

Advanced Tools for Mission Editor

ATME - Manuel de référence pour le développement de modules

Auteur Sunski34

Traductions et tests généraux Snowsniper

Table des matières

| | |
|---|-----------|
| Introduction..... | 14 |
| Le concept de callback | 15 |
| Le concept d'événements ATME..... | 16 |
| L'éditeur de mission DCS et ATME | 18 |
| Définitions et notations | 19 |
| Restrictions & limitations..... | 19 |
| La création de modules utilisateur ATME | 20 |
| Module 1 : module utilisateur : « Hello world » | 20 |
| Code lua..... | 20 |
| Explications..... | 21 |
| Tests et essais..... | 22 |
| Exercice..... | 25 |
| Module 1 : Ajout d'un affichage dynamique | 25 |
| Code lua..... | 25 |
| Explications..... | 26 |
| Tests et essais..... | 26 |
| Exercice..... | 27 |
| Module 1 : remplacement de l'affichage dynamique par un menu spécifique | 27 |
| Code lua..... | 27 |
| Explications..... | 28 |
| Tests et essais..... | 28 |
| Exercice..... | 30 |
| Module 1 : Ajout d'un menu visible selon certains cas | 30 |
| Code lua..... | 30 |
| Explications..... | 32 |
| Tests et essais..... | 32 |
| Exercice..... | 33 |
| Module 1 : Ajout d'un sous-menu et d'une variable spécifique aux joueurs | 34 |

| | |
|--|-----------|
| Code lua..... | 34 |
| Explications..... | 35 |
| Tests et essais..... | 36 |
| Exercice..... | 36 |
| Module 2 : Interactions DCS et gestion de triggers utilisateur..... | 37 |
| Code lua..... | 37 |
| Explications..... | 38 |
| Tests et essais..... | 39 |
| Exercice..... | 40 |
| Module 2 : Gestion de triggers utilisateur relatifs à un flag | 40 |
| Code lua..... | 41 |
| Explications..... | 42 |
| Tests et essais..... | 42 |
| Exercice..... | 42 |
| Module 2 : Gestion de triggers utilisateur avec une durée..... | 43 |
| Code lua..... | 43 |
| Explications..... | 44 |
| Tests et essais..... | 45 |
| Exercice..... | 45 |
| Module 3 : Données de mission pour les groupes et création dynamique de groupes | 46 |
| Code lua..... | 46 |
| Explications..... | 47 |
| Tests et essais..... | 47 |
| Exercice..... | 49 |
| Module 4 : Gestion des Core Events | 50 |
| Code lua..... | 50 |
| Explications..... | 51 |
| Tests et essais..... | 51 |
| Exercice..... | 52 |
| Module 4 : Gestion des pickupzones..... | 55 |
| Code lua..... | 55 |
| Explications..... | 56 |
| Tests et essais..... | 56 |
| Exercice..... | 57 |

| | |
|---|-----------|
| Module 4 : Gestion multilangue | 57 |
| Code lua..... | 58 |
| Explications..... | 60 |
| Tests et essais..... | 60 |
| Module 5 : Gestion des courses..... | 62 |
| Code lua..... | 62 |
| Explications..... | 64 |
| Tests et essais..... | 64 |
| Fonctions globales de la table ATME | 67 |
| ATME.arePointsVisible(pA, pB) | 67 |
| ATME.convertFtstoM(distance) | 67 |
| ATME.convertKphtoMps(speed) | 68 |
| ATME.convertLLToMGRS(latitude, longitude) | 68 |
| ATME.convertLLToPoint(latitude, longitude, altitude) | 69 |
| ATME.convertMGRSToLL(tableMGRS) | 69 |
| ATME.convertMpstoKph(speed) | 70 |
| ATME.convertMpstoNds(speed) | 70 |
| ATME.convertMtoFts(distance) | 70 |
| ATME.convertMtoNM(distance) | 71 |
| ATME.convertNdstoMps(speed) | 71 |
| ATME.convertNMtoM(distance) | 71 |
| ATME.convertPointToLL(point) | 72 |
| ATME.convertToDMS(angle, secDec)..... | 72 |
| ATME.convertToHMSd(seconds)..... | 73 |
| ATME.convertToPoint3D(point) | 73 |
| ATME.displayForAll(text, duration)..... | 73 |
| ATME.displayForCoalition(coalitionName, text, duration) | 74 |
| ATME.displayForCountry(countryName, text, duration) | 74 |
| ATME.explode(point, power) | 75 |
| ATME.getAIUnits() | 75 |
| ATME.getFlag(id) | 75 |
| ATME.getGroups() | 76 |
| ATME.getLanguage() | 76 |
| ATME.getPlayers() | 76 |

| | |
|---|-----------|
| ATME.getPointAltitude(point)..... | 77 |
| ATME.getSurfaceAltitude(point) | 77 |
| ATME.getTheatre() | 77 |
| ATME.getTime()..... | 78 |
| ATME.getWindAzimuth(point3D, withTurbulence) | 78 |
| ATME.getWindHSpeed(point3D, withTurbulence) | 78 |
| ATME.getWindSpeed(point3D, withTurbulence)..... | 79 |
| ATME.getWindVector(point3D, withTurbulence) | 79 |
| ATME.getDCSZone(zoneName)..... | 80 |
| ATME.isObjectClass(object, class) | 80 |
| ATME.isPoint(tableToVerify) | 81 |
| ATME.isStaticObject(objectName) | 81 |
| ATME.isSurfaceLand(point) | 81 |
| ATME.isSurfaceRoad(point) | 82 |
| ATME.isSurfaceRunway(point)..... | 82 |
| ATME.isSurfaceWater(point)..... | 82 |
| ATME.isUnitControllable(unitName)..... | 83 |
| ATME.loopTransmission(file, point, modulation, frequency, power)..... | 83 |
| ATME.rotationH(point, center, angle)..... | 84 |
| ATME.run(language)..... | 85 |
| ATME.scalaireHFromPoints(pA, pB, pC) | 85 |
| ATME.sendTransmissionOnce(file, point, modulation, frequency, power) | 86 |
| ATME.setDebugLevel(level)..... | 86 |
| ATME.setF10groupIDAlphaOrder() | 86 |
| ATME.setF10AlphaOrder() | 87 |
| ATME.setFlag(id, value) | 87 |
| ATME.soundForAll(file) | 87 |
| ATME.soundForCoalition(coalition, file) | 88 |
| ATME.soundForCountry(countryName, file)..... | 88 |
| ATME.startTransmission(file, point, modulation, frequency, power, interval) | 89 |
| ATME.stopTransmission(id)..... | 89 |
| ATME.whichSide(pA, pB, pC)..... | 90 |
| Fonctions de création d'objets statiques de la table ATME | 91 |
| ATME.staticObjects.createCargo(countryName, name, type, position, mass, isDead) | 91 |

| | |
|--|-----------|
| ATME.staticObjects.createWhiteContainer(countryName, name, position, heading, isDead) | 92 |
| Classes ATME | 93 |
| Classe ATME.C_AIUnit | 93 |
| ATME.C_AIUnit.exists(name) | 93 |
| ATME.C_AIUnit.getByName(name)..... | 93 |
| Object:crossAxisFromLeftToRight(pA, pB) | 94 |
| Object:crossAxisFromRightToLeft(pA, pB) | 94 |
| Object:disable() | 95 |
| Object:explode(power) | 95 |
| Object:fireFlare(color) | 95 |
| Object:fireIlluminationBomb(power)..... | 96 |
| Object:fireSmoke(color) | 96 |
| Object:getAGLAltitude() | 96 |
| Object:getAzimuth() | 97 |
| Object:getCallsign() | 97 |
| Object:getClassName() | 97 |
| Object:getCoalitionName() | 98 |
| Object:getCountryName() | 98 |
| Object:getDCSUnit() | 99 |
| Object:getFuelRatio() | 99 |
| Object:getGroup() | 99 |
| Object:getGroupsNameOnBoard()..... | 100 |
| Object:getHSpeed() | 100 |
| Object:getLife()..... | 100 |
| Object:getLifeRatio() | 101 |
| Object:getMSLAltitude()..... | 101 |
| Object:getName() | 101 |
| Object:getNbUnitsOnBoard() | 101 |
| Object:getNearestReadyToBoardGroups(radius) | 102 |
| Object:getPosition() | 102 |
| Object:getRollAxisVector() | 102 |
| Object:getSpeed()..... | 103 |
| Object:getSpeedAzimuth() | 103 |
| Object:getTypeName() | 103 |

| | |
|---|------------|
| Object:getVelocityVector()..... | 103 |
| Object:getVSpeed() | 104 |
| Object:inAir() | 104 |
| Object:isAAA() | 104 |
| Object:isAirDefence() | 105 |
| Object:isGroundVehicle() | 105 |
| Object:isHelicopter() | 105 |
| Object:isInfantry() | 106 |
| Object:isInDCSZone(zoneName) | 106 |
| Object:isInZone2D(reference, radius) | 107 |
| Object:isInZone3D(reference, radius) | 108 |
| Object:isNear(reference, radius, deltaAltitude)..... | 109 |
| Object:isManPad()..... | 110 |
| Object:isPersonnelCarrier() | 110 |
| Object:isPlane() | 110 |
| Object:isSAMVehicle()..... | 111 |
| Object:isSAMSiteCommandCenter()..... | 111 |
| Object:isSAMSiteLauncher()..... | 111 |
| Object:isSAMSiteRadar() | 112 |
| Object:load(groupToBoard, radius)..... | 113 |
| Object:loopTransmission(file, modulation, frequency, power) | 114 |
| Object:sendTransmissionOnce(file, modulation, frequency, power) | 114 |
| Object:startTransmission(file, modulation, frequency, power, interval)..... | 115 |
| Object:stopAllTransmissions()..... | 115 |
| Object:stopTransmission(id) | 116 |
| Object:unload(id)..... | 116 |
| Object:whichSide(reference)..... | 117 |
| Classe ATME.C_AirBase..... | 118 |
| Object:getClassName() | 118 |
| Object:getName() | 118 |
| Classe ATME.C_EventMgr | 119 |
| Object:getCoreEventType(id) | 119 |
| Object:getCoreEventDatas(id)..... | 121 |
| Object:isCoreEvent(id) | 123 |

| | |
|--|------------|
| Object:isUserTriggerEvent(id) | 124 |
| Object:isUserTriggerEventToggle(id)..... | 124 |
| Object:pairs() | 125 |
| Classe ATME.C_Flare..... | 125 |
| ATME.C_Flare(colorName, point) | 125 |
| Object:fire(duration) | 125 |
| Object:fireOnce() | 126 |
| Object:getClassName() | 126 |
| Object:getColor() | 126 |
| Object:getDuration() | 126 |
| Object:getPoint() | 127 |
| Object:getTimeStart() | 127 |
| Object:stop()..... | 127 |
| Classe ATME. C_Group | 128 |
| ATME.C_Group.exists(name) | 128 |
| ATME.C_Group.getByname(name)..... | 128 |
| ATME.C_Group.getGroupsInDCSZoneForAll(zoneName, allGroup) | 129 |
| ATME.C_Group.getGroupsInDCSZoneForCoalition(coalitionName, zoneName, allGroup)..... | 129 |
| ATME.C_Group.getGroupsInZone2DForAll(reference, radius, allGroup) | 130 |
| ATME.C_Group.getGroupsInZone2DForCoalition(coalitionName, reference, allGroup) | 131 |
| ATME.C_Group.getReadyToBoardGroupsForAll()..... | 132 |
| ATME.C_Group.getReadyToBoardGroupsForCoalition(coalitionName, zoneName, allGroup) . | 132 |
| Object:changeReadyToBoard(value)..... | 132 |
| Object:activate() | 133 |
| Object:disable() | 133 |
| Object:freeze(on) | 133 |
| Object:getBarycentre() | 134 |
| Object:getCategoryName() | 134 |
| Object:getClassName() | 134 |
| Object:getCoalitionName() | 134 |
| Object:getDCSGroup() | 135 |
| Object:getFirstUnit()..... | 135 |
| Object:getLastUnit() | 135 |
| Object:getMaxDistance(reference)..... | 135 |

| | |
|---|------------|
| Object:getMaxHDistance(reference) | 136 |
| Object:getMinDistance(reference) | 136 |
| Object:getMinHDistance(reference) | 137 |
| Object:getName() | 137 |
| Object:getNbUnits() | 138 |
| Object:getNextUnit() | 138 |
| Object:getPickupZone() | 138 |
| Object:getPreviousUnit() | 139 |
| Object:getUnitByName(name) | 139 |
| Object:getUnits() | 139 |
| Object:hasPickupZone() | 140 |
| Object:isActivated() | 140 |
| Object:isBoardingStarted() | 141 |
| Object:isInDCSZone(zoneName, allGroup) | 141 |
| Object:isInZone2D(reference, radius, allGroup) | 142 |
| Object:isInZone3D(reference, radius, allGroup) | 143 |
| Object:isNear(reference, radius, deltaAltitude, allGroup) | 144 |
| Object:isOnlyInfantry() | 145 |
| Object:isOnlyAI() | 145 |
| Object:isReadyToBoard() | 145 |
| Object:isSignalSet() | 146 |
| Object:isStopped() | 146 |
| Object:move() | 146 |
| Object:resetPickupZone() | 147 |
| Object:resetSignal() | 147 |
| Object:setPickupZone(zoneName, random) | 148 |
| Object:setRoute(...) | 149 |
| Object:setSignal(radius, signalType) | 150 |
| Object:stop() | 151 |
| Classe ATME. C_GroupSpawnDatas | 151 |
| ATME.C_GroupSpawnDatas.duplicateFromMissionDatas(groupName, newGroupName) | 151 |
| Object:getClassName() | 151 |
| Object:getName() | 152 |
| Object:spawn(...) | 152 |

| | |
|---|------------|
| Classe ATME.C_IndexList..... | 154 |
| ATME.C_IndexList() | 154 |
| Object:add(item) | 154 |
| Object:get(index)..... | 154 |
| Object:getName()..... | 155 |
| Object:getCount()..... | 155 |
| Object:remove(index) | 155 |
| Object:removeAll() | 155 |
| Object:pairs()..... | 156 |
| Classe ATME.C_Line2D..... | 157 |
| ATME.C_Line2D(...)..... | 157 |
| Object:get()..... | 159 |
| Object:getA() | 159 |
| Object:getB() | 159 |
| Object:getC() | 159 |
| Object:getName()..... | 160 |
| Object:getOrigine()..... | 160 |
| Object:getPerpendicular(offset)..... | 161 |
| Object:getPointFromOrigine(offset) | 161 |
| Object:getX() | 162 |
| Object:getZ()..... | 162 |
| Object:isNSAxis() | 164 |
| Object:isWEAxis() | 164 |
| Classe ATME.C_MenuF10..... | 165 |
| ATME.C_MenuF10(player, label, parent)..... | 165 |
| Object:append(groupId, menuLabel, radioHandler, argsRadioHandler) | 165 |
| Object:remove(groupId)..... | 166 |
| Object:removeAll() | 166 |
| Classe ATME.C_Module | 167 |
| ATME.C_Module(name, handlers, debugOn) | 167 |
| Object:error(text) | 170 |
| Object:getName()..... | 170 |
| Object:isDebugOn() | 170 |
| Object:output(text, level)..... | 171 |

| | |
|--|------------|
| Object:createAbsoluteUserTrigger(name, condition)..... | 171 |
| Object:createFlagRelativeUserTrigger(name, flagNumber, condition)..... | 172 |
| Object:getUserTriggerByName(name) | 174 |
| Object:removeUserTrigger(name) | 174 |
| Object:userTriggerExists(name) | 174 |
| Classe ATME. C_Player | 175 |
| ATME.C_Player.exists(name) | 175 |
| ATME.C_Player.getByname(name)..... | 175 |
| Object:crossAxisFromLeftToRight(pA, pB) | 176 |
| Object:crossAxisFromRightToLeft(pA, pB) | 176 |
| Object:display(text, duration) | 177 |
| Object:explode(power) | 177 |
| Object:getAGLAltitude() | 177 |
| Object:getAzimuth() | 178 |
| Object:getCallsign() | 178 |
| Object:getClassName() | 178 |
| Object:getCoalitionName() | 179 |
| Object:getCountryName() | 179 |
| Object:getDCSUnit() | 180 |
| Object:getF10MenuRoot() | 180 |
| Object:getFuelRatio() | 180 |
| Object:getGroup() | 181 |
| Object:getGroupsNameOnBoard() | 181 |
| Object:getHSpeed() | 181 |
| Object:getLife() | 182 |
| Object:getLifeRatio() | 182 |
| Object:getMSLAltitude()..... | 182 |
| Object:getName() | 183 |
| Object:getNbUnitsOnBoard() | 183 |
| Object:getNearestReadyToBoardGroups(radius) | 183 |
| Object:getPosition() | 184 |
| Object:getPseudo()..... | 184 |
| Object:getRollAxisVector() | 184 |
| Object:getSpeed()..... | 185 |

| | |
|---|------------|
| Object:getSpeedAzimuth() | 185 |
| Object:getTypeName() | 185 |
| Object:getVelocityVector() | 186 |
| Object:getVSpeed() | 186 |
| Object:inAir() | 186 |
| Object:isEngineStarted() | 187 |
| Object:isGroundVehicle() | 187 |
| Object:isHelicopter() | 187 |
| Object:isInDCSZone(zoneName) | 188 |
| Object:isRouteInDirection(reference) | 188 |
| Object:isInZone2D(reference, radius) | 189 |
| Object:isInZone3D(reference, radius) | 190 |
| Object:isNear(reference, radius, deltaAltitude) | 191 |
| Object:isPersonnelCarrier() | 192 |
| Object:isPlane() | 192 |
| Object:load(groupToBoard, radius) | 192 |
| Object:soundOnce(file) | 194 |
| Object:soundChange(file, interval, forced) | 194 |
| Object:soundPause(duration) | 194 |
| Object:soundStart(file, interval) | 195 |
| Object:soundStop() | 195 |
| Object:soundChangePlayList(playList, interval, forced) | 195 |
| Object:soundStartPlayList(playList, randomRead, interval) | 196 |
| Object:unload(id) | 196 |
| Object:whichSide(reference) | 197 |
| Classe ATME.C_Race | 198 |
| ATME.C_Race(name, leftSideName, rightSideName, nbTurns) | 198 |
| ATME.C_Race.exists(name) | 199 |
| ATME.C_Race.getBy_name(name) | 199 |
| Object:addPlayer(player) | 199 |
| Object:delete() | 200 |
| Object:displayForAllPlayers(text, duration) | 200 |
| Object:isPlayerInRace(player) | 200 |
| Object:getClassName() | 201 |

| | |
|---|------------|
| Object:getDoorIndex(offset) | 201 |
| Object:getDoorSides(index) | 201 |
| Object:getLapAtDoor(index) | 202 |
| Object:getMaxAltitude()..... | 202 |
| Object:getMaxAltitudePenalty() | 202 |
| Object:getMissedDoorPenalty() | 203 |
| Object:getName() | 203 |
| Object:getNbDoors() | 203 |
| Object:getNbTurns() | 203 |
| Object:getPlayerNextDoorIndex(player) | 204 |
| Object:getRanking()..... | 204 |
| Object:removeAllPlayers()..... | 205 |
| Object:removePlayer(player) | 205 |
| Object:resetMaxAltitudeRule() | 205 |
| Object:resetMissedDoorRule()..... | 205 |
| Object:setLapAtDoor(index)..... | 206 |
| Object:setMaxAltitudeRule(maxAltitude, penalty) | 206 |
| Object:setMissedDoorRule(penalty) | 206 |
| Object:soundForAllPlayers(file)..... | 207 |
| Classe ATME.C_Smoke | 208 |
| ATME.C_Smoke(colorName, point)..... | 208 |
| Object:activate(restart) | 208 |
| Object:getClassName() | 209 |
| Object:getColor() | 209 |
| Object:getPoint() | 209 |
| Object:stop()..... | 209 |
| Classe ATME.C_StaticObject..... | 210 |
| ATME.C_StaticObject.exists(name)..... | 210 |
| ATME.C_StaticObject.getByname(name) | 210 |
| Object:explode(power) | 210 |
| Object:fireFlare(color) | 211 |
| Object:fireIlluminationBomb(power) | 211 |
| Object:fireSmoke(color) | 211 |
| Object:getAzimuth() | 212 |

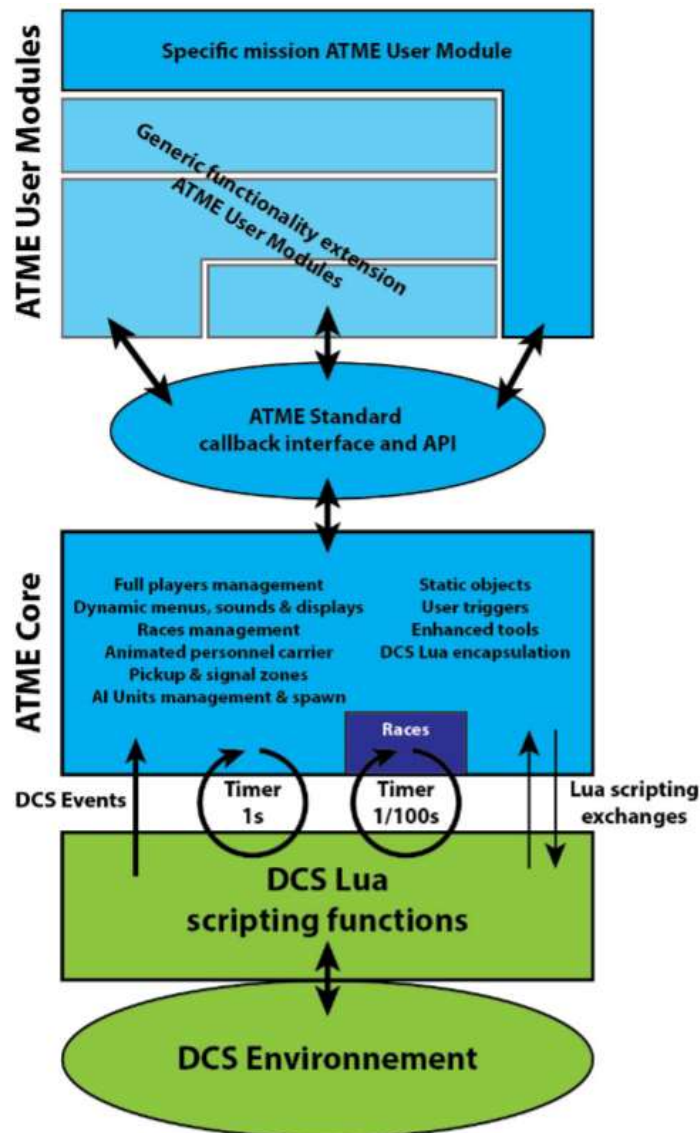
| | |
|----------------------------------|------------|
| Object:getName() | 212 |
| Object:getCoalitionName() | 212 |
| Object:getCountryName() | 213 |
| Object:getDCSStaticObject() | 213 |
| Object:getLife() | 214 |
| Object:getName() | 214 |
| Object:getPosition() | 214 |
| Object:getTypeName() | 215 |
| Classe ATME.C_UserTrigger | 215 |
| Object:getName() | 215 |
| Object:getFlag() | 216 |
| Object:getName() | 216 |
| Object:getTimeEnd() | 216 |
| Object:getTimeStart() | 217 |
| Object:isActivated() | 217 |
| Object:isArmed() | 217 |
| Classe ATME.C_Vector3D | 218 |
| ATME.C_Vector3D(...) | 218 |
| Object:get() | 219 |
| Object:getAzimuth() | 219 |
| Object:getName() | 219 |
| Object:getHModule() | 220 |
| Object:getModule() | 220 |
| Object:getX() | 220 |
| Object:getY() | 221 |
| Object:getZ() | 221 |
| Object:toUnitVector() | 221 |
| Object:scalaireH(vector) | 222 |

Introduction

Advanced Tools for Mission Editor ou ATME est un ensemble de classes et de fonctions lua simplifiant la mise en œuvre de script Lua dans l'éditeur de mission de DCS World.

