaxe**

`SetControlMode(`*`side, mode`*`);`

**Description**

La commande active/désactive le mode de combat automatique et active/désactive la possibilité au joueur humain de passer en mode manuel. Le camp spécifié doit être contrôlé par un humain.

*`side`* – id du camp (`ATTACKER=0 ou DEFENDER=1`)

*`mode`* – id du mode :

- `MODE_NORMAL` – le combat automatique est accessible, au choix du joueur
- `MODE_MANUAL` – le combat automatique est bloqué en mode désactivé
- `MODE_AUTO` – le combat automatique est bloqué en mode activé

## EnableAutoFinish

Active/désactive le mécanisme standard de vérification du résultat du combat.

**Syntaxe**

`EnableAutoFinish(`*`enable`*`);`

**Description**

Le paramètre envoyé à la commande détermine si le combat finit quand un des deux camps n'a plus de troupe. Si le paramètre est à *`nil`*, le combat ne finira pas. La vérification est activée par défaut.

## Finish

Termine le combat

**Syntaxe**

`Finish(`*`winnerSide`*`);`

**Description**

La commande termine le combat, et le camp *`winnerSide`* est considéré comme gagnant le combat.

*`winnerSide`* – id du camp qui gagne le combat (`ATTACKER=0 ou DEFENDER=1`)

## GetAttackerHero

Retourne l'identifiant de combat du héros qui attaque.

**Syntaxe**

`GetAttackerHero();`

**Description**

L'identifiant est '`attacker-hero`', ou *`nil`* s'il n'y a pas de héros.

## GetAttackerCreatures

Retourne les créatures du camp attaquant

**Syntaxe**

```
GetAttackerCreatures();
```

**Description**

Cette fonction retourne une liste de nom des créatures du camp attaquant

## GetAttackerWarMachines

Retourne les créatures du camp attaquant

**Syntaxe**

```
GetAttackerWarMachines();
```

**Description**

Cette fonction retourne une liste de nom des machines de guerre du camp attaquant

## GetAttackerWarMachine

Retourne le nom de la machine guerre du camp attaquant du type spécifié

**Syntaxe**

`GetAttackerWarMachine(`*`type`*`);`

**Description**

Cette fonction retourne le nom de la machine de guerre du camp attaquant du type spécifié, ou *`nil`* si le héros n'a pas cette machine de guerre.

*`Type`* – le type de la machine de guerre ; on peut utiliser les variables globales suivantes :

- `WAR_MACHINE_BALLISTA` – la baliste
- `WAR_MACHINE_CATAPULT` – la catapulte
- `WAR_MACHINE_FIRST_AID` – la tante de soin
- `WAR_MACHINE_AMMO_CART` – le chariot de munition

## GetDefenderHero

Retourne l'identifiant de combat du héros qui défend

**Syntaxe**

```
GetDefenderHero();
```

**Description**

L'identifiant est '`defender-hero`', ou *`nil`* s'il n'y a pas de héros.

## GetDefenderCreatures

Retourne les créatures du camp qui défend

**Syntaxe**

```
GetDefenderCreatures();
```

**Description**

Cette fonction retourne une liste de nom des créatures du camp qui défend

## GetDefenderWarMachines

Retourne les créatures du camp qui défend

**Syntaxe**

```
GetDefenderWarMachines();
```

**Description**

Cette fonction retourne une liste de nom des machines de guerre du camp qui défend

## GetAttackerWarMachine

Retourne le nom de la machine guerre du camp qui défend du type spécifié

**Syntaxe**

```
GetDefenderWarMachine(type);
```

**Description**

Cette fonction retourne le nom de la machine de guerre du camp qui défend du type spécifié, ou `nil` si le héros n'a pas cette machine de guerre.