L'objectif d'ATME est de faciliter la création de missions compatibles mono et multi joueurs, sans limitation du nombre de joueurs. La dynamique induite par l'entrée en jeu des joueurs, leur départ/destruction, ou le changement de slot, est intégralement prise en charge.

ATME est basé sur un concept de modules, le module de base s'appelant **ATME Core** et devant être chargé impérativement en premier. Chaque module supplémentaire, ou **ATME User Module**, créé peut être générique ou dédié à une mission. Les futurs modules génériques apporteront des fonctionnalités supplémentaires. Ce concept offre une grande souplesse en permettant à des utilisateurs qui ne sont pas à l'aise avec Lua de créer des missions complexes avec peu de code.



Une attention particulière a été portée pour éviter les corruptions involontaires de données, notamment par l'impossibilité de modifier simplement certaines données d'objet. Il conviendra d'utiliser les fonctions mises à disposition pour ce faire. Les données internes aux objets ne sont pas directement accessibles aux créateurs de modules utilisateurs.

Le module **ATME Core** apporte toute l'abstraction nécessaire et assure l'interface avec une grande partie des fonctions de DCS World. L'organisation de ces fonctions est avant tout orientée vers les concepteurs de mission. Une approche objet apporte la clarté nécessaire.

ATME Core regroupe les grandes fonctionnalités suivantes :

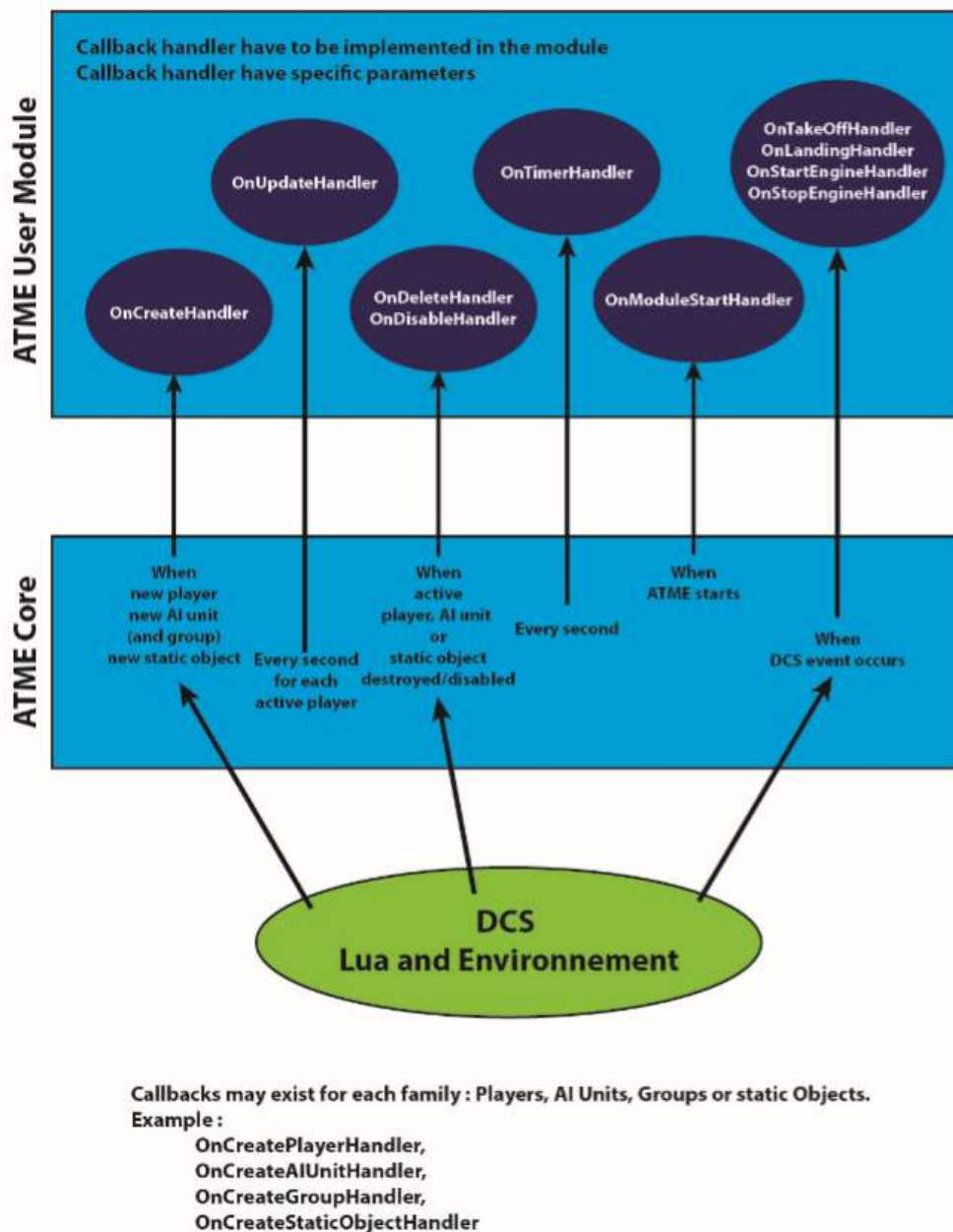
- Mise à disposition de fonctions globales reprise de DCS (encapsulation), génériques (conversions, etc...) ou mathématiques (vecteurs 3D, droites 2D, positionnement relatif, etc...)
- Référencement des FARP & Airfields d'une mission en fonction du théâtre d'opérations (NEVADA ou CAUCASUS pour l'instant).
- Gestion des objets joueurs, unités AI, groupes et objets statiques. Possibilité de spawn, d'association de routes ou de reprises de route à partir d'un waypoint donné etc...
- Ajout d'objets particuliers pour en faciliter la gestion (menu radio joueurs, fumées, flares, etc...)
- Gestion complète de courses en multijoueur avec temps au 1/100è, temps intermédiaires, temps au tour, temps total et classements. Possibilité de créer des parcours avec plusieurs courses.
- Gestion complète de l'embarquement/débarquement d'unités d'infanteries avec dynamique de montée dans le véhicule. Ajout d'une notion de transport de troupes (personnel carrier) autorisant l'embarquement à bord d'unité AI (sol ou hélicoptères) ou d'unité joueurs (hélicoptères). Possibilité de créer des zones d'embarquement ou des signalements de position par fumée ou flare.
- Déclenchement d'événements Core pouvant être traités dans les modules (fin d'embarquement, événements de course, entrée dans une zone de signalement, etc...).
- Création et gestion de triggers utilisateur (**User Trigger**) permettant le déclenchement d'un événement unique ou répété à instant T définissable en temps absolu ou en relatif à un flag. Cet événement User Trigger ne sera émis qu'au module ayant créé le trigger.
- Support multi-langue : Français, Anglais, Allemand et russe actuellement pour les messages et label. La mise en place de cette fonctionnalité est à la charge du créateur du module.

ATME est compatible pour DCS 1.5.5 (et suivantes) et DCS 2.0.4 (et suivantes)

Le concept de callback

Ce concept permet au module de base **ATME Core** de communiquer avec les modules utilisateur (**ATME User Modules**) en apportant la dynamique nécessaire. Une callback est une fonction définie et implémentée dans les modules utilisateurs. Elle sera appelée par le module ATME Core en fonction du contexte de la mission. Ces fonctions ont des paramètres définis et imposés. Elle concerne la création d'objet, leur destruction ou désactivation, le traitement de certains événements DCS comme le décollage ou l'atterrissage, ou l'activation régulière à la seconde par l'activation d'un timer interne à ATME Core des callbacks onUpdate ou onTimer. Ces callbacks doivent être connues

du module ATME Core, aussi, tout nouveau module devra déclarer ses callbacks par l'intermédiaire d'une liste de handlers dédié. Un handler est en fait un identifiant de la fonction permettant ensuite au module ATME Core d'appeler la fonction lorsque nécessaire.



Le concept d'événements ATME

Les événements ATME sont émis au travers de deux callbacks liées au timer interne à ATME. Ce timer se déclenche toutes les secondes.

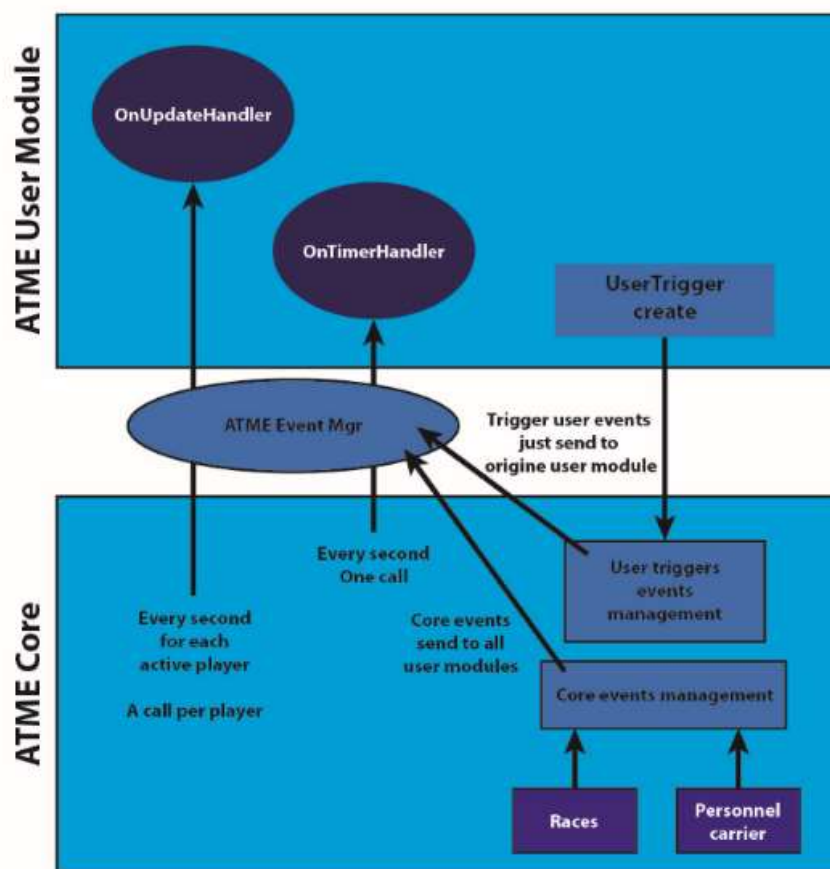
Il existe deux types d'événements ATME :

- Les événements ATME Core (**Core Event**) : Ces événements sont la conséquence de traitements dans le module ATME Core. Il s'agit notamment des événements liés au

transport de troupes, à la gestion des zones de signalement ou à la gestion des courses. **Ces événements Core sont envoyés à tous les modules utilisateurs de la mission en cours.**

- Les événements induits par les triggers utilisateur (**User Trigger Event**) : Un trigger utilisateur permet de déclencher un événement à un moment donné de la mission pour une durée précise. Cette durée pourra être nulle, dans ce cas l'événement ne sera émis qu'une seule fois. Un **User Trigger** doit être créé préalablement dans un module utilisateur. **Les événements générés par les User Triggers ne seront envoyés qu'aux modules utilisateurs les ayant créés.**

Ces événements sont intégrés dans une instance d'objet ATME nommé ATME Event Mgr (Manager). Cette instance sera transmise deux callbacks OnUpdate et OnTimer. Cette instance ne contiendra que les événements actifs au moment de l'appel.



ATME Event manager (Mgr) is OnUpdateHandlers and OnTimerHandlers parameter.

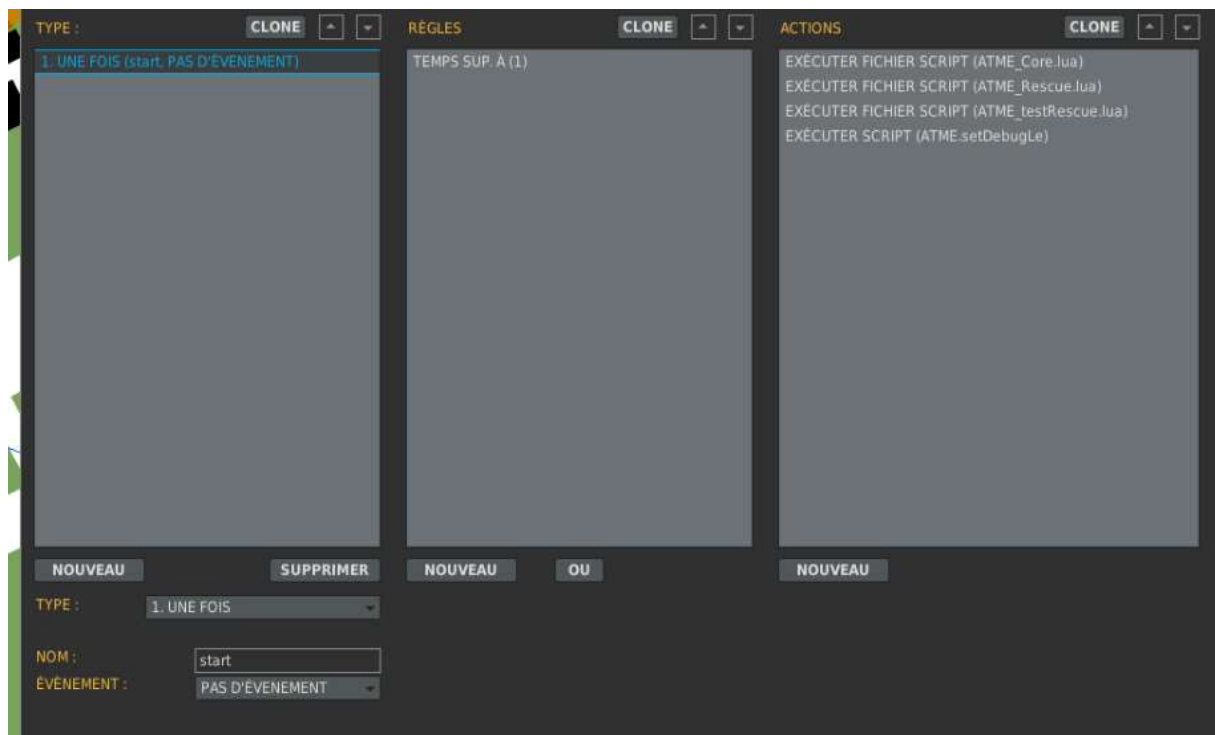
Chaque événement est caractérisé par sa source (Core ou Trigger User), son type pour les événements Core, et ses données associées variables en fonction de l'événement. Ces données sont rassemblées dans une table et mises à disposition du créateur du module par des fonctions appropriées.

L'éditeur de mission DCS et ATME

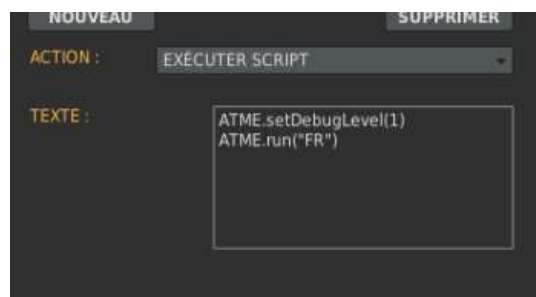
Les modules ATME s'intègrent dans une mission comme tous les fichiers lua.

Dans l'éditeur de mission, il convient donc :

- De créer un trigger qui se déclenche au bout d'une seconde et va charger l'environnement nécessaire pour faire fonctionner ATME. On trouvera les fichiers scripts des modules ATME (Core en premier suivi des modules utilisateurs eux-même dans l'ordre approprié).



- Un script créé dans l'éditeur lançant ATME par sa fonction **ATME.run** qui spécifie également la langue souhaitée (si cette fonctionnalité est supportée par les modules utilisateurs déclarés). La fonction **ATME.setDebugLevel** n'est utile que pour les tests. La fonction run spécifie également la langue utilisée.



Il est également possible de créer des fonctions ou variables globales dans un module utilisateur (ATME User module) qui pourront directement être appelées au travers de l'éditeur de mission. ATME Core met à disposition du créateur du module une table globale par module, référencée sous le nom **ATME.modules["nom du module"]**

Définitions et notations

Une classe ATME est définie par :

- **Classe** : Structure définissant les propriétés des objets gérés par ATME et/ou issus de DCS. On retrouvera une classe particulière pour les joueurs, les unités AI, les groupes mais aussi des classes plus abstraites comme les modules ou les vecteurs 3D.
- **Instance** : Une instance est la représentation concrète d'un objet particulier dans la mission ou dans ATME. Une instance est forcément rattachée à une et une seule classe dans ATME. Il existe donc au cours d'une mission une multitude d'instances toute classe confondues. Une instance peut être créée ou détruite. Certaines instances associées à des classes particulières sont complètement gérées par ATME (joueurs, unités AI, groupes, objet statiques) ; il n'est pas possible d'en créer pour le créateur d'un module.

Il convient de retenir la terminologie suivante qui sera utilisée tout au long de ce manuel :

- Des fonctions ATME standard nommées **ATME.Nom_de_fonction** ou **ATME.staticObject.Nom_de_fonction_spawn**
- La fonction de construction d'instance pour une classe donnée, si cela est possible, **ATME.Nom_de_la_classe**. Cette fonction particulière retourne une table lua correspondant à une nouvelle instance de la classe. **Cette fonction est appelée constructeur.**
- Des fonctions de classe **ATME.Nom_de_la_classe.fonction_de_classe** : ces fonctions ne font pas référence à une instance de classe particulière. Pour les objets de base référençant les joueurs, les unités AI, les groupes, les objets statiques ou les course, on retiendra **getByName** (ou **exists**) qui permettent de récupérer l'instance de l'objet (ou d'en vérifier l'existence). D'autres fonctions existent cependant.
- Des méthodes (ou fonctions) d'instance **Object:fonction_d_instance** liées à une instance d'objet particulier. **Attention à cette syntaxe, elle est importante.**
- Des attributs (variables) associés accessibles uniquement par des fonctions particulières. Certainement propriétés sont inaccessible au créateur d'une mission car utilisées uniquement pour le fonctionnement interne d'ATME.

Ces trois notations seront utilisées dans ce document :

- **ATME.Nom_de_la_classe**
- **ATME.Nom_de_la_classe.fonction_de_classe**
- **Object:fonction_d_instance**

Restrictions & limitations

ATME n'a pas été testé ni validé avec le module DCS Combined Arms. Aussi la prise en charge du contrôle de véhicule au sol par un joueur n'est pas garantie. Ces tests vont être menés ultérieurement.