*Type* – le type de la machine de guerre ; on peut utiliser les variables globales suivantes :

- `WAR_MACHINE_BALLISTA` – la baliste
- `WAR_MACHINE_CATAPULT` – la catapulte
- `WAR_MACHINE_FIRST_AID` – la tante de soin
- `WAR_MACHINE_AMMO_CART` – le chariot de munition

## GetDefenderBuidings

Retourne les bâtiments du camp qui défend

**Syntaxe**

```
GetDefenderBuidings();
```

**Description**

Cette fonction retourne la liste de noms des bâtiments du camp qui défend

## GetDefenderBuiding

Retourne le nom du bâtiment spécifié du camp qui défend.

**Syntaxe**

```
GetDefenderBuiding(type);
```

**Description**

Cette fonction retourne le nom du bâtiment spécifique du camp qui défend, ou *nil* s'il n'existe pas.

*type* – le type de bâtiment, on peut utiliser les variables globales suivantes :

- BUILDING_WALL – le mur
- BUILDING_GATE – la porte
- BUILDING_LEFT_TOWER – la tour de gauche
- BUILDING_BIG_TOWER – la tour centrale
- BUILDING_RIGHT_TOWER – la tour de droite
- BUILDING_MOAT – le fossé

## IsAttacker

Détermine si l'objet appartient au camp attaquant.

**Syntaxe**

`IsAttacker(unitName);`

**Description**

Cette fonction retourne *not nil* si l'objet appartient au camp attaquant, sinon *nil*.

*unitName* – nom de l'objet

## IsHero

Détermine si l'objet est un héros

**Syntaxe**

`IsHero(unitName);`

**Description**

Cette fonction retourne *not nil* si l'objet est un héros sinon *nil*.

*unitName* – nom de l'objet

## IsCreature

Détermine si l'objet est une créature

**Syntaxe**

`IsCreature(unitName);`

**Description**

Cette fonction retourne *not nil* si l'objet est une créature sinon *nil*.

*unitName* – nom de l'objet

## IsWarMachine

Détermine si l'objet est une machine de guerre

**Syntaxe**

`IsWarMachine(unitName);`

**Description**

Cette fonction retourne *not nil* si l'objet est une machine de guerre sinon *nil*.

*unitName* – nom de l'objet

## IsBuilding

Détermine si l'objet est un bâtiment

**Syntaxe**

`IsBuilding(unitName);`

**Description**

Cette fonction retourne *not nil* si l'objet est un bâtiment sinon *nil*.

*unitName* – nom de l'objet

## GetHeroName

Retourne l'identifiant du héros

**Syntaxe**

`GetHeroName(unitName);`

**Description**

Cette fonction retourne l'identifiant du héros (pas l'identifiant de combat).

*unitName* – nom de l'objet

## GetCreatureType

Retourne le type de la créature

**Syntaxe**

`GetCreatureType(unitName);`

**Description**

Cette fonction retourne le type de la créature.

*unitName* – nom de l'objet

## GetCreatureNumber

Retourne le nombre de créature dans le peloton

**Syntaxe**

`GetCreatureNumber(unitName);`

**Description**

Cette fonction retourne le nombre de créatures dans le peloton.

*unitName* – nom de l'objet

## GetWarMachineType

Retourne le type de la machine de guerre

**Syntaxe**

`GetWarMachineType(`*unitName*`);`

**Description**

Cette fonction retourne le type de la machine de guerre :

- `WAR_MACHINE_BALLISTA` – la baliste
- `WAR_MACHINE_CATAPULT` – la catapulte
- `WAR_MACHINE_FIRST_AID` – la tente de soin
- `WAR_MACHINE_AMMO_CART` – le chariot de munition

*unitName* – nom de l'objet

## GetBuildingType

Retourne le type du bâtiment

**Syntaxe**

`GetBuildingType(`*unitName*`);`

**Description**

Cette fonction retourne le type du bâtiment :

- `BUILDING_WALL` – le mur
- `BUILDING_GATE` – la porte
- `BUILDING_LEFT_TOWER` – la tour de gauche
- `BUILDING_BIG_TOWER` – la tour centrale
- `BUILDING_RIGHT_TOWER` – la tour de droite
- `BUILDING_MOAT` – le fossé

*unitName* – nom de l'objet

## GetUnitPosition

Retourne les coordonnées de l'objet.