La création de modules utilisateur ATME

La création de module ATME va être abordée au travers d'exemple de plus en plus complexes. Pour plus d'informations, on se reportera sur la définition des fonctions globales et des classes ATME exposées plus loin dans ce manuel. En effet, dans ces exemples, la description en profondeur des fonctions utilisées ne sera pas exposée.

ATME Core encapsule toute la complexité de gestion des joueurs, des unités AI, des courses et autres possibilités d'embarquement/débarquement, menus dynamiques de joueurs, zone de signalement, etc...

Aussi la mise en place d'un nouveau module reste simple :

- Création d'un fichier lua
- Mise en place de l'ossature du module par l'implémentation des callbacks et la création de l'instance de la classe module permettant la prise en charge du module par ATME Core.
- Mise en place des variables et fonctions internes au module
- Déclaration et mise en place de fonctions et variables globales utilisables directement dans l'éditeur de mission DCS ou dans un autre module.

Module 1 : module utilisateur : « Hello world »

Ce module très simple permet l'affichage de « Hello World... » suivi du pseudo multijoueur (« nouveau surnom » en monojoueur) dès qu'un joueur prend le contrôle d'un avion ou d'un hélicoptère. Pour des raisons de clarté, les handlers de callback non utilisées sont renseignées à **nil**. Ce n'est pas obligatoire, toute variable lua non initialisée étant à **nil**. Mais ceci permet de visualiser la liste exhaustive des handlers de callback existants dans ATME.

Ce module n'implémente qu'une seule callback lié à l'entrée d'un joueur dans la mission qui utilisera ce module ATME.

Code lua

```
-- Advanced Tools for Mission Editor
local thisModule

local function onCreatePlayer(player)
    player:display("Hello World ... " .. player:getPseudo(), 10)
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
```

```

onDeleteAIUnitHandler = nil,
onDisableAIUnitHandler = nil,
onTakeoffAIUnitHandler = nil,
onLandingAIUnitHandler = nil,
onStartEngineAIUnitHandler = nil,
onStopEngineAIUnitHandler = nil,

onCreateGroupHandler = nil,
onDeleteGroupHandler = nil,
onDisableGroupHandler = nil,

onCreateStaticObjectHandler = nil,
onDeleteStaticObjectHandler = nil,

onTimerHandler = nil,
onModuleStartHandler = nil,
}

thisModule = ATME.C_Module("HelloWorld", newHandlers, true)
end

```

Explications

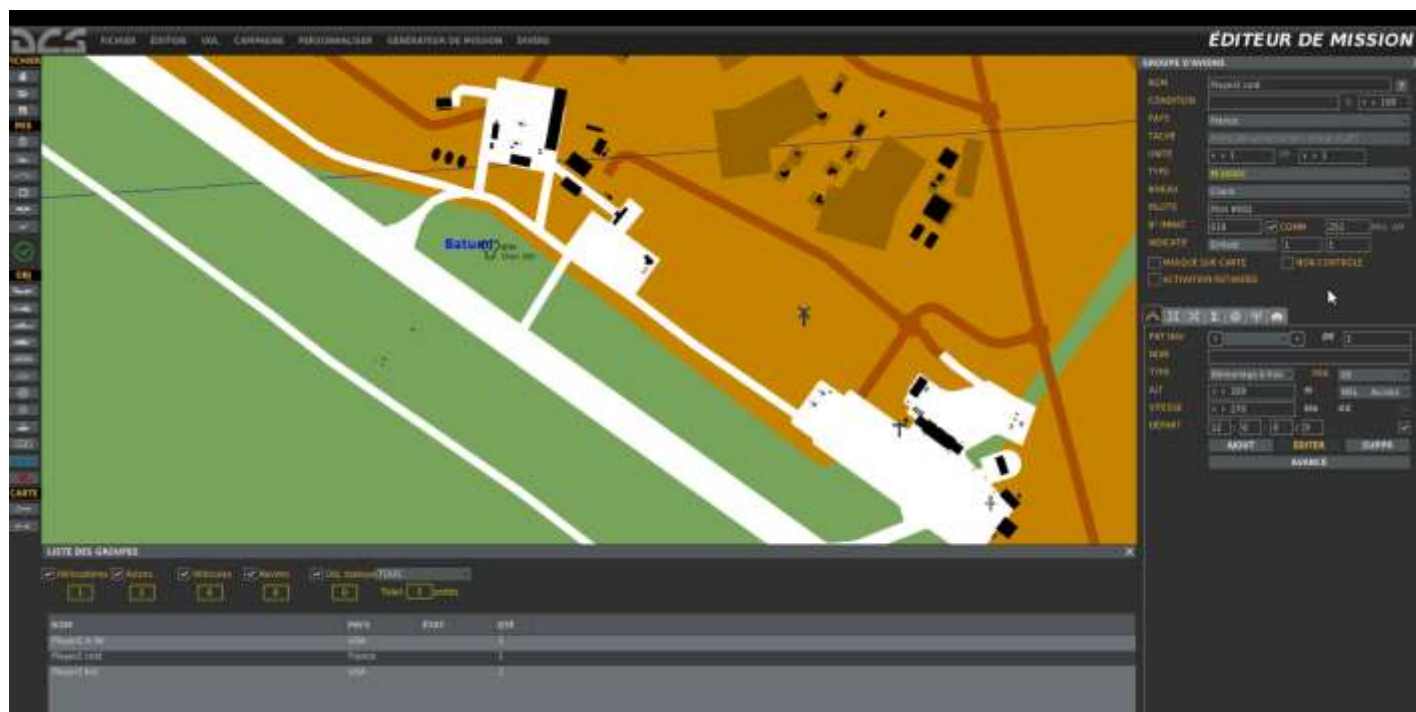
Le module est composé de trois parties :

- La définition d'une variable globale interne au module : **thisModule**. Cette variable est l'instance de la classe **ATME.C_Module** pour ce module. Elle sera initialisée dans la 3^{ème} partie. Elle servira lors de l'utilisation de fonctions de la classe **ATME.C_Module**.
- La définition de la callback **onCreatePlayer** : Le paramètre de la callback est imposé dans son type par ATME Core ; **player** est une instance de la classe **ATME.C_Player** représentant les unités joueur. Dès qu'un joueur arrive dans la mission, cette callback sera appelée. Les fonctions **display** et **getPseudo** sont des fonctions de la classe **ATME.C_Player**. On se reportera à la définition des classes ATME plus loin dans ce document. La notation **player:display** est une notation objet lua. Ceci indique que nous souhaitons utiliser la fonction **display** pour l'instance **player**. Cette notation sera toujours à utiliser. Pour plus de renseignement, il faudra se reporter aux documents sur le langage lua existant sur Internet. **L'utilisation de `player.display` à la place de `player:display` va engendrer une erreur dans le passage des paramètres et donc un dysfonctionnement.**
- L'initialisation du module, après le commentaire **--MAIN** et encadré de **do .. end** pour des raisons de clarté : Cette partie est exécutée dès que le fichier lua est déclaré dans l'éditeur de mission (EXECUTER FICHER SCRIPT). Elle définit la table des handlers de callback (ici uniquement le handler de la callback **onCreatePlayer** définie en 2^{ème} partie. Ce handler sera copier dans **onCreatePlayerHandler** de la table **newHandlers**. Les autres handlers de callback n'étant pas utilisés dans ce module, il restent à la valeur **nil**. Enfin, l'instance du module est initialisée par le constructeur **ATME.C_Module**. Le premier paramètre est le nom du module qui devra être unique pour une mission donnée, le deuxième paramètre est la table des handlers qui sera utilisée par ATME Cord. Enfin, le dernier paramètre, ici à **true** indique si un module est en mode debug. Il devra être à **false** une fois le module finalisé et testé.

Dès lors le module est terminé. Il est possible de l'enrichir ou de changer le message affiché en utilisant d'autres fonctions existantes dans ATME.

Tests et essais

Pour les tests et essais, il convient de créer une mission et de définir des unités en client. Le nombre d'unité importe peu, car tout sera pris en charge par ATME.

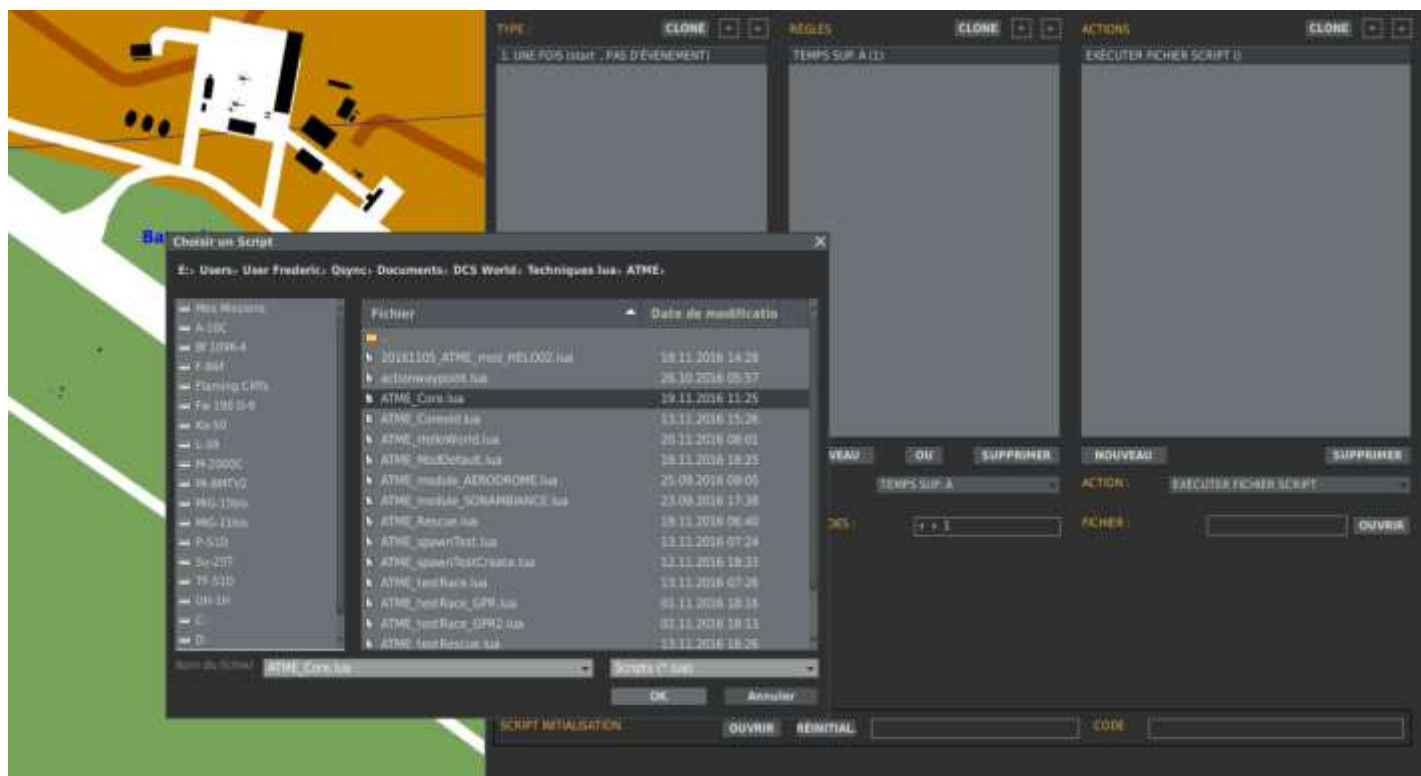


Dans l'exemple ci-dessus, 3 unités joueur ont été créées, deux avions au sol et un hélicoptère en vol. Il est bien sûr possible de créer des unités joueur pour les coalitions bleue ou rouge. Pour ce module, aucune distinction n'est faite. Tous les joueurs verront donc le message créé.

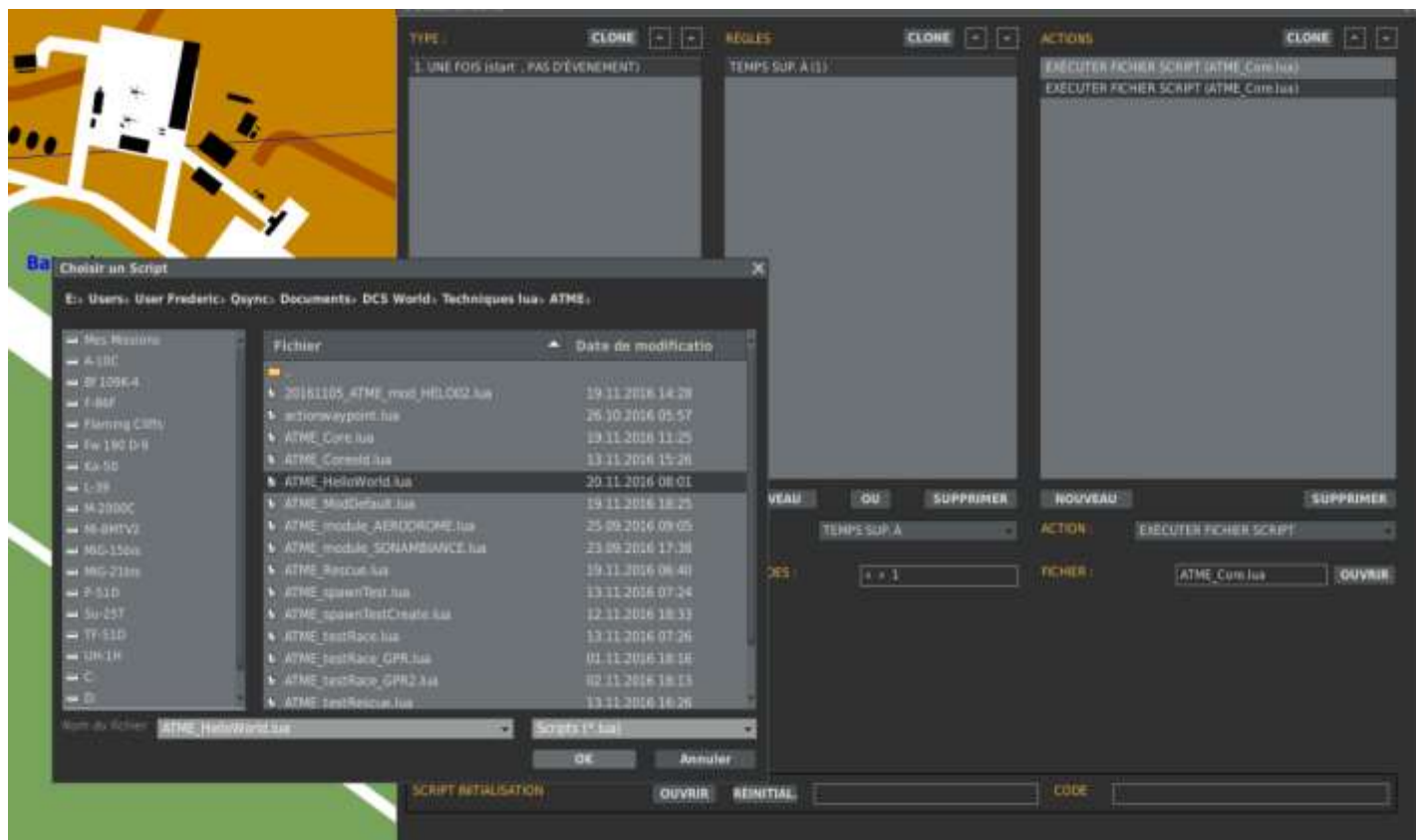
Il faut ajouter le trigger pour démarrer ATME, ici nommé start :



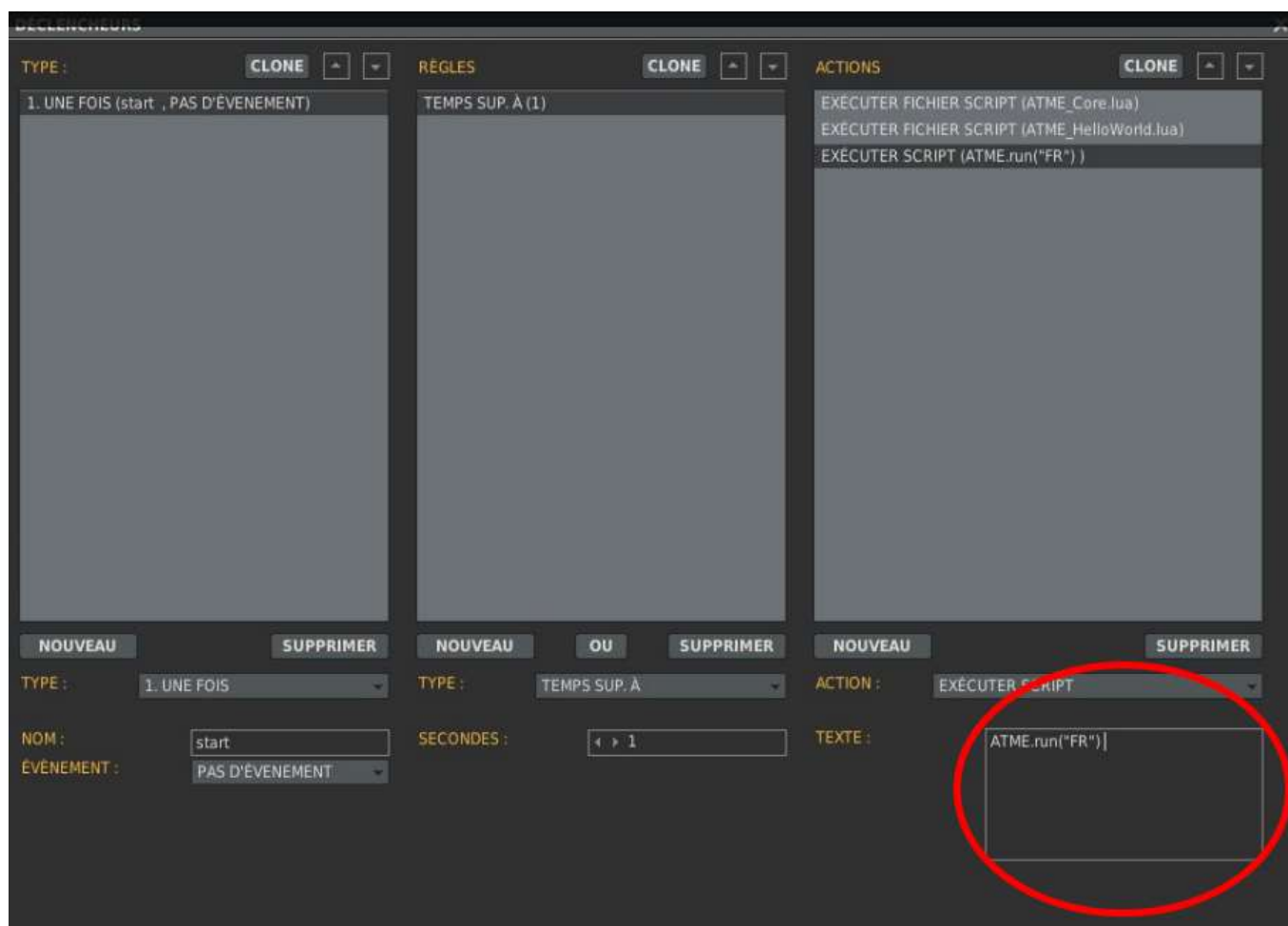
Puis lui mettre une règle pour un déclenchement après 1s et enfin définir les actions EXECUTER FICHIER SCRIPT pour ATME Core :



Il faut ensuite cloner pour le module HelloWorld :



Et enfin, créer le script contenant la fonction **ATME.run** (et si souhaité la fonction **ATME.setDebugLevel**, non fait ci après) :



Dès lors, ATME est prêt à être utilisé dans cette mission avec le module créé, ci-dessous un test en monojoueur :



Naturellement, le fonctionnement sera identique en multijoueur avec en affichage le pseudo du joueur. Il est aussi possible de faire la même chose en version 2.0.4 et suivantes, sur le théâtre d'opération NEVADA.

Exercice

Exercice : Afficher « Welcome in ... » et le nom du théâtre à la place de « Hello World ». Le fonction utile est **ATME.getTheatre()**

Réponse :

```
local function onCreatePlayer(player)
    player:display("Welcome in " .. ATME.getTheatre() .. " ... " .. player.getPseudo(), 10)
end
```

Pour les autres modules, le processus de création dans l'éditeur de mission restera le même. Si un module utilise d'autres modules, tous les modules nécessaires devront être intégrés dans le bon ordre.

Dans la suite du document, cette intégration et le test et ne sera plus exposé sauf en cas de nécessité. **Les modifications seront indiquées en surligné vert.**

Module 1 : Ajout d'un affichage dynamique

Ce module reprend la base du module précédent mais en permettant une mise à jour régulière des informations affichées.