**Syntaxe**

`GetUnitPosition(`*unitName*`);`

**Description**

Cette fonction retourne les coordonnées *x* et *y* de la position de l'objet.

*unitName* – nom de l'objet

## AddCreature

Ajoute une créature du type spécifié à un des camps

**Syntaxe**

`AddCreature(side, type, number, x = -1, y = -1);`

**Description**

Cette fonction ajoute un groupe de *number* créatures de type *type* pour le camp *side*. Si les paramètres *x* et *y* sont indiqués, la créature apparaît dans la case libre le plus proche des coordonnées (*x,y*) ; sinon elle apparaît dans une case aléatoire de l'arène. Il est aussi possible de ne donner qu'une seule coordonnée de la case (par exemple seulement *x* ou seulement *y*). Les créatures qui sont ajoutées par cette commande restent dans l'armée du héros après le combat (s'il y a assez de place pour elles).

*side* – camp pour qui la créature est invoquée ; peut prendre une de ces deux valeurs :

- `ATTACKER` – le camp qui attaque
- `DEFENDER` – le camp qui défend

*type* – type de créature

*number* – nombre de créatures

*x, y* – coordonnées où apparaîtra la créature (-1 pour des coordonnées au hasard)

## EnableCinematicCamera

Active/désactive la caméra de cinématique dans le combat.

**Syntaxe**

`EnableCinematicCamera(enable);`

**Description**

Cette fonction ne concerne que pour le combat en cours. La caméra peut maintenant passer en mode cinématique seulement si cela est autorisé dans le paramétrage et non interdit par un script.

# TUTORIEL

## IsTutorialItemEnable

Indique si un élément du tutoriel va être affiché.

**Syntaxe**

`IsTutorialItemEnable(`*`name`*`);`

**Description**

Cette fonction indique si un élément du tutoriel va être affiché.

*`name`* – nom de l'élément du tutoriel

## IsTutorialMessageBoxOpen

Indique si une fenêtre du tutoriel est ouverte.

**Syntaxe**

`IsTutorialMessageBoxOpen();`

**Description**

Cette fonction retourne *`true`* s'il y a une fenêtre du tutoriel ouverte par la fonction `TutorialMessageBox`.

## TutorialActivateHint

Autorise l'affichage d'une fenêtre du tutoriel.

**Syntaxe**

`TutorialActivateHint(`*`stringID`*`);`

**Description**

La fenêtre du tutoriel d'identifiant *`stringID`* sera affichée au déclenchement de l'événement associé.

## TutorialMessageBox

Affiche une fenêtre avec le texte du tutoriel

**Syntaxe**

`TutorialMessageBox(`*`stringID`*`);`

**Description**

Cette fonction trouve l'élément *`stringID`* dans *`UIConst.tutorialOptions.hints`* et affiche une fenêtre avec le texte associé.

## TutorialSetBlink

Active/désactive la mise en relief de l'élément

**Syntaxe**

`TutorialSetBlink(`*`stringID`*`, `*`turnOn`*`);`

**Description**

La mise en relief de l'élément du tutoriel d'identifiant *`stringID`* est activée/désactivée en fonction de la valeur du paramètre *`turnOn`* (*`true/false`*).

# EN VILLE

## HeroHired

Déclenché lorsqu'un héros est engagé.

**Syntaxe**

```
HeroHired(name);
```

**Description**

Cette fonction est appelée lorsqu'un joueur engage un héros.

## CreatureHired

Déclenché lorsqu'une créature est engagée.

**Syntaxe**

```
CreatureHired(type, number);
```

**Description**

Cette fonction est appelée lorsqu'un joueur engage des créatures