Code lua

```
-- Advanced Tools for Mission Editor
local thisModule

local function onCreatePlayer(player)
    player:display("Welcome in " .. ATME.getTheatre() .. " ... " .. player.getPseudo(), 10)
end

local function onUpdatePlayer(player, events)
    local text = "Azimuth : " .. player.getAzimuth()

    if player.inAir() == true then
        player:display("In Air : " .. text, 5)
    else
        player:display("On floor : " .. text, 5)
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = onUpdatePlayer,
    }
end
```

```

onTakeoffPlayerHandler = nil,
onLandingPlayerHandler = nil,
onStartEnginePlayerHandler = nil,
onStopEnginePlayerHandler = nil,

onCreateAIUnitHandler = nil,
onDeleteAIUnitHandler = nil,
onDisableAIUnitHandler = nil,
onTakeoffAIUnitHandler = nil,
onLandingAIUnitHandler = nil,
onStartEngineAIUnitHandler = nil,
onStopEngineAIUnitHandler = nil,

onCreateGroupHandler = nil,
onDeleteGroupHandler = nil,
onDisableGroupHandler = nil,

onCreateStaticObjectHandler = nil,
onDeleteStaticObjectHandler = nil,

onTimerHandler = nil,
onModuleStartHandler = nil,
}

thisModule = ATME.C_Module("Module1", newHandlers, true)
end

```

Explications

- Le module déclare à nouveau handler sur OnUpdatePlayer. Cette callback sera appelée une fois toutes les secondes pour chaque joueur actif ; le joueur est passé en paramètre, instance **ATME.C_Player**. Les éventuels événements activés seront également transmis au travers du paramètre events, instance de la classe **ATME.C_EventsMgr**. Dans cet exemple, ces événements ne sont pas traités. La fonction callback OnUpdatePlayer crée un affichage différent si le joueur est en l'air ou au sol. L'affichage correspond à l'azimuth du joueur à un instant T (sans prise en compte de la variation magnétique).

Tests et essais

Pour cet exemple rien de particulier, il suffit de suivre la procédure décrite précédemment.



L'azimuth peut fluctuer à l'arrêt.

Le rafraichissement se répète toute les secondes.

Exercice

Exercice : Afficher l'altitude MSL du joueur en plus de son azimuth

Réponse :

```
local function onUpdatePlayer(player)
    local text = "Azimuth : " .. player:getAzimuth()
    text = text .. " - Alt MSL : " .. player:getMSLAltitude()

    if player:inAir() == true then
        player:display("In Air : " .. text, 5)
    else
        player:display("On floor : " .. text, 5)
    end
end
```

Module 1 : remplacement de l'affichage dynamique par un menu spécifique

Ce module reprend la base du module précédent mais cette fois, les informations ne seront affichées que sur demande du joueur au travers du menu F10.

Code lua

```
-- Advanced Tools for Mission Editor

local thisModule

local F10DisplayCommand
F10DisplayCommand = function(args)
    local player = args[1]

    local text = "Azimuth : " .. player:getAzimuth()
    text = text .. " - Alt MSL : " .. player:getMSLAltitude()

    if player:inAir() == true then
        player:display("In Air : " .. text, 5)
    else
        player:display("On floor : " .. text, 5)
    end
end

local function onCreatePlayer(player)
    player:display("Welcome in " .. ATME.getTheatre() .. " ... " .. player:getPseudo(), 10)

    local menu = player:getF10MenuRoot()
    menu:append(1, "Display informations", F10DisplayCommand, {player})
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,
    }
```

```

onCreateAIUnitHandler = nil,
onDeleteAIUnitHandler = nil,
onDisableAIUnitHandler = nil,
onTakeoffAIUnitHandler = nil,
onLandingAIUnitHandler = nil,
onStartEngineAIUnitHandler = nil,
onStopEngineAIUnitHandler = nil,

onCreateGroupHandler = nil,
onDeleteGroupHandler = nil,
onDisableGroupHandler = nil,

onCreateStaticObjectHandler = nil,
onDeleteStaticObjectHandler = nil,

onTimerHandler = nil,
onModuleStartHandler = nil,
}

thisModule = ATME.C_Module("Module1", newHandlers, true)
end

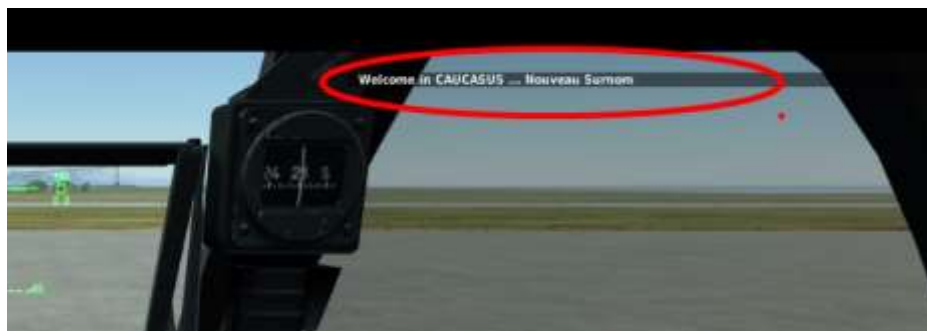
```

Explications

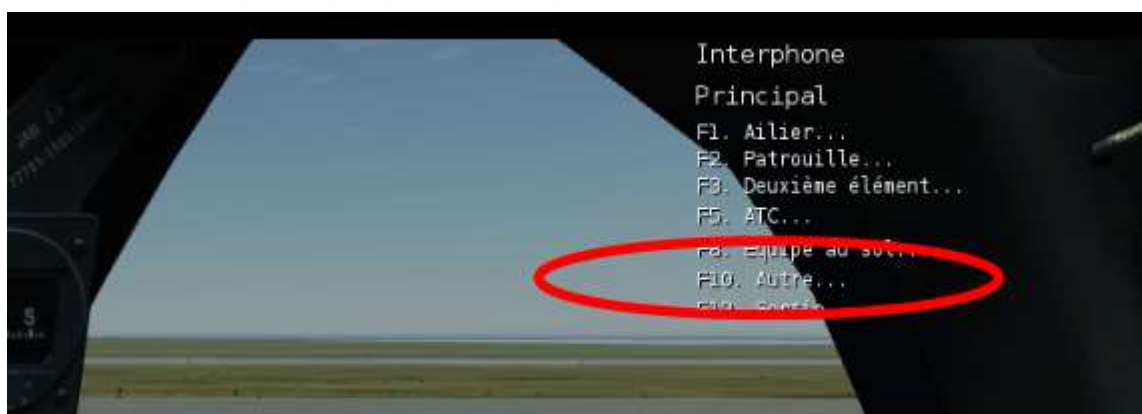
- Un menu est toujours associé à sa propre fonction qui ne sera exécutée que lorsque le menu sera utilisé par le joueur (menu radio F10). Cette fonction, ici **F10DisplayCommand**, a une déclaration un peu spéciale du fait qu'elle est locale (contrainte lua) : le nom est déclaré (**local F10DisplayCommand**) puis à la ligne suivante la fonction elle-même. Cette fonction accepte un paramètre particulier **args** qui est un tableau. Ce tableau est initialisé dans le **onCreatePlayer**. La fonction **F10displayCommand** reprend l'affichage de la callback **onUpdatePlayer** de l'exemple précédent.
- La fonction callback **OnCreatePlayer** se voit compléter d'une variable **menu** initialisée avec une instance de la classe **ATME.C_F10Menu** correspondant au menu de base F10 du joueur grâce à la fonction **player:getF10MenuRoot**. A la ligne suivante, une entrée est ajoutée à ce menu par la fonction **menu:append** de la classe **ATME.C_F10Menu** qui admet :
 - un premier paramètre qui sera expliqué ultérieurement,
 - le deuxième paramètre est le libellé affiché dans le menu F10,
 - le troisième correspond à la fonction **F10DisplayCommand** qui sera donc appelée lors du choix de l'entrée du menu F10 par le joueur
 - le dernier paramètre correspond aux paramètres passés à la fonction **F10DisplayCommand**. Pour pouvoir réaliser les affichages souhaités, la fonction **F10DisplayCommand** doit connaître l'instance du joueur **player**.

Tests et essais

Cette fois, seul le Welcome s'affiche dans un premier temps.



Il faut ensuite activer le menu radio, éventuellement appuyer sur F11 avant d'avoir :



Puis un appui sur F10 donne :



Et enfin l'affichage des informations souhaitées :



Exercice

Exercice : Créer une deuxième entrée de menu F10 qui affiche le nom de l'unité du joueur définie dans l'éditeur de mission. La fonction associée s'appellera **F10DisplayNameCommand**

Réponse : Création d'une nouvelle fonction **F10DisplayNameCommand** et modification de la fonction callback onCreatePlayer.

```
local F10DisplayNameCommand
F10displayCommand = function(args)
    local player = args[1]

    player:display("Unit player name : " .. player:getName(), 5)
end

local function onCreatePlayer(player)
    player:display("Welcome in " .. ATME.getTheatre() .. " ... " .. player.getPseudo(), 10)

    local menu = player:getF10MenuRoot()
    menu:append(1, "Display informations", F10DisplayCommand, {player})
    menu:append(1, "Display unit player name", F10DisplayNameCommand, {player})
end
```

L'ajout entrées dans des menus (et sous-menus vus plus loin) restera identique à ce qui a été décrit pour tous les modules qui seront créés par des utilisateurs.

Module 1 : Ajout d'un menu visible selon certains cas

Ce module reprend la base du module précédent mais cette fois, une nouvelle entrée indiquant la position en longitude/latitude est créé uniquement quand le joueur est au sol.

Ce module va utiliser le premier argument de la fonction **menu:append** qui permet de supprimer un ou plusieurs items de menu. La suppression se base en effet sur cette valeur.

Code lua

```
-- Advanced Tools for Mission Editor

local thisModule

local F10DisplayCommand
F10DisplayCommand = function(args)
    local player = args[1]

    local text = "Azimuth : " .. player:getAzimuth()
    text = text .. " - Alt MSL : " .. player:getMSLAltitude()

    if player:inAir() == true then
        player:display("In Air : " .. text, 5)
    else
        player:display("On floor : " .. text, 5)
    end
end

local F10DisplayNameCommand
F10DisplayNameCommand = function(args)
    local player = args[1]

    player:display("Unit player name : " .. player:getName(), 5)
end
```

```

local F10DisplayPositionCommand
F10DisplayPositionCommand = function(args)
    local player = args[1]

    local position = player:getPosition()
    local lat, lon, alt = ATME.convertPointToLL(position)

    local dirlat, dlat, mlat, slat = ATME.convertToDMS(lat, false)
    local dirlon, dlon, mlon, slon = ATME.convertToDMS(lon, false)

    local westEast = "E"
    if dirlat == -1 then
        westEast = "W"
    end

    local northSouth = "N"
    if dirlon == -1 then
        northSouth = "S"
    end

    local text = string.format("Lat : %s%d°%d'%d\"\\n", westEast, dlat, mlat, slat)
    text = text .. string.format("Lon : %s%d°%d'%d\"\\n", northSouth, dlon, mlon, slon)

    player:display(text, 5)
end

local function onCreatePlayer(player)
    player:display("Welcome in " .. ATME.getTheatre() .. " ... " .. player:getPseudo(), 10)

    local menu = player:getF10MenuRoot()
    menu:append(1, "Display informations", F10DisplayCommand , {player})
    menu:append(1, "Display unit player name", F10DisplayNameCommand , {player})

    if player:inAir() == false then
        menu:append(2, "Display position", F10DisplayPositionCommand , {player})
    end
end

local function onTakeOffPlayer(player)
    local menu = player:getF10MenuRoot()

    menu:remove(2)
end

local function onLandingPlayer(player)
    local menu = player:getF10MenuRoot()

    menu:append(2, "Display position", F10DisplayPositionCommand , {player})
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = onTakeOffPlayer,
        onLandingPlayerHandler = onLandingPlayer,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,
    }

```



```

    onTimerHandler = nil,
    onModuleStartHandler = nil,
}

thisModule = ATME.C_Module("Module1", newHandlers, true)
end

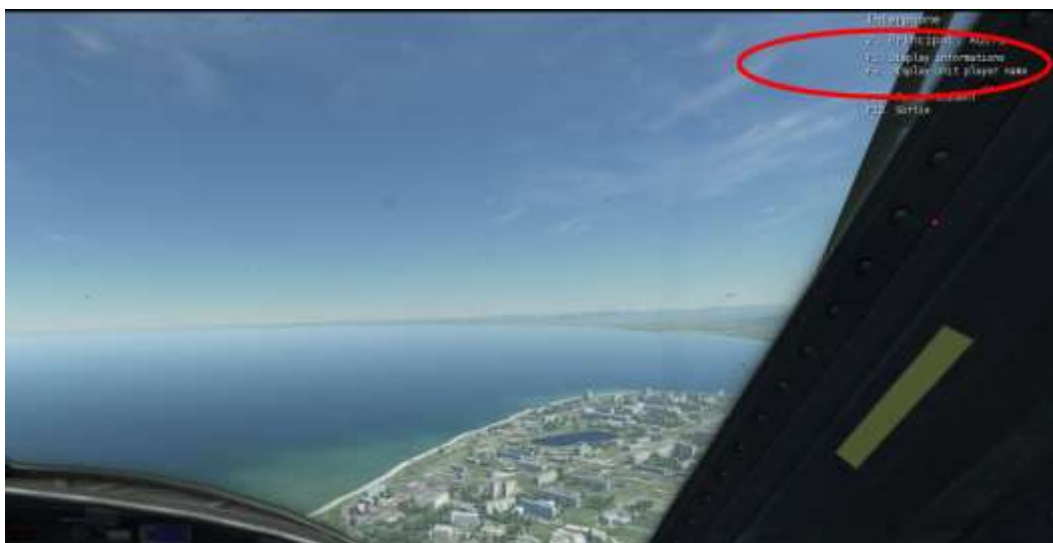
```

Explications

- La nouvelle fonction de menu **F10DisplayPositionCommand** est ajoutée. Cette fonction s'attache à convertir une position de joueur en coordonnées affichables :
 - **player:getPosition** : pour récupérer la position du joueur (un point)
 - **ATME.convertPointToLL** convertit le point en latitude/longitude/altitude
 - **ATME.convertToDMS** transforme des degrés décimaux en degrés minutes secondes complété de la direction.
 - **string.format** est une fonction lua permettant de mettre en forme l'affichage texte.
- Pour gérer le décollage et l'atterrissage, on se basera sur les deux callback **onTakeOffPlayer** et **onLandingPlayer** :
 - **onTakeOffPlayer** supprime l'entrée menu lorsque le joueur décolle grâce à la fonction **menu:remove** qui utilise le **groupId** passé en paramètre (valeur 2).
 - **onLandingPlayer** crée une nouvelle entrée de menu pour l'affichage de la position au sol. Le premier paramètre de **menu:append** est 2 et correspond au **groupId**. Il est différent des deux autres car ce menu sera supprimé en fonction du contexte (joueur au sol/joueur en l'air)
- **onCreatePlayer** est modifié pour ajouter un traitement si le joueur débute au sol. En effet, dans ce cas, le menu de position devra exister dès le début.

Tests et essais

Cette fois, les menus F10 changent. En l'air, la demande de position ne s'affiche pas :



Au sol, la demande de position est affichée :



Et le résultat de cette demande est :



Exercice

Exercice : Supprimer les deux autres menus dès le premier décollage. Ils ne seront plus accessibles.

Réponse :

```
local function onLandingPlayer(player)
    local menu = player:getFl0MenuRoot()
    menu:remove(1)
    menu:remove(2)
end
```

ou

```
local function onLandingPlayer(player)
    local menu = player:getFl0MenuRoot()
    menu:removeAll()
end
```

Le premier cas supprime toutes les entrées ayant un **groupId** à 1 et à 2.

Le deuxième cas supprime toutes les entrées sans distinction.

Module 1 : Ajout d'un sous-menu et d'une variable spécifique aux joueurs

Ce module reprend la base du module précédent. Les deux entrées fixes permettant d'afficher l'azimuth et l'altitude vont être regroupées dans un sous-menu. Pour gérer le sous menu, une variable submenu va être ajoutée aux joueurs. Pour ce faire, on utilisera :

`player.modules["Module1"].submenu`

"Module1" est le nom du module. Il est possible d'ajouter autant de variables que souhaité.

Code lua

```
-- Advanced Tools for Mission Editor

local thisModule

local F10DisplayCommand
F10DisplayCommand = function(args)
    local player = args[1]

    local text = "Azimuth : " .. player:getAzimuth()
    text = text .. " - Alt MSL : " .. player:getMSLAltitude()

    if player:inAir() == true then
        player:display("In Air : " .. text, 5)
    else
        player:display("On floor : " .. text, 5)
    end
end

local F10DisplayNameCommand
F10DisplayNameCommand = function(args)
    local player = args[1]

    player:display("Unit player name : " .. player:getName(), 5)
end

local F10DisplayPositionCommand
F10DisplayPositionCommand = function(args)
    local player = args[1]

    local position = player:getPosition()
    local lat, lon, alt = ATME.convertPointToLL(position)

    local dirlat, dlat, mlat, slat = ATME.convertToDMS(lat, false)
    local dirlon, dlon, mlon, slon = ATME.convertToDMS(lon, false)

    local westEast = "E"
    if dirlat == -1 then
        westEast = "W"
    end

    local northSouth = "N"
    if dirlon == -1 then
        northSouth = "S"
    end
end
```

```

local text = string.format("Lat : %s%d°%d'%d\"\\n", westEast, dlat, mlat, slat)
text = text .. string.format("Lon : %s%d°%d'%d\"\\n", northSouth, dlon, mlon, slon)

player:display(text, 5)
end

local function onCreatePlayer(player)
    player:display("Welcome in " .. ATME.getTheatre() .. " ... " .. player:getPseudo(), 10)

    player.modules["Module1"].submenu = ATME.C_F10Menu(player, "General informations", nil)
    player.modules["Module1"].submenu:append(1, "Display informations",
                                                F10DisplayCommand , {player})
    player.modules["Module1"].submenu:append(1, "Display unit player name",
                                                F10DisplayNameCommand , {player})

    if player:inAir() == false then
        local menu = player:getF10MenuRoot()
        menu:append(2, "Display position", F10DisplayPositionCommand , {player})
    end
end

local function onTakeOffPlayer(player)
    local menu = player:getF10MenuRoot()
    menu:remove(2)
end

local function onLandingPlayer(player)
    local menu = player:getF10MenuRoot()
    menu:append(2, "Display position", F10DisplayPositionCommand , {player})
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = onTakeOffPlayer,
        onLandingPlayerHandler = onLandingPlayer,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,

        onTimerHandler = nil,
        onModuleStartHandler = nil,
    }

    thisModule = ATME.C_Module("Module1", newHandlers, true)
end

```

Explications

- **onCreatePlayer** est modifiée :
 - Création d'une variable **submenu** attachée aux joueurs. Cette variable est initialisée avec une instance de sous menu créé par le constructeur de la classe **ATME.C_F10Menu**. Le libellé qui s'affichera sera « **General informations** » . Le

dernier paramètre à **nil** définit l'endroit où le sous-menu est créé. Comme dans notre cas, le sous menu est créé directement dans le menu de base (root), **nil** est suffisant. Si le menu avait été créé dans un autre sous menu (appelé **parent**), il aurait fallu passer parent en paramètre.

- Deux entrées sont créées dans le sous-menu (**submenu**). La syntaxe semble complexe, en fait, il s'agit de la variable **submenu** créée suivi de **:append** déjà utilisé précédemment.
- La variable **menu** nécessaire au menu de base pour l'ajout de l'entrée de position est déplacée dans le test car utilisée uniquement là.

Tests et essais

Cette fois un sous menu dans le menu principal est créé : « General informations ... »



Et ce sous menu dispose de deux entrées :



Exercice

Exercice : Supprimer les deux autres menus dès le premier décollage. Ils ne seront plus accessibles.

Réponse :

```
local function onTakeOffPlayer(player)
    local menu = player:getFl0MenuRoot()
    menu:remove(1)
    menu:remove(2)
end
```

ou

```
local function onTakeOffPlayer(player)
    local menu = player:getFl0MenuRoot()
```

```
menu:removeAll()  
end
```

Il n'y a pas de changement par rapport à l'exemple précédent. A l'exécution, et après avoir décollé, le sous menu disparaîtra de l'affichage F10, car il n'y a plus d'item associé.

Module 2 : Interactions DCS et gestion de triggers utilisateur

Jusqu'ici, des affichages ont été réalisés, soit régulièrement, soit grâce à l'utilisation de menus ou sous-menus F10. Ce module aborde les triggers utilisateurs (User Trigger). Pour des raisons de compréhension, un nouveau module est créé, le module « Module2 ».

Pour fonctionner avec ce module, la mission doit comprendre une unité au sol ayant pour nom « Unit to explode ». A défaut, rien ne se passera.

L'objet de ce module est de faire exploser une unité soit par le menu utilisateur soit un bout de 50 secondes. Dès l'explosion, l'entrée F10 est supprimée pour tous les joueurs.

On utilisera aussi la fonction **ATME.C_AIUnit.getByNome** qui permet de retrouver une unité à partir de son nom. Si l'unité n'existe pas, cette fonction retourne **nil**. On notera que les classes **ATME.C_Player**, **ATME.C_Group**, **ATME.StaticObject** et **ATME.C_Race** disposent également de cette fonction.

Une attention particulière sur la cohérence a été faite : Le menu n'apparaît plus aux joueurs après la destruction de l'unité, même si le joueur entre dans la mission.

Code lua

```
local thisModule  
local moduleName = "Module2"  
  
local unitIsDestroyed = false  
  
local function explodeUnit()  
    local unit = ATME.C_AIUnit.getByNome("Unit to explode")  
  
    if unit ~= nil then  
        unit:explode(100)  
        unitIsDestroyed = true  
    end  
end  
  
local F10ExplodeCommand  
F10ExplodeCommand = function()  
    thisModule:removeUserTrigger("TEST1")  
    explodeUnit()  
    ATME.displayForAll("Unité détruite", 5)  
end  
  
local function onCreatePlayer(player)  
    player:display("Welcome in " .. moduleName, 10)  
  
    if unitIsDestroyed == false then  
        local menu = player:getF10MenuRoot()  
        menu:append(1, "Explode unit", F10ExplodeCommand, nil)  
        player.modules["Module2"].menuOK = true  
        if thisModule:userTriggerExists("TEST1") == false then
```

```

        thisModule:createAbsoluteUserTrigger("TEST1", "00:00:50")
    end
else
    player.modules["Module2"].menuOK = false
end
end

local function onUpdatePlayer(player, events)
    if unitIsDestroyed == true and player.modules["Module2"].menuOK == true then
        local menu = player:getF10MenuRoot()
        menu:remove(1)
        player.modules["Module2"].menuOK = false
    end
end

local function onTimer(events)
    if events.isUserTriggerEvent("TEST1") == true then
        explodeUnit()
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = onUpdatePlayer,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,

        onTimerHandler = onTimer,
        onModuleStartHandler = nil,
    }

    thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

Explications

- **explodeUnit** : Cette fonction est appelée soit par la fonction **F10ExplodeCommand** associée au menu soit par un événement lié au trigger utilisateur dans la fonction callback **onTimer**. Elle ne présente rien de compliqué. Elle récupère l'unité « Unit to explode » par **ATME.C_AIUnit.getByName** car une unité AI est une instance de classe **ATME.C_AIUnit**. Si elle existe, l'unité est détruite par une explosion. La variable globale **unitIsDestroyed** initialement à **false** passe à **true** au moment de l'explosion. Cette variable sert à gérer l'entrée menu F10 des joueurs.

- **F10ExplodeCommand** : c'est la fonction appelée par l'entrée du menu F10 des joueurs. Après l'appel de **explodeUnit** , cette fonction supprime le trigger utilisateur **TEST1** qui n'a plus lieu d'être l'unité étant détruite.
- **onCreatePlayer** réalise les traitements suivant uniquement si l'unité n'est pas détruite :
 - Création de l'entrée menu F10 pour les joueurs par la fonction **menu:append**
 - Affectation à true de la variable joueur **player.modules["Module2"].menuOK**. Cette variable est utile pour éviter de passer de multiple fois dans le traitement associé à la suppression des menus dans **onUpdatePlayer**. Même si cela n'engendrera pas de problème, cela pourrait ralentir la mise à jour des menus en cas de gros menus.
 - Création d'un trigger utilisateur (s'il n'existe pas) de nom **TEST1** et dont le déclenchement est fixé à 50s après le début de la mission par la fonction **thisModule:createAbsoluteUserTrigger("TEST1", "00:00:50")**.
- **onUpdatePlayer** est utilisé pour vérifier toutes les secondes si l'unité est détruite, et si oui pour supprimer l'entrée de menu du joueur passé en paramètre. Une fois supprimé, et pour éviter que le traitement ne se refasse la seconde d'après, la variable **player.modules["Module2"].menuOK** est mise à **false** pour chacun des joueurs. Pour rappel, **onUpdatePlayer** est appelé une fois par seconde pour chaque joueur actif.
- **onTimer** va permettre de déclencher l'explosion par contrôle du déclenchement de l'événement associé au trigger utilisateur **TEST1**.

Tests et essais

Le menu apparait en début de mission.



En l'activant :



Et le menu n'apparaît plus :



Par ailleurs, si rien n'est fait avant 50secondes, l'unité explosera sans intervention et le menu disparaîtra de même.

Exercice

Exercice : Créer un deuxième trigger Utilisateur nommé **TEST2** qui se déclenchera 20s après l'explosion de l'unité et qui affichera « End of Mission » à l'ensemble des joueurs. Essayer ensuite en le nommant **TEST1** et vérifier que cela fonctionne. Pourquoi ?

Réponse :

```
local function explodeUnit()  
    local unit = ATME.C_AIUnit.getByname("Unit to explode")  
  
    if unit ~= nil then  
        unit:explode(100)  
        unitIsDestroyed = true  
        thisModule:createAbsoluteUserTrigger("TEST2", "+00:00:20")  
    end  
end
```

Le + devant le délai indique un délai à partir de la création du trigger utilisateur.

```
local function onTimer(events)  
    if events.isUserTriggerEvent("TEST1") == true then  
        explodeUnit()  
    elseif events.isUserTriggerEvent("TEST2") == true then  
        ATME.displayForAll("End of mission", 10)  
    end  
end
```

Réponse à la question : Bien qu'il soit possible de créer un nouveau trigger utilisateur **TEST1**, celui-ci ayant été activé préalablement ou supprimé, le traitement des deux triggers est différent dans la fonction **onTimer**. Ils ne pourront donc pas avoir le même nom.

Module 2 : Gestion de triggers utilisateur relatifs à un flag

L'exemple précédent est repris, mais le trigger utilisateur **TEST1** se déclenchera 20s après l'activation du flag 1 dans l'éditeur de mission, flag armé au bout de 10s. Aussi, l'unité explosera dorénavant une minute après le début de la mission

Code lua

```
local thisModule
local moduleName = "Module2"

local unitIsDestroyed = false

local function explodeUnit()
    local unit = ATME.C_AIUnit.getBy_name("Unit to explode")

    if unit ~= nil then
        unit:explode(100)
        unitIsDestroyed = true
    end
end

local F10ExplodeCommand
F10ExplodeCommand = function()
    thisModule:removeUserTrigger("TEST1")
    explodeUnit()
    ATME.displayForAll("Unité détruite", 5)
end

local function onCreatePlayer(player)
    player:display("Welcome in " .. moduleName, 10)

    if unitIsDestroyed == false then
        local menu = player:getF10MenuRoot()
        menu:append(1, "Explode unit", F10ExplodeCommand, nil)
        player.modules["Module2"].menuOK = true
        if thisModule:userTriggerExists("TEST1") == false then
            thisModule:createFlagRelativeUserTrigger("TEST1", 1, "00:00:50")
        end
    else
        player.modules["Module2"].menuOK = false
    end
end

local function onUpdatePlayer(player, events)
    if unitIsDestroyed == true and player.modules["Module2"].menuOK == true then
        local menu = player:getF10MenuRoot()
        menu:remove(1)
        player.modules["Module2"].menuOK = false
    end
end

local function onTimer(events)
    if events.isUserTriggerEvent("TEST1") == true then
        explodeUnit()
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = onUpdatePlayer,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
```

```

onStopEngineAIUnitHandler = nil,

onCreateGroupHandler = nil,
onDeleteGroupHandler = nil,
onDisableGroupHandler = nil,

onCreateStaticObjectHandler = nil,
onDeleteStaticObjectHandler = nil,

onTimerHandler = onTimer,
onModuleStartHandler = nil,
}

thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

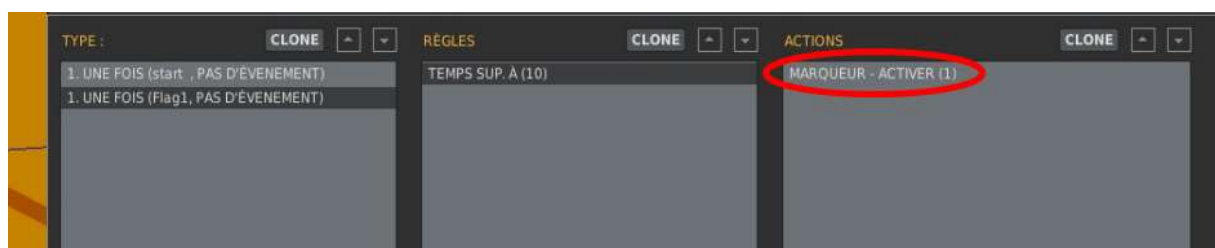
Explications

- **onCreatePlayer** : La fonction **thisModule:createFlagRelativeUserTrigger** permet de créer un trigger utilisateur dépendant de l'activation d'un flag. Ce flag peut être armé dans l'éditeur de mission ou par la fonction **ATME.setFlag**. Dans le cas présent, le deuxième paramètre est à 1 ce qui indique que le trigger utilisateur se déclenchera 50 secondes après l'activation du flag 1.

Tests et essais

Dans un premier temps, le flag 1 n'est pas activé. Il ne se passera rien si aucune action n'est menée par les joueurs.

Dans un deuxième temps, le flag 1 est activé en ajoutant un trigger dans l'éditeur de mission :



Dès lors, sans action de joueur, l'unité explosera au bout de 60 secondes (50 secondes liée au user trigger d'ATME et 10 secondes liée au délai d'activation du flag).

Exercice

Exercice : Armer le flag 2 lors de l'explosion et procéder à l'affichage « End of mission » dans l'éditeur de mission 20s après que ce flag est activé.

Réponse :

```
local function explodeUnit()
    local unit = ATME.C_AIUnit.getByName("Unit to explode")

    if unit ~= nil then
        unit:explode(100)
        unitIsDestroyed = true
        ATME.setFlag(2, true)
    end
end
```

et

```
local function onTimer(events)
    if events.isUserTriggerEvent("TEST1") == true then
        explodeUnit()
    end
end
```

Le trigger utilisateur **TEST2** devient inutile.

Il convient de finaliser en créant les triggers DCS nécessaires dans l'éditeur de mission.

Module 2 : Gestion de triggers utilisateur avec une durée

L'exemple précédent est repris, mais cette fois, si elle n'est pas détruite avant par un joueur, elle va lancer un flare par seconde pendant 10 secondes, après 20 secondes de mission.

Code lua

```
local thisModule
local moduleName = "Module2"

local unitIsDestroyed = false

local function explodeUnit()
    local unit = ATME.C_AIUnit.getByName("Unit to explode")

    if unit ~= nil then
        unit:explode(100)
        unitIsDestroyed = true
    end
end

local F10ExplodeCommand
F10ExplodeCommand = function()
    thisModule:removeUserTrigger("TEST1")
    explodeUnit()
    ATME.displayForAll("Unité détruite", 5)
end

local function onCreatePlayer(player)
    player:display("Welcome in " .. moduleName, 10)

    if unitIsDestroyed == false then
        local menu = player:getF10MenuRoot()
        menu:append(1, "Explode unit", F10ExplodeCommand, nil)
        player.modules["Module2"].menuOK = true
        if thisModule:userTriggerExists("TEST1") == false then
            thisModule:createAbsoluteUserTrigger("TEST1", "00:00:50")
            thisModule:createAbsoluteUserTrigger("TEST2", "00:00:20 +10")
        end
    end
end
```

```

    else
        player.modules["Module2"].menuOK = false
    end
end

local function onUpdatePlayer(player, events)
    if unitIsDestroyed == true and player.modules["Module2"].menuOK == true then
        local menu = player:getF10MenuRoot()
        menu:remove(1)
        player.modules["Module2"].menuOK = false
    end
end

local function onTimer(events)
    if events.isUserTriggerEvent("TEST1") == true then
        explodeUnit()
    elseif events.isUserTriggerEvent("TEST2") == true then
        local unit = ATME.C_AIUnit.getByName("Unit to explode")

        if unit ~= nil then
            unit:fireFlare("RANDOM")
        end
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = onUpdatePlayer,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,

        onTimerHandler = onTimer,
        onModuleStartHandler = nil,
    }

    thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

Explications

- **onCreatePlayer** : Le deuxième paramètre de la fonction **thisModule:createAbsoluteUserTrigger** vaut maintenant **"00:00:20 +10"** ce qui définit une heure de début à 20s du début de mission et un arrêt 10 seconde après. L'événement associé au trigger utilisateur **TEST2** va donc être répété pendant 10 secondes.
- **onTimer** va lancer un flare de couleur aléatoire pendant toute la durée d'activation du trigger utilisateur **TEST2**

Tests et essais

Rien de particulier ici, l'unité va tirer ses flares pendant le laps de temps indiqué :



Exercice

Exercice : Sans rien changer au trigger utilisateur, faire en sorte qu'un seul flare soit envoyé, en d'autres termes, la fonction **unit:fireFlare** ne devra être appelée qu'une seule fois, lors du changement d'état du trigger utilisateur.

Réponse :

```
local function onTimer(events)
    if events.isUserTriggerEvent("TEST1") == true then
        explodeUnit()
    elseif events.isUserTriggerEventToggle("TEST2") == true then
        local unit = ATME.C_AIUnit.getByName("Unit to explode")

        if unit ~= nil then
            unit:fireFlare("RANDOM")
        end
    end
end
```

end

isUserTriggerEventToggle ne s'activera qu'une seule fois même si le triggerUtilisateur a une durée définie.

Module 3 : Données de mission pour les groupes et création dynamique de groupes

Dans ce module, des groupes d'unités vont être créés à partir de groupes défini dans la mission. Dans ATME, c'est le seul moyen de créer un groupe d'unités dynamiquement. Par contre, il est possible aisément de lui affecter les waypoints d'une autre groupe de la mission, tout ou en partie. Il est également possible de créer dynamiquement un groupe à partir d'un groupe dont l'activation est retardée dans l'éditeur de mission.

Actuellement, seuls les groupes d'unités au sol peuvent être dupliqués dynamiquement.

La classe **ATME.C_GroupSpawnDatas** permet la gestion des duplications. Deux fonctions sont nécessaires :

- **ATME.C_GroupSpawnDatas.duplicateFromMissionDatas** qui retourne une instance de la classe **ATME.C_GroupSpawnDatas** et permet de récupérer les données d'un groupe défini dans la mission et de lui attribuer un nouveau nom. Le nom des unités associées sera également modifié
- **datas:spawn** qui permet la création dynamique du nouveau groupe. Si le groupe existe déjà, une erreur sera générée. **datas** est l'instance retournée par la fonction précédente en l'absence d'erreur.

Il est possible de repositionner le nouveau groupe et de lui attribuer tout ou partie des waypoints d'un autre groupe également défini dans la mission.

Ce module s'attache à dupliquer un groupe nommé « origine » en un groupe nommé « copy0 ». Le nouveau groupe sera positionné au hasard dans une zone DCS définie dans l'éditeur de mission et nommée « spawnZone ». La création se fait dès le démarrage de la mission par la fonction callback onStart.

Code lua

```
local thisModule
local moduleName = "Module3"

local function onStart(areUnitsAndPlayersInitialized)
    if areUnitsAndPlayersInitialized == true then
        local err, var1 = ATME.C_GroupSpawnDatas.duplicateFromMissionDatas("origine", "copy0")
        if err == false then
            var1:spawn("spawnZone")
        end
    end
end

-- MAIN
```

```

do
    local newHandlers = {
        onCreatePlayerHandler = nil,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,

        onTimerHandler = nil,
        onModuleStartHandler = onStart,
    }

    thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

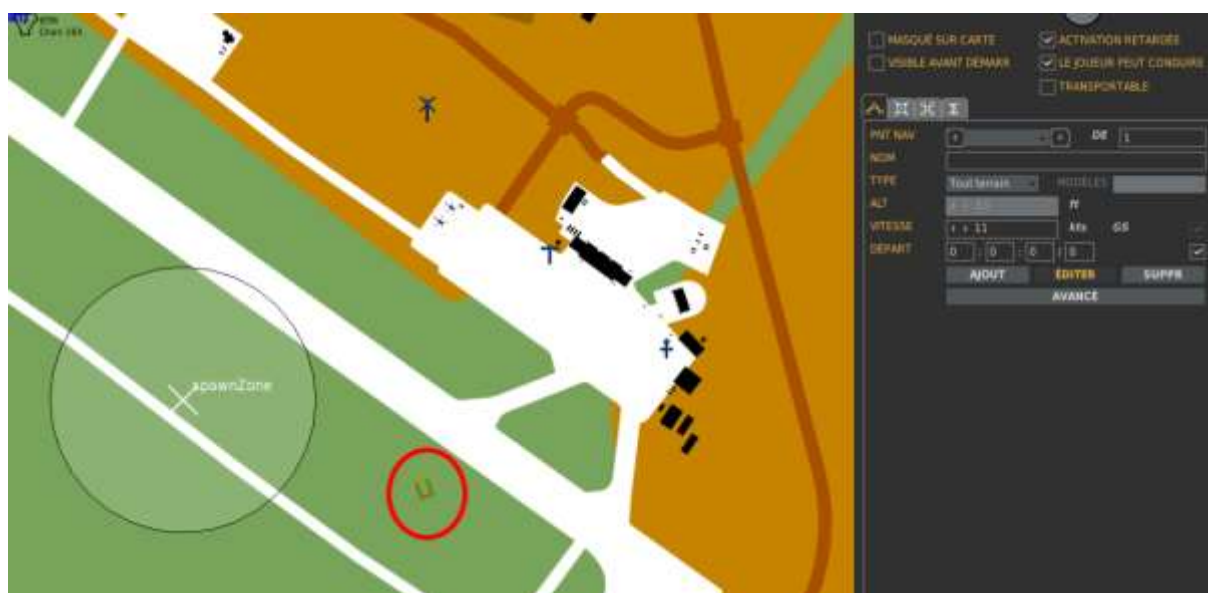
Explications

- **onStart** : Ce module est appelé deux fois lors du démarrage. Si le paramètre **areUnitsAndPlayersInitialized** est à **true**, c'est le deuxième appel après l'initialisation des unités définies dans la mission. Rien de compliqué ; si la copie des données se passe sans erreur, alors le groupe sera dupliqué quelque part dans la zone DCS **spawnZone**.

Il est bien sur possible de combiner les fonctions présentées précédemment pour aboutir à un scénario complexe.

Tests et essais

Le groupe est créé avec une activation retardée.



Puis au lancement de la mission, le nouveau groupe est bien présent dans la bonne zone.



Exercice

Exercice : Dans l'éditeur de mission, associer au moins 4 waypoints au groupe **origine** si cela n'est pas fait. Ajouter une unité sol nommée **reference** à proximité du 3^{ème} waypoint. Puis cocher activer ultérieurement pour que le groupe **origine** n'apparaisse plus. Dans lua, créer dynamiquement un nouveau groupe nommé **copy1** après 10s dans la zone DCS **spawnZone** et lui associer tous les waypoints du groupe **origine** à partir du waypoint le plus proche de l'unité créée.

Réponse :

```
local function onTimer(events)
    if events.isUserTriggerEvent("TEST1") == true then
        local unit = ATME.C_AIUnit.getByname("reference")
        if unit ~= nil then
            local err, var1 = ATME.C_GroupSpawnDatas.duplicateFromMissionDatas("origine",
                                                                                "copy1")

            if err == false then
                var1:spawn("spawnZone", "origine", unit)
            end
        end
    end
end

local function onStart(areUnitsAndPlayersInitialized)
    if areUnitsAndPlayersInitialized == true then
        local err, var1 = ATME.C_GroupSpawnDatas.duplicateFromMissionDatas("origine", "copy0")
        if err == false then
            var1:spawn("spawnZone")
        end
    end

    thisModule:createAbsoluteUserTrigger("TEST1", "00:00:10")
end
```

On crée un trigger utilisateur qui se déclenchera au travers de la fonction callback **onTimer**. La fonction **var1:spawn** permet de récupérer directement tous les waypoints du groupe **origine** à partir du waypoint le plus proche de l'unité **reference**

```
-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = nil,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,

        onTimerHandler = onTimer,
        onModuleStartHandler = onStart,
    }
}
```

```
        thisModule = ATME.C_Module(moduleName, newHandlers, true)
    end
```

onTimer est maintenant utilisé.

Module 4 : Gestion des Core Events

ATME gère des événements au niveau d'ATME Core comme les compteurs pour les courses, les signaux ou l'embarquement d'infanterie dans une unité de transport.

Lorsque ces événements se déclenchent, ils sont transmis à l'ensemble des modules actifs grâce à la classe **ATME.C_EventMgr** au travers des deux fonctions callback **onUpdatePlayerHandler** et **onTimerHandler**. On se reportera à la classe **ATME.C_Module** pour les détails. Le détail des types de Core Event existant dans ATME sont listés dans la classe **ATME.C_EventMgr**.

Cette mission s'attache à signaler un véhicule (dont le groupe se nomme « My Vehicle ») dès qu'un hélicoptère ami survole la zone de détection définie. Comme pour tout signalement automatique, cet hélicoptère doit avoir la capacité de transport de troupes. Ce signalement se fera par une fumée verte et un message sera affiché. La liste des hélicoptères habilité au transport de troupes est donnée dans la fonction **load** des classes **ATME.C_Player** ou **ATME.C_AIUnit**.

Dans DCS, la fumée est présente pendant 5 minutes. Si l'hélicoptère survole la zone au-delà de cette durée, le groupe se resignalera automatiquement dans un délai de 10 à 90 secondes.

Code lua

```
-- Advanced Tools for Mission Editor
local thisModule
local moduleName = "Module4"

local function onTimer(events)
    for _id, _ in events:pairs() do
        if events:isCoreEvent(_id) == true then
            if events:getCoreEventType(_id) == "SIGNAL_UNIT_IN_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous entrez dans la zone", 5)
            elseif events:getCoreEventType(_id) == "SIGNAL_UNIT_OUT_OF_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous sortez de la zone", 5)
            end
        end
    end
end

local function onCreateGroup(_group)
    if _group:getName() == "My Vehicle" then
        _group:setSignal(2500, "SIGNAL_SMOKE GREEN")
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = nil,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
```

```

onLandingPlayerHandler = nil,
onStartEnginePlayerHandler = nil,
onStopEnginePlayerHandler = nil,

onCreateAIUnitHandler = nil,
onDeleteAIUnitHandler = nil,
onDisableAIUnitHandler = nil,
onTakeoffAIUnitHandler = nil,
onLandingAIUnitHandler = nil,
onStartEngineAIUnitHandler = nil,
onStopEngineAIUnitHandler = nil,

onCreateGroupHandler = onCreateGroup,
onDeleteGroupHandler = nil,
onDisableGroupHandler = nil,

onCreateStaticObjectHandler = nil,
onDeleteStaticObjectHandler = nil,

onTimerHandler = onTimer,
onModuleStartHandler = nil,
}

thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

Explications

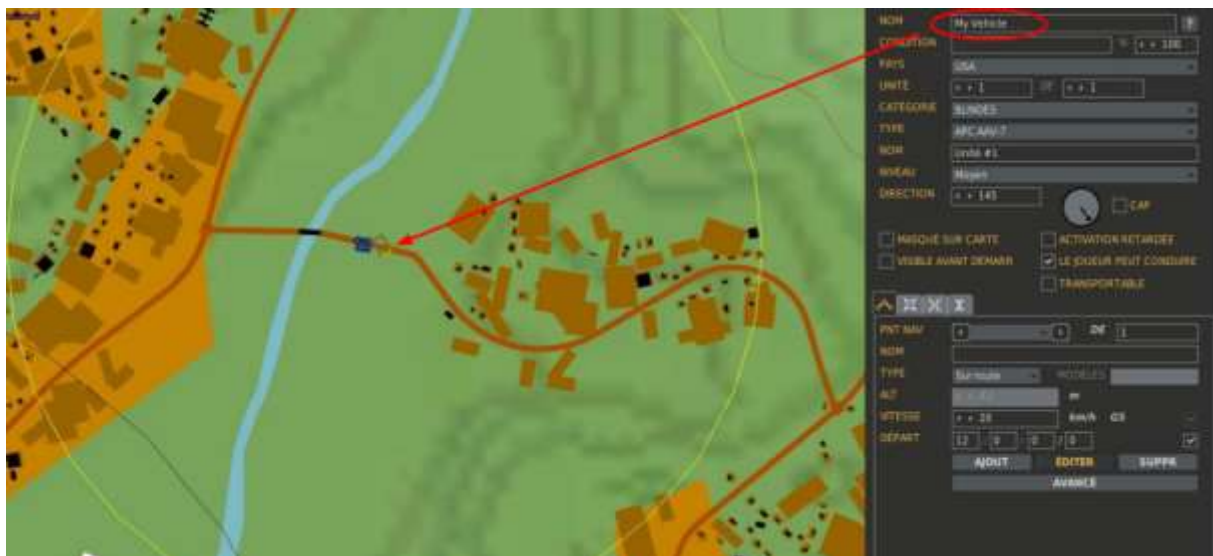
- **onCreateGroup**: Le groupe associé au véhicule active un signallement automatique sur une zone de rayon 3km. Le signallement sera une fumée verte. Il est possible aussi de se signaler par des flares.
- **onTimer** va traiter les événements core par la boucle **for** et la fonction **isCoreEvent**. Un este sur le type permet de connaitre l'événement Core émis. Il est ensuite possible de récupérer les données propres à l'événement et notamment le joueur dans le cas présent (**data.unit** est l'unité entrant ou sortant de la zone).

Tests et essais

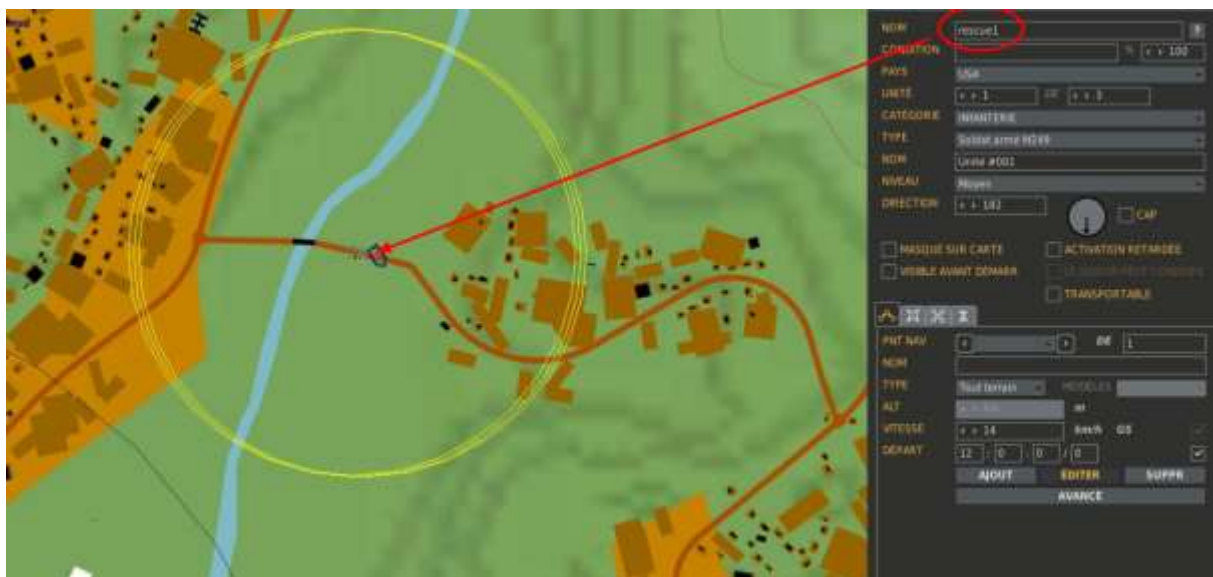
La création des unités se fait sur Sochi pour l'hélicoptère et à proximité pour le transport et le groupe d'infanterie.



Le véhicule de transport de troupe est créé :



Les unités d'infanterie ne servent que pour l'exercice ci après :



Exercice

Exercice : Dans l'éditeur de mission, ajouter un groupe d'infanterie de nom « rescue1 » à proximité (< 200m) du véhicule créé. Vérifier que ce véhicule a une capacité de transport de troupe. Dès que l'hélicoptère se pose à proximité (<500m), le groupe d'infanterie montera à bord immédiatement. Si au moins un hélicoptère entre dans la zone et ressort sans qu'aucun autre ne soit en cours de survol, le groupe embarquera alors dans le véhicule. Informer le joueur dès que le le groupe aura embarqué (dans tous les cas dans l'hélicoptère ou dans le véhicule).

Précision utile : Pour pouvoir embarquer, un groupe d'infanterie doit être prêt à embarquer. Ceci se fait par la fonction **changeReadyToBoard(true)** de la classe **ATME.C_Group**.

Réponse :

```
local nbHelicoptersInZone = 0
local noMoreHelicopterInZone = false

local function onTimer(events)
    local group = ATME.C_Group.getByName("rescue1")

    for _id, _ in events:pairs() do
        if events:isCoreEvent(_id) == true then
            if events:getCoreEventType(_id) == "SIGNAL_UNIT_IN_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous entrez dans la zone", 5)
                nbHelicoptersInZone = nbHelicoptersInZone + 1
            elseif events:getCoreEventType(_id) == "SIGNAL_UNIT_OUT_OF_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous sortez de la zone", 5)
                nbHelicoptersInZone = nbHelicoptersInZone - 1
                if nbHelicoptersInZone == 0 and group ~= nil then
                    noMoreHelicopterInZone = true
                end
            end
        end
    end

    if noMoreHelicopterInZone == true then
        local unit = ATME.C_Group.getByName("My Vehicle"):getFirstUnit()
        unit:load(group, 200)
    end
end

local function onLanding(player)
    local group = ATME.C_Group.getByName("rescue1")
    if group ~= nil then
        local err, errName = player:load(group, 500)

        if err == true then
            if errName == "LOAD_UNIT_TOO_FAR" then
                player:display(displayLabels[ATME.getLanguage()][4], 5)
            else
                player:display("Erreur ... " .. errName, 5)
            end
        end
    end
end

local function onCreateGroup(_group)
    if _group:getName() == "My Vehicle" then
        _group:setSignal(2500, "SIGNAL_SMOKE_GREEN")
    elseif _group:getName() == "rescue1" then
        _group:changeReadyToBoard(true)
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = nil,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = onLanding,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = onCreateGroup,
```

```

onDeleteGroupHandler = nil,
onDisableGroupHandler = nil,

onCreateStaticObjectHandler = nil,
onDeleteStaticObjectHandler = nil,

onTimerHandler = onTimer,
onModuleStartHandler = nil,
}

thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

Les variables **nbHelicoptersInZone** et **noMoreHelicopterInZone** sont créées et initialisées pour gérer les hélicoptères s'il y en a plusieurs (multijoueur). Dès qu'un joueur entre dans la zone, **nbHelicoptersInZone** augmente de 1. Dès qu'un hélicoptère sort de la zone, **nbHelicoptersInZone** baisse de 1. Dès qu'il n'y a plus d'hélicoptère dans la zone, le flag **noMoreHelicopterInZone** passe à vrai si personne n'a récupéré le groupe d'infanterie. Dans ce cas, c'est le véhicule qui embarque le groupe (rayon de 200m pour le chargement, si il est positionné au-delà, rien ne se passera).

Si un hélicoptère se pose à proximité (onLanding, distance max de chargement 500m), que le groupe n'a pas encore été embarqué, il est alors embarqué si la vitesse est inférieure à 3 m/s, sinon une erreur s'affichera. Si l'hélicoptère est trop loin, un message spécifique s'affiche.

Lors de la création du groupe « rescue1 » il est positionné en attente d'embarquement **changeReadyToBoard**.

Exercice en complément : Ajouter un message à tous les joueurs dès que le groupe est embarqué.

```

local function onTimer(events)
    local group = ATME.C_Group.getByname("rescue1")

    for _id, _ in events:pairs() do
        if events:isCoreEvent(_id) == true then
            if events:getCoreEventType(_id) == "SIGNAL_UNIT_IN_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous entrez dans la zone", 5)
                nbHelicoptersInZone = nbHelicoptersInZone + 1
            elseif events:getCoreEventType(_id) == "SIGNAL_UNIT_OUT_OF_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous sortez de la zone", 5)
                nbHelicoptersInZone = nbHelicoptersInZone - 1
                if nbHelicoptersInZone == 0 and group ~= nil then
                    noMoreHelicopterInZone = true
                end
            elseif events:getCoreEventType(_id) == "TRANSPORT_END_OF_BOARDING" then
                ATME.displayForAll("Le groupe d'infanterie a été embarqué", 5)
            end
        end
    end

    if noMoreHelicopterInZone == true then
        local unit = ATME.C_Group.getByname("My Vehicle"):getFirstUnit()
        unit:load(group, 200)
    end
end

```

Module 4 : Gestion des pickupzones

Les pickupzones sont des zones DCS où les groupes d'infanteries se mettent en attente pour embarquer. Les groupes d'infanterie doivent impérativement se trouver à un moment donné dans la zone. Aussi le chemin emprunté devra passer au travers de la zone.

Ces zones peuvent être signalées au travers des groupes d'infanterie grâce à la fonction **setSignal** de la classe **ATME.C_Group**. Dans ce cas, si plusieurs groupes ont cette possibilité, seul un s'occupera du signalement auprès des hélicoptères habilités de la coalition. S'il est détruit, un autre prendra le relais si ce dernier s'est également signalé.

Le signalement n'est pas obligatoire.

Code lua

```
-- Advanced Tools for Mission Editor
local thisModule
local moduleName = "Module4"

local function onTimer(events)
    for _id, _ in events:pairs() do
        if events:isCoreEvent(_id) == true then
            if events:getCoreEventType(_id) == "SIGNAL_UNIT_IN_ZONE" then
                local datas = events:getCoreEventDatas(_id)
                datas.unit:display("Vous entrez dans la zone", 5)
            end
        end
    end
end

local function onCreateGroup(_group)
    if _group:getName() == "rescue1" or _group:getName() == "rescue2" then
        _group:setPickupZone("ZonePickup1", 1)
        _group:setSignal(2500, "SIGNAL_SMOKE RED")
    end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = nil,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = onCreateGroup,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,
    }
```



```
        onTimerHandler = onTimer,  
        onModuleStartHandler = nil,  
    }  
  
    thisModule = ATME.C_Module(moduleName, newHandlers, true)  
end
```

Explications

- **onCreateGroup**: En l'état, deux groupes d'infanterie « rescue1 » et « rescue2 » . Ces deux groupes passent par la zone DCS « ZonePickup1 » par construction dans l'éditeur de mission. Dès que ces groupes seront dans la zone, ils se mettent en attente. La fumée ne s'activera que lorsqu'un hélicoptère sera dans la zone définie par la fonction **setSignal** (ici 2.5km).

Le reste est entièrement automatique. Ce test peut être complété par le module ATME_Rescue accessible en téléchargement. Ce module permet de gérer les embarquements/débarquements de troupes d'infanterie.

Tests et essais

La création des 2 groupes d'infanterie se fait à proximité de Sochi. L'hélicoptère est sur Sochi.

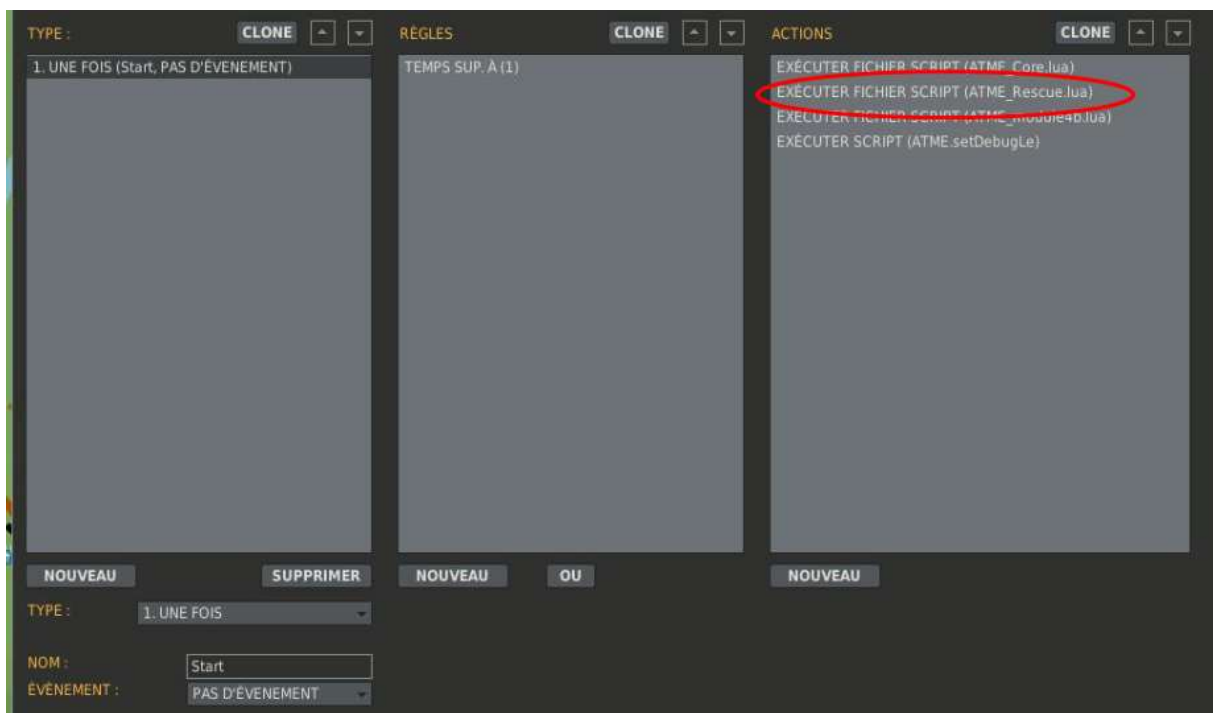


Au démarrage de la mission, et avant de décoller, on observera les groupes d'infanterie en mouvement et se positionnant en attente sur la zone.

Quand l'hélicoptère entre dans la zone, une fumée rouge (unique pour les deux groupes) s'enclenche.

Exercice

Exercice : Ce test peut être complété par le module ATME_Rescue accessible en téléchargement. Ce module permet de gérer les embarquements/débarquements de troupes d'infanterie. Le télécharger et l'intégrer dans la mission pour ramener les groupes d'infanterie sur Sochi avec l'hélicoptère.



Module 4 : Gestion multilingue

ATME supporte la gestion multilingue. Les messages de l'exercice « Module 4 : Gestion des Core Events » vont être traduits en Anglais. Le module supportera alors l'anglais et le français.

Le fonctionnement est simple :

- Chaque message doit être inséré dans une table qui elle même dispose d'une entrée pour le français, FR, et d'une entrée pour l'anglais, EN. Cette table sera locale au module. Elle pourra s'appeler `displayLabels`
- Les messages avec des valeurs dynamiques seront insérés grâce à la fonction **`string.format`** de lua.

Exemple :

```
local displayLabels = {
  FR = {
    "Transport de troupes",
    "Embarquer un groupe",
    "Faire débarquer %s",
    "Vous n'êtes pas posé !",
    "Vous êtes trop loin du groupe à secourir !",
    "Groupe %s embarqué !",
    "Groupe %s débarqué sans pertes !",
    "Groupe %s débarqué avec une perte !",
    "Groupe %s débarqué avec %d pertes !",
    "Il reste %d unités à bord.",
    "Il ne reste aucune unité à bord.",
    "Vous ne pouvez pas débarquer d'unité ici, trop de présence d'eau !",
    "Tout le groupe s'est noyé.",
    "Aucun groupe à récupérer.",
    "Vous n'avez pas assez de place pour embarquer tout le groupe !"
  },
  EN = {
    "Personnel carrier",
    "Load group",
    "Unload %s",
    "You're in air !",
    "You're too far from troops to rescue !",
    "Group %s on board !",
    "Group %s disembarked without losses !",
    "Group %s disembarked with losses !",
    "Group %s disembarked with losses !",
    "There's %d units on board.",
    "No more unit on board.",
    "You can't disembark units here. Too much water !",
    "All group drowned.",
    "No group to rescue.",
    "You can't embark all units !"
  },
}
```

Code lua

```
-- Advanced Tools for Mission Editor
local thisModule
local moduleName = "Module4"

local nbHelicoptersInZone = 0
local noMoreHelicopterInZone = false

local displayLabels = {
  FR = {
    "Vous entrez dans la zone",
    "Vous sortez de la zone",
    "Le groupe d'infanterie a été embarqué",
    "Vous vous etes posé trop loin",
  },
  EN = {
    "You enter in zone",
    "You exit the zone",
    "Infantry group is on board",
    "You land too far",
  },
}

local function onTimer(events)
  local group = ATME.C_Group.getByName("rescue1")
```

```

for _id, _ in events:pairs() do
    if events:isCoreEvent(_id) == true then
        if events:getCoreEventType(_id) == "SIGNAL_UNIT_IN_ZONE" then
            local datas = events:getCoreEventDatas(_id)
            datas.unit:display(displayLabels[ATME.getLanguage()][1], 5)
            nbHelicoptersInZone = nbHelicoptersInZone + 1
        elseif events:getCoreEventType(_id) == "SIGNAL_UNIT_OUT_OF_ZONE" then
            local datas = events:getCoreEventDatas(_id)
            datas.unit:display(displayLabels[ATME.getLanguage()][2], 5)
            nbHelicoptersInZone = nbHelicoptersInZone - 1
            if nbHelicoptersInZone == 0 and group ~= nil then
                noMoreHelicopterInZone = true
            end
        elseif events:getCoreEventType(_id) == "TRANSPORT_END_OF_BOARDING" then
            ATME.displayForAll(displayLabels[ATME.getLanguage()][3], 5)
        end
    end
end

if noMoreHelicopterInZone == true then
    local unit = ATME.C_Group.getBy_name("My Vehicle"):getFirstUnit()
    unit:load(group, 200)
end

end

local function onLanding(player)
    local group = ATME.C_Group.getBy_name("rescue1")

    if group ~= nil then
        local err, errName = player:load(group, 500)

        if err == true then
            if errName == "LOAD_UNIT_TOO_FAR" then
                player:display(displayLabels[ATME.getLanguage()][4], 5)
            else
                player:display("Erreur ... " .. errName, 5)
            end
        end
    end
end

end

local function onCreateGroup(_group)
    if _group:get_name() == "My Vehicle" then
        _group:setSignal(2500, "SIGNAL_SMOKE_GREEN")
    elseif _group:get_name() == "rescue1" then
        _group:changeReadyToBoard(true)
    end
end

end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = nil,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = nil,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = onLanding,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onDisableAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = onCreateGroup,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,
    }

```

```
    onTimerHandler = onTimer,  
    onModuleStartHandler = nil,  
  }  
  
  thisModule = ATME.C_Module(moduleName, newHandlers, true)  
end
```

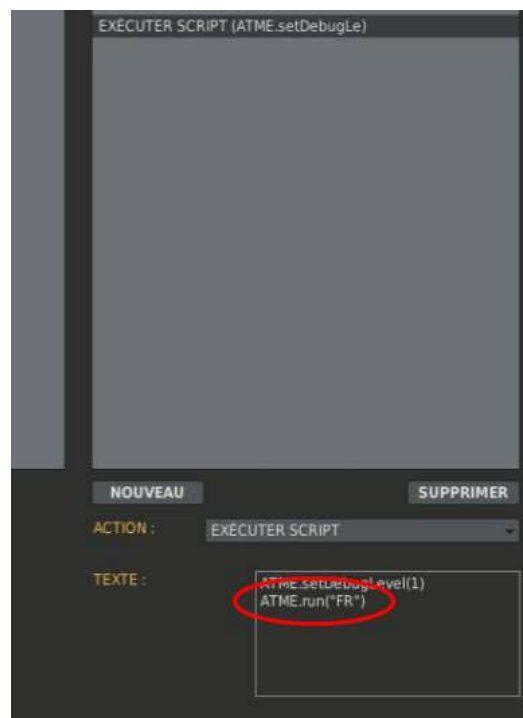
Explications

En surligné, l'évolution multilingue

- **displayLabels**: Cette table contient tous les textes en anglais et en français. L'ordre de traduction est essentiel, chaque index de message pour l'ensemble des langues doit être identique
- Chaque affichage fera appel au bon label et utilisant la fonction **ATME.getLanguage** et le bon index.

Tests et essais

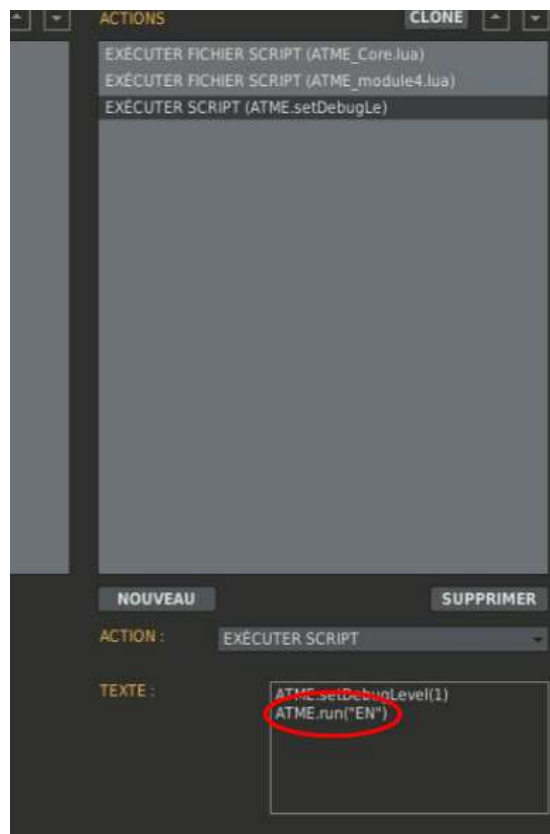
Le choix de la langue se fera ensuite dans l'éditeur de mission, au travers de la fonction **ATME.run**.



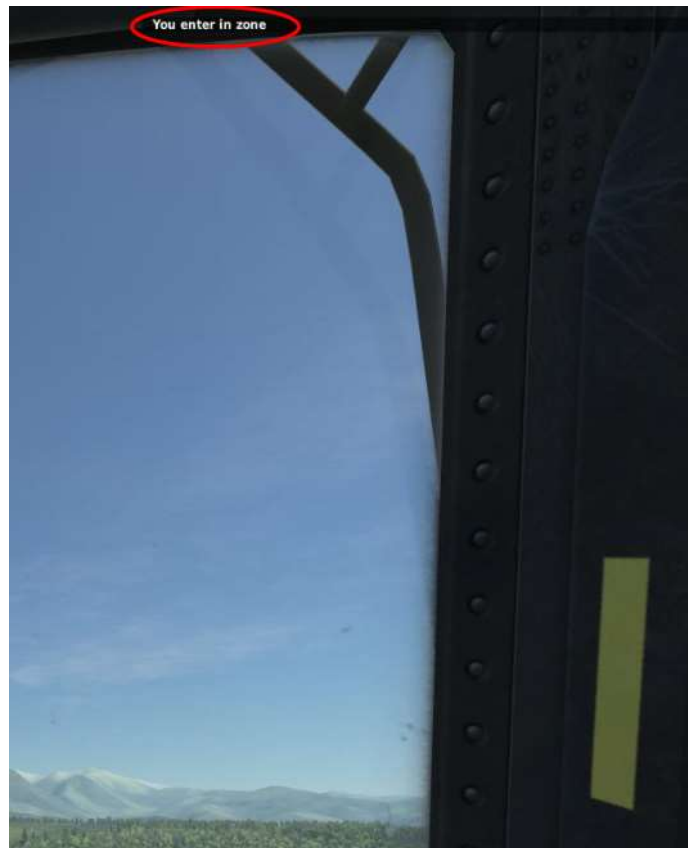
La langue choisie est le français et on obtient l’affichage :



La langue choisie est l’anglais :



On obtient l’affichage :



Module 5 : Gestion des courses

ATME offre la capacité de créer et de gérer des courses avec mesure des temps et classements.

Ce module gère une petite course autour de Batumi avec quelques portes, une mesure de temps intermédiaire et final. La distance maximale entre deux piquets d’une porte est de 500m.

Code lua

```
-- Advanced Tools for Mission Editor
local thisModule
local moduleName = "Module5"

local function onCreatePlayer(player)

    local race = ATME.C_Race.getBy_name("Ma course")

    race:addPlayer(player)
end

local function onTimer(events)
    for _id, _ in events:pairs() do
        if events:isCoreEvent(_id) == true then
            thisModule:output(events:getCoreEventType(_id), 1)
            local typeEvent = events:getCoreEventType(_id)
```

```

local datas = events:getCoreEventDatas(_id)

if ATME.isObjectClass(datas.unit, ATME.C_Player) == true then
    if typeEvent == "RACE_START" then
        datas.unit:display("Départ ...", 5)
    elseif typeEvent == "RACE_TURN_TIME" then
        local err, h, m, s, d = ATME.convertToHMSd(datas.playTime)
        local text = string.format("%02d:%02d:%02d", m, s, d)
        datas.unit:display("Fin du tour N°" .. datas.turnNumber ..
            " - Temps : " .. text, 5)

    elseif typeEvent == "RACE_BEST_TURN_TIME" then
        local err, h, m, s, d = ATME.convertToHMSd(datas.playTime)
        local text = string.format("%02d:%02d:%02d", m, s, d)
        datas.race:displayForAllPlayers("Nouveau record du tour pour " ..
            datas.unit:getPseudo() .. " : " .. text,
            5)

    elseif typeEvent == "RACE_PLAYER_BEST_TURN_TIME" then
        datas.unit:display("Votre meilleur tour", 5)

    elseif typeEvent == "RACE_LAP_TIME" then
        local err, h, m, s, d = ATME.convertToHMSd(datas.playTime)
        local text = string.format("%02d:%02d:%02d", m, s, d)
        datas.unit:display("Porte N°" .. datas.doorNumber ..
            " - Temps : " .. text, 5)

    elseif typeEvent == "RACE_BEST_LAP_TIME" then
        local err, h, m, s, d = ATME.convertToHMSd(datas.playTime)
        local text = string.format("%02d:%02d:%02d", m, s, d)
        datas.race:displayForAllPlayers("Meilleur tps intermédiaire Porte N° " ..
            datas.doorNumber .. " pour " ..
            datas.unit:getPseudo() .. " : " .. text,
            5)

    elseif typeEvent == "RACE_BEST_FINAL_TIME" then
        datas.race:displayForAllPlayers("Nouveau record pour " ..
            datas.unit:getPseudo(), 5)

    elseif typeEvent == "RACE_PLAYER_BEST_FINAL_TIME" then
        datas.unit:display("Vous battez votre record de course", 5)

    elseif typeEvent == "RACE_FINAL_TIME" then
        local err, h, m, s, d = ATME.convertToHMSd(datas.totalPlayTime)
        local text = "Temps final pour " .. datas.unit:getPseudo() .. " : " ..
            string.format("%02d:%02d:%02d", m, s, d) .. "\n"
        local ranking = datas.race:getRanking()

        text = text .. "Classement général :\n"
        for _i, _item in ipairs(ranking) do
            local err, h, m, s, d = ATME.convertToHMSd(_item.bestTime)
            local timeText = "--:--:--"
            if err == false then
                timeText = string.format("%02d:%02d:%02d", m, s, d)
            end
            text = text .. string.format("%2d - ", _i) ..
                _item.player:getPseudo() .. " avec " .. timeText .. "\n"
        end

        datas.race:displayForAllPlayers(text, 5)

    elseif typeEvent == "RACE_DOOR_MISSED" then
        datas.unit:display("Porte N°" .. datas.doorNumber ..
            " loupée : pénalité de 10s", 5)

    elseif typeEvent == "RACE_PLAYER_TOO_HIGH" then
        datas.unit:display("Altitude trop élevée : pénalité de 10s", 5)
    end
end
end
end
end

local function startOn(areUnitsAndPlayersInitialized)
    if areUnitsAndPlayersInitialized == false then
        local myRace = ATME.C_Race("Ma course", "PiquetG", "PiquetD", 2)
    end
end

```



```

    myRace:setMaxAltitudeRule(200, 10)
    myRace:setMissedDoorRule(10)

    if myRace ~= nil then
        myRace:setLapAtDoor(4)
    end
end
end

-- MAIN
do
    local newHandlers = {
        onCreatePlayerHandler = onCreatePlayer,
        onDeletePlayerHandler = nil,
        onUpdatePlayerHandler = onUpdatePlayer,
        onTakeoffPlayerHandler = nil,
        onLandingPlayerHandler = nil,
        onStartEnginePlayerHandler = nil,
        onStopEnginePlayerHandler = nil,

        onCreateAIUnitHandler = nil,
        onDeleteAIUnitHandler = nil,
        onTakeoffAIUnitHandler = nil,
        onLandingAIUnitHandler = nil,
        onStartEngineAIUnitHandler = nil,
        onStopEngineAIUnitHandler = nil,

        onCreateGroupHandler = nil,
        onDeleteGroupHandler = nil,
        onDisableGroupHandler = nil,

        onCreateStaticObjectHandler = nil,
        onDeleteStaticObjectHandler = nil,

        onTimerHandler = onTimer,
        onModuleStartHandler = startOn,
    }

    thisModule = ATME.C_Module(moduleName, newHandlers, true)
end

```

Explications

- **onCreatePlayer** va permettre d'associer les joueurs à la course. La gestion est ensuite automatique. Il est bien sûr possible de désinscrire un joueur à tout moment.
- **onStart** va créer la course de nom « Ma course ». Cette course possède deux tours. Des pénalités sont associées en cas de hauteur maximale dépassée ou de porte loupée.
- **onTimer** va gérer les événements de course et donc l'affichage des temps ou des pénalités.

Tests et essais

Dans l'éditeur de mission, créer deux avions (warbirds) et la course (ensemble de piquets) :



Les piquets représentés ci-dessus par des rectangles sont numérotés de #001 à #006 (sans piquets #003) :

| Side | Port | Etat | QTR |
|------------|------|------|-----|
| Piquet 001 | 100A | 1 | 1 |
| Piquet 002 | 100A | 1 | 1 |
| Piquet 004 | 100A | 1 | 1 |
| Piquet 005 | 100A | 1 | 1 |
| Piquet 006 | 100A | 1 | 1 |
| Piquet 007 | 100A | 1 | 1 |
| Piquet 008 | 100A | 1 | 1 |
| Piquet 009 | 100A | 1 | 1 |
| Piquet 010 | 100A | 1 | 1 |
| Piquet 011 | 100A | 1 | 1 |
| Piquet 012 | 100A | 1 | 1 |

Un piquet gauche d'un numéro donné est associé à un piquet droit d'un numéro donné pour formé une porte.

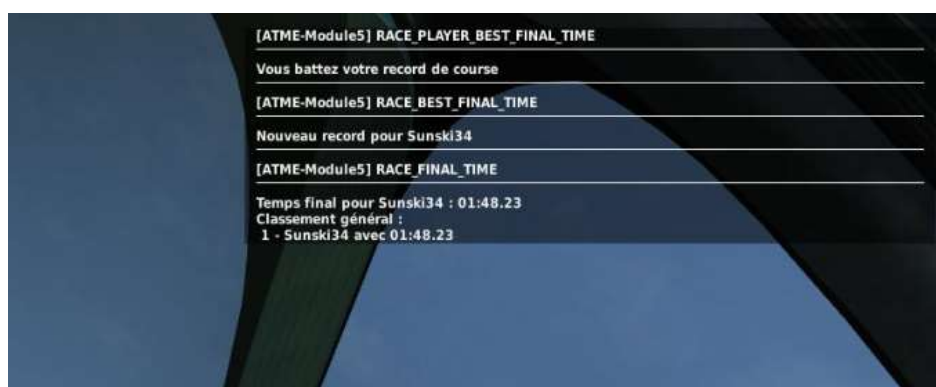
Pendant la course, temps intermédiaire :



Après un tour :



A la fin de course :



Le classement comportera plusieurs joueurs s'il y en a plusieurs.

Fonctions globales de la table ATME

ATME.arePointsVisible(pA, pB)

Description : Fonction vérifiant si deux points sont visibles l'un de l'autre, sans prise en compte des bâtiments et des obstacles éventuels autre que terrain.

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True si les deux points se « voient », false sinon. |
| Si erreur : | nil | |

ATME.convertFtstoM(distance)

Description : Fonction convertissant en mètres une distance exprimée en pieds.

Paramètres :

| | | |
|-----------------|--------|-------------------|
| distance | number | Distance en pieds |
|-----------------|--------|-------------------|

Valeurs de retour :

| | | |
|---------------|--------|-----------------------------|
| Sans erreur : | number | Distance convertie en pieds |
| Si erreur : | nil | |

ATME.convertKphtoMps(speed)

Description : Fonction convertissant en m/s une vitesse exprimée en km/h.

Paramètres :

| | | |
|--------------|--------|-----------------|
| speed | number | Vitesse en km/h |
|--------------|--------|-----------------|

Valeurs de retour :

| | | |
|---------------|--------|--------------------------|
| Sans erreur : | number | Vitesse convertie en m/s |
| Si erreur : | nil | |

ATME.convertLLToMGRS(latitude, longitude)

Description : Fonction permettant la conversion d'une latitude et d'une longitude en une table au format MGRS.

Paramètres :

| | | |
|------------------|--------|----------------------|
| latitude | number | Latitude en degrés |
| longitude | number | Longitude en degrés. |

Valeurs de retour :

| | | |
|---------------|-------|------------------------|
| Sans erreur : | table | Table MGRS initialisée |
| Si erreur : | nil | |

ATME.convertLLToPoint(latitude, longitude, altitude)

Description : Fonction permettant la conversion d'une latitude, longitude et altitude en un point 3D.

Paramètres :

| | | |
|------------------|--------|--|
| latitude | number | Latitude en degrés |
| longitude | number | Longitude en degrés. |
| altitude | number | Altitude à partir du niveau de la mer en m |

Valeurs de retour :

| | | |
|---------------|-------|-----------------------|
| Sans erreur : | table | Point 3D en x, y et z |
| Si erreur : | nil | |

ATME.convertMGRSToLL(tableMGRS)

Description : Fonction permettant la conversion d'une table MGRS correctement initialisée en latitude/longitude.

Paramètres :

| | | |
|------------------|-------|-------------------------------------|
| tableMGRS | table | Table MGRS correctement initialisée |
|------------------|-------|-------------------------------------|

Valeurs de retour :

| | | |
|---------------|--------|---------------------|
| Sans erreur : | number | Latitude en degrés |
| | number | Longitude en degrés |
| Si erreur : | nil | |

ATME.convertMpstoKph(speed)

Description : Fonction convertissant en km/h une vitesse exprimée en m/s.

Paramètres :

| | | |
|--------------|--------|----------------|
| speed | number | Vitesse en m/s |
|--------------|--------|----------------|

Valeurs de retour :

| | | |
|---------------|--------|---------------------------|
| Sans erreur : | number | Vitesse convertie en km/h |
| Si erreur : | nil | |

ATME.convertMpstoNds(speed)

Description : Fonction convertissant en noeuds une vitesse exprimée en m/s.

Paramètres :

| | | |
|--------------|--------|----------------|
| speed | number | Vitesse en m/s |
|--------------|--------|----------------|

Valeurs de retour :

| | | |
|---------------|--------|-----------------------------|
| Sans erreur : | number | Vitesse convertie en noeuds |
| Si erreur : | nil | |

ATME.convertMtoFts(distance)

Description : Fonction convertissant en pieds une distance exprimée en mètres.

Paramètres :

| | | |
|-----------------|--------|---------------|
| distance | number | distance en m |
|-----------------|--------|---------------|

Valeurs de retour :

| | | |
|---------------|--------|-----------------------------|
| Sans erreur : | number | distance convertie en pieds |
| Si erreur : | nil | |

ATME.convertMtoNM(distance)

Description : Fonction convertissant en miles nautiques une distance exprimée en mètres.

Paramètres :

| | | |
|-----------------|--------|---------------|
| distance | number | distance en m |
|-----------------|--------|---------------|

Valeurs de retour :

| | | |
|---------------|--------|-------------------------------------|
| Sans erreur : | number | distance convertie en mile nautique |
| Si erreur : | nil | |

ATME.convertNdstoMps(speed)

Description : Fonction convertissant en m/s une vitesse exprimée en noeuds.

Paramètres :

| | | |
|--------------|--------|-------------------|
| speed | number | Vitesse en noeuds |
|--------------|--------|-------------------|

Valeurs de retour :

| | | |
|---------------|--------|--------------------------|
| Sans erreur : | number | Vitesse convertie en m/s |
| Si erreur : | nil | |

ATME.convertNMtoM(distance)

Description : Fonction convertissant en mètres une distance exprimée en miles nautiques.

Paramètres :

| | | |
|-----------------|--------|---------------------------|
| distance | number | distance en mile nautique |
|-----------------|--------|---------------------------|

Valeurs de retour :

| | | |
|---------------|--------|-------------------------|
| Sans erreur : | number | distance convertie en m |
| Si erreur : | nil | |

ATME.convertPointToLL(point)

Description : Fonction permettant la conversion d'un point 3D en latitude, longitude et altitude.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|---------------|--------|---------------------------------|
| Sans erreur : | number | Latitude en degrés |
| | number | Longitude en degrés |
| | number | Altitude relative à la mer en m |
| Si erreur : | nil | |

ATME.convertToDMS(angle, secDec)

Description : Fonction permettant la conversion d'une latitude ou d'une longitude décimale en latitude ou longitude en degrés, minutes, secondes ou degrés, minutes, minute décimale.

Paramètres :

| | | |
|---------------|---------|--|
| angle | number | Angle à convertir (idéalement -180° à +180°) |
| secDec | boolean | Si false, conversion en degrés, minute, secondes. Sinon conversion degrés, minutes et centième de minutes. |

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | number | Direction 1 si angle >= 0, -1 sinon |
| | number | Degrés entiers en valeur absolue |
| | number | Minutes entières en valeur absolue |
| | number | Centièmes entiers de minutes ou secondes entières suivant secDec |
| Si erreur : | nil | |

ATME.convertToHMSd(seconds)

Description : Fonction permettant la conversion de secondes en heures, minutes, secondes et 1/100^{ème} de secondes.

Paramètres :

| | | |
|----------------|--------|----------------------|
| seconds | number | Secondes à convertir |
|----------------|--------|----------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | False, seconds >= 0, true sinon dans ce dernier cas, pas d'autres retours |
| | number | Heures si pas d'erreur |
| | number | Minutes si pas d'erreur |
| | number | Secondes si pas d'erreur |
| | number | 1/100 ^{ème} de secondes si pas d'erreur |
| Si erreur : | nil | |

ATME.convertToPoint3D(point)

Description : Fonction convertissant un point 2D (x,y) en 3D (x,y,z). Si le point est déjà 3D, la fonction ne fait rien.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|---------------|-------|-----------------------|
| Sans erreur : | table | Point 3D en x, y et z |
| Si erreur : | nil | |

ATME.displayForAll(text, duration)

Description : Cette fonction permet d'afficher un message destiné à tous les joueurs de la mission.

Paramètres :

| | | |
|-----------------|--------|--|
| text | string | Texte du message à afficher. |
| duration | number | Durée en secondes de l'affichage du message. |

Valeurs de retour : Aucune

ATME.displayForCoalition(coalitionName, text, duration)

Description : Cette fonction permet d'afficher un message destiné à tous les joueurs d'une coalition.

Paramètres :

| | | |
|----------------------|--------|--|
| coalitionName | string | Nom de la coalition : "NEUTRAL", "RED" ou "BLUE" |
| text | string | Texte du message à afficher. |
| duration | number | Durée en secondes de l'affichage du message. |

Valeurs de retour : Aucune

ATME.displayForCountry(countryName, text, duration)

Description : Cette fonction permet d'afficher un message destiné à tous les joueurs d'un pays.

Paramètres :

| | | |
|--------------------|--------|--|
| countryName | string | Nom du pays |
| text | string | Texte du message à afficher. |
| duration | number | Durée en secondes de l'affichage du message. |

Valeurs de retour : Aucune

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

ATME.explode(point, power)

Description : Cette fonction déclenche une explosion de force donnée au point défini.

Paramètres : Aucun

| | | |
|--------------|--------|--|
| point | table | Point en x, y et z, ou point en x,y où l'explosion sera déclenchée |
| power | number | Puissance de l'explosion |

Valeurs de retour : Aucune

ATME.getAIUnits()

Description : Fonction retournant la liste de toutes les unités AI en vie (actives) à un instant T.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|---|
| | table | Table sur les instances actives de C_AIUnit |
|--|-------|---|

ATME.getFlag(id)

Description : Fonction permettant de récupérer la valeur d'un marqueur. La valeur peut être modifiée par un trigger de l'éditeur de mission.

Paramètres :

| | | |
|-----------|--------|-------------------|
| id | number | Numéro du drapeau |
|-----------|--------|-------------------|

Valeurs de retour :

| | | |
|--------------------------|--------|--------------------|
| Sans erreur : | number | Valeur du drapeau. |
| Si id incorrect : | nil | |

ATME.getGroups()

Description : Fonction retournant la liste de tous les groupes en vie (actifs) à un instant T.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Table sur les instances actives de C_Group |
|-------|--|

ATME.getLanguage()

Description : Fonction retournant la langue utilisée pour les messages affichés.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| string | Langue utilisée : "FR" pour le français "EN" pour l'anglais. |
|--------|--|

ATME.getPlayers()

Description : Fonction retournant la liste de toutes les unités de joueur en vie à un instant T.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Table sur les instances en vie de C_Player |
|-------|--|

ATME.getPointAltitude(point)

Description : Fonction retournant l'altitude d'un point donné. Pour un point 2D en x,y, l'altitude sera l'altitude du terrain.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|------------------------------|--------|---|
| Sans erreur : | number | Altitude du point en m, relatif à la mer. |
| Si point incohérent : | nil | |

ATME.getSurfaceAltitude(point)

Description : Fonction retournant l'altitude du terrain à un point donné.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|------------------------------|--------|--|
| Sans erreur : | number | Altitude du terrain au point donné en m, relatif à la mer. |
| Si point incohérent : | nil | |

ATME.getTheatre()

Description : Fonction retournant la map active de la mission.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| string | Nom de la map active : "CAUCASUS" "NEVADA" |
|--------|--|

ATME.getTime()

Description : Fonction permettant de récupérer le nombre de secondes depuis le début de la mission.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|--------------------|
| | number | Nombre de secondes |
|--|--------|--------------------|

ATME.getWindAzimuth(point3D, withTurbulence)

Description : Fonction permettant de connaître la direction horizontale du vent à un point donné.

Paramètres :

| | | |
|-----------------------|---------|---|
| Point3D | table | Point 3D en x, y et z |
| withTurbulence | boolean | Si true prise en compte des turbulences, sinon false. |

Valeurs de retour :

| | | |
|---------------|--------|-------------------|
| Sans erreur : | number | Azimuth 0 à 360°. |
| Si erreur : | nil | |

ATME.getWindHSpeed(point3D, withTurbulence)

Description : Fonction permettant de connaître la vitesse horizontale du vent à un point donné. Le vent peut être évalué avec ou sans les turbulences.

Paramètres :

| | | |
|-----------------------|---------|---|
| Point3D | table | Point 3D en x, y et z |
| withTurbulence | boolean | Si true prise en compte des turbulences, sinon false. |

Valeurs de retour :

| | | |
|---------------|--------|-------------------------------------|
| Sans erreur : | number | Vitesse horizontale du vent en m/s. |
| Si erreur : | nil | |

ATME.getWindSpeed(point3D, withTurbulence)

Description : Fonction permettant de connaître la vitesse du vent à un point donné, calculée sur les 3 axes. Le vent peut être évalué avec ou sans les turbulences.

Paramètres :

| | | |
|-----------------------|---------|---|
| Point3D | table | Point 3D en x, y et z |
| withTurbulence | boolean | Si true prise en compte des turbulences, sinon false. |

Valeurs de retour :

| | | |
|---------------|--------|-------------------------|
| Sans erreur : | number | Vitesse du vent en m/s. |
| Si erreur : | nil | |

ATME.getWindVector(point3D, withTurbulence)

Description : Fonction permettant de connaître le vecteur vitesse du vent à un point donné. Le vent peut être évalué avec ou sans les turbulences.

Paramètres :

| | | |
|-----------------------|---------|---|
| Point3D | table | Point 3D en x, y et z |
| withTurbulence | boolean | Si true prise en compte des turbulences, sinon false. |

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Instance C_Vecteur3D représentant le vecteur vitesse du vent en m/s. |
| Si erreur : | nil | |

ATME.getDCSZone(zoneName)

Description : Fonction permettant de récupérer les infos de centre et rayon d'une zone DCS définie dans la mission.

Paramètres :

| | | |
|-----------------|--------|-------------|
| zoneName | string | Nom la zone |
|-----------------|--------|-------------|

Valeurs de retour :

| | | |
|---------------|-------|---|
| Sans erreur : | table | Table zone contenant le centre (zone.point) et le rayon (zone.radius) de la zone nommée. |
| Si erreur : | nil | |

ATME.isObjectClass(object, class)

Description : Fonction vérifiant si un objet est d'une classe donnée.

Paramètres :

| | | |
|---------------|-------|------------------------------|
| object | table | Objet à vérifier |
| class | table | Classe pour vérifier l'objet |

Valeurs de retour :

| | |
|---------|--|
| boolean | True si object est de classe class , false sinon |
|---------|--|

ATME.isPoint(tableToVerify)

Description : Fonction vérifiant si la table point est bien un point 2D ou 3D.

Paramètres :

| | | |
|----------------------|-------|------------------|
| tableToVerify | table | Table à vérifier |
|----------------------|-------|------------------|

Valeurs de retour :

| | |
|--------|---|
| number | ATME.POINT_2D pour un point 2D (x,y) ATME.POINT_3D pour un point 3D (x,y et z) ATME.POINT_BAD si la table n'est pas un point |
|--------|---|

ATME.isStaticObject(objectName)

Description : Fonction permettant de savoir si un objet dont on connaît le nom est statique ou non.

Paramètres :

| | | |
|-------------------|--------|----------------|
| objectName | string | Nom de l'unité |
|-------------------|--------|----------------|

Valeurs de retour :

| | |
|---------|---|
| boolean | True si l'unité est une unité joueur, false sinon |
|---------|---|

ATME.isSurfaceLand(point)

Description : Fonction retournant si un point du terrain est sur terre, hors piste d'atterrissage.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|---------------|---------|--------------------------------------|
| Sans erreur : | boolean | True si c'est sur terre, false sinon |
| Si erreur : | nil | |

ATME.isSurfaceRoad(point)

Description : Fonction retournant si un point du terrain est sur une route.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | True si c'est sur une route, false sinon |
| Si erreur : | nil | |

ATME.isSurfaceRunway(point)

Description : Fonction retournant si un point du terrain est sur une piste d'atterrissage.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True si c'est sur une piste d'atterrissage, false sinon |
| Si erreur : | nil | |

ATME.isSurfaceWater(point)

Description : Fonction retournant si un point du terrain est dans l'eau.

Paramètres :

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
|--------------|-------|---|

Valeurs de retour :

| | | |
|---------------|---------|---------------------------------------|
| Sans erreur : | boolean | True si c'est dans l'eau, false sinon |
| Si erreur : | nil | |

ATME.isUnitControllable(unitName)

Description : Fonction permettant de savoir si une unité dont le nom est passé en paramètre est une unité joueur (active ou non) telle que défini dans la mission.

Paramètres :

| | | |
|-----------------|--------|----------------|
| unitName | string | Nom de l'unité |
|-----------------|--------|----------------|

Valeurs de retour :

| | | |
|--|---------|---|
| | boolean | True si l'unité est une unité joueur, false sinon |
|--|---------|---|

ATME.loopTransmission(file, point, modulation, frequency, power)

Description : Cette fonction déclenche l'émission en boucle d'un fichier son à la fréquence radio choisie. La source de l'émission est située au point passé en paramètre. Le fichier est donc rejoué systématiquement dès que sa lecture se termine. Pour stopper l'émission, il convient d'utiliser la fonction **ATME.stopTransmission**. Le fichier son doit être définie dans la mission.

Paramètres :

| | | |
|-------------------|--------|---|
| file | string | Nom du fichier son |
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
| modulation | string | "AM" ou "FM" |
| frequency | number | Fréquence d'émission en Hz |
| power | number | Puissance de l'émission (en W) |

Valeurs de retour :

| | | |
|---------------|--------|--------------------------------------|
| Sans erreur : | number | Identifiant de la radio transmission |
| Si erreur : | nil | |

Note : une puissance de 5W ou 25W est classique

ATME.rotationH(point, center, angle)

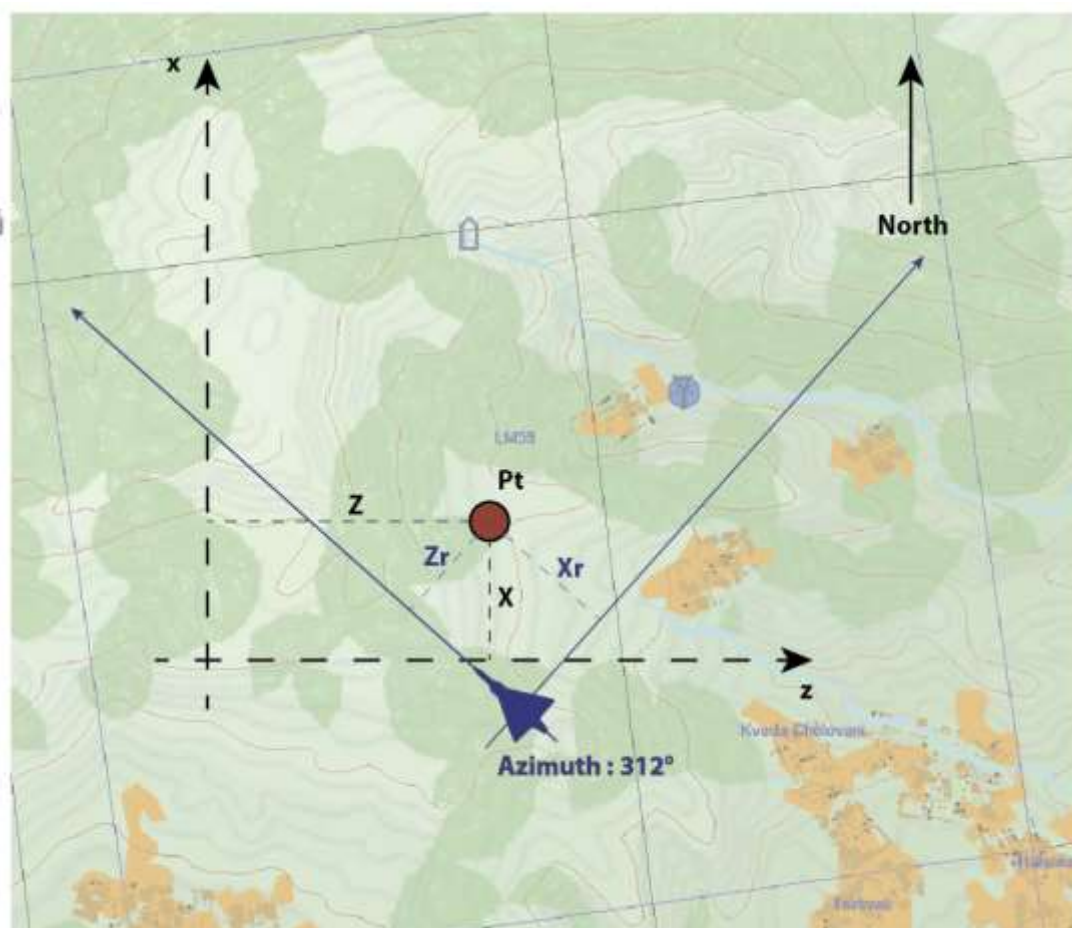
Description : Cette fonction transforme un point donné dans le référentiel (center/angle) en un point 3D dans le référentiel de DCS (x,y,z).

Paramètres :

| | | |
|---------------|--------|---|
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
| center | table | Point 3D en x, y et z, ou point 2D en x,y |
| angle | number | Angle de rotation en degrés |

Valeurs de retour :

| | | |
|---------------|-------|-----------------------|
| Sans erreur : | table | Point 3D en x, y et z |
| Si erreur : | nil | |



Sur le dessin, l'avion a un azimuth de 312° (équivalent à -48°, l'axe X représentant le Nord (Azimuth 0°), et les coordonnées de Pt dans le référentiel de l'avion sont Zr et Xr, la fonction permettra de calculer les coordonnées de Pt(Z,X) dans le référentiel DCS. L'avion sera le centre de la rotation et l'angle de la rotation sera -azimuth (soit 48° pour le cas présenté).

ATME.run(language)

Description : Cette fonction lance l'exécution d'ATME après le chargement de tous les modules nécessaire. Il fixe également la langue. Cette fonction se met directement dans un script dans l'éditeur de mission. Ceci permet de définir des libellés en fonction de la langue choisie. Si les libellés n'existent pas, une erreur d'exécution peut avoir lieu. La définition d'un module multilangue demande une approche particulière.

Après son exécution, il n'est plus possible de charger de nouveaux modules.

Paramètres :

| | | |
|-----------------|--------|--|
| language | string | Langue des messages à afficher : "FR" : en français "EN" : en anglais "GER" : en allemand "RUS" : en russe |
|-----------------|--------|--|

Valeurs de retour : Aucune

ATME.scaireHFFromPoints(pA, pB, pC)

Description : Cette fonction calcule le produit scalaire des vecteurs AB et AC en 2D (x,z au format DCS).

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |
| pC | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|--------|----------------------------|
| Sans erreur : | number | Produit scalaire résultant |
| Si erreur : | nil | |

ATME.sendTransmissionOnce(file, point, modulation, frequency, power)

Description : Cette fonction déclenche l'émission unique d'un fichier son à la fréquence radio choisie. La source de l'émission est située au point passé en paramètre.

Paramètres :

| | | |
|-------------------|--------|---|
| file | string | Nom du fichier son |
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
| modulation | string | "AM" ou "FM" |
| frequency | number | Fréquence d'émission en Hz |
| power | number | Puissance de l'émission (en W) |

Valeurs de retour : Aucune

Note : une puissance de 5W ou 25W est classique

ATME.setDebugLevel(level)

Description : Cette fonction permet de définir le niveau maximal d'affichage des messages de debug émis par la méthode **C_ModuleInfos.output**. Les messages des niveaux supérieurs ne seront pas affichés. Tous les messages sont enregistrés dans dcs.log, quel que soit leur niveau. Cette fonction se met directement dans un script dans l'éditeur de mission lors des tests. Par défaut, le niveau est à 0, ce qui empêche tout affichage de niveau de debug même si le module concerné est en mode debug.

Paramètres :

| | | |
|--------------|--------|--|
| level | number | Niveau maximal d'affichage des messages de debug |
|--------------|--------|--|

Valeurs de retour : Aucune

ATME.setF10groupIDAlphaOrder()

Description : Fonction modifiant le classement des items des menus joueurs. Après l'exécution de cette fonction, les items d'un menu F10 sont classés d'abord en fonction de son numéro de groupe et si les numéros sont identiques, par ordre alphabétique.

Paramètres : Aucun

Valeurs de retour : Aucune

ATME.setF10AlphaOrder()

Description : Fonction modifiant le classement des items des menus joueurs. Après l'exécution de cette fonction, les items d'un menu F10 sont classés uniquement par ordre alphabétique. C'est l'ordre par défaut dans ATME.

Paramètres : Aucun

Valeurs de retour : Aucune

ATME.setFlag(id, value)

Description : Fonction affectant une valeur à un flag. Le flag est identifié par un nombre entier strictement positif.

Paramètres :

| | | |
|--------------|-------------------------|---|
| id | number | Numéro du drapeau |
| value | boolean ou number | Si boolean affecte 1 ou 0, sinon c'est la valeur qui est affectée. Value est transformé en entier si besoin. |

Valeurs de retour : Aucune

ATME.soundForAll(file)

Description : Cette fonction permet de jouer un fichier son pour tous les joueurs. Le fichier son doit exister dans la mission.

Paramètres :

| | | |
|-------------|--------|-----------------------------|
| file | string | Nom du fichier son à jouer. |
|-------------|--------|-----------------------------|

Valeurs de retour : Aucune

ATME.soundForCoalition(coalition, file)

Description : Cette fonction permet de jouer un fichier son pour tous les joueurs d'une coalition. Le fichier son doit exister dans la mission.

Paramètres :

| | | |
|----------------------|--------|--|
| coalitionName | string | Nom de la coalition : "NEUTRAL", "RED" ou "BLUE" |
| file | string | Nom du fichier son à jouer. |

Valeurs de retour : Aucune

ATME.soundForCountry(countryName, file)

Description : Cette fonction permet de jouer un fichier son pour tous les joueurs d'un pays. Le fichier son doit exister dans la mission.

Paramètres :

| | | |
|--------------------|--------|-----------------------------|
| countryName | string | Nom du pays |
| file | string | Nom du fichier son à jouer. |

Valeurs de retour : Aucune

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

ATME.startTransmission(file, point, modulation, frequency, power, interval)

Description : Cette fonction déclenche l'émission à intervalle de temps régulier d'un fichier son à la fréquence radio choisie. La source de l'émission est située au point passé en paramètre. Si l'intervalle de temps est inférieur à la durée de lecture, la lecture sera interrompue pour reprendre au début. Si l'intervalle de temps est supérieur à la durée de lecture, aucun son ne sera émis après la lecture jusqu'au redéclenchement. Pour stopper l'émission, il convient d'utiliser la fonction **ATME.stopTransmission**. Le fichier son doit être définie dans la mission.

Paramètres :

| | | |
|-------------------|--------|---|
| file | string | Nom du fichier son |
| point | table | Point 3D en x, y et z, ou point 2D en x,y |
| modulation | string | "AM" ou "FM" |
| frequency | number | Fréquence d'émission en Hz |
| power | number | Puissance de l'émission (en W) |
| interval | number | Nombre indiquant l'intervalle de réémission en secondes. La valeur 0 indique une émission unique. interval est transformé en entier si besoin. |

Valeurs de retour :

| | | |
|---------------|--------|--------------------------------------|
| Sans erreur : | number | Identifiant de la radio transmission |
| Si erreur : | nil | |

Note : une puissance de 5W ou 25W est classique

ATME.stopTransmission(id)

Description : Cette fonction arrête une transmission radio en boucle ou à intervalle régulier.

Paramètres :

| | | |
|-----------|--------|---|
| id | number | Identifiant spécifique retourné par les fonctions startTransmission / loopTransmission |
|-----------|--------|---|

Valeurs de retour : Aucune

ATME.whichSide(pA, pB, pC)

Description : Cette fonction permet de connaître la position relative (droite ou gauche) du point C par rapport à la droite orientée AB.

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |
| pC | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | number | -1 : C est à gauche de AB (orienté) 1 : C est à droite de AB (orienté) 0 : C est sur AB |
| Si erreur : | nil | |

Fonctions de création d'objets statiques de la table ATME

```
ATME.staticObjects.createCargo(countryName, name, type, position, mass, isDead)
```

Description : Cette fonction permet de créer dynamiquement (spawn) un objet statique Cargo. Son azimuth est déterminé aléatoirement. La position sera un point précis ou bien un tirage aléatoire dans la zone DCS passée en paramètre. Il aura une masse donnée (mass) et sera soit entier (isDead = false) ou détruit (isDead = true).

Paramètres :

| | | |
|--------------------|---------|--|
| countryName | string | Nation de l'objet (cf ci-après) |
| name | string | Nom de l'objet |
| position | table | Point 3D en x, y et z, ou point 2D en x,y. Nom de la zone DCS si tirage aléatoire |
| type | string | Type de la charge (voir ci-après) |
| mass | number | Fréquence d'émission en Hz |
| isDead | boolean | Puissance de l'émission (en W) |

Valeurs de retour : Aucune

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

Liste des types de charge

"BASIC" pour l'original (seul fonctionnant sur la 2.0.4 dernière version également), "TRUNKS SMALL", "TRUNKS LONG", "TETRAPOD", "PIPE SMALL", "PIPE BIG", "OIL TANK", "M117", "SMALL CONTAINER", "F BAR", "FUEL TANK", "BW CONTAINER", "CONTAINER", "BARRELS", "AMMO BOX"

| |
|--|
| ATME.staticObjects.createWhiteContainer(countryName, name, position, heading, isDead) |
|--|

Description : Cette fonction permet de créer dynamiquement (spawn) un objet statique White Container. La position sera un point précis ou bien un tirage aléatoire dans la zone DCS passée en paramètre. Il aura un azimuth précis (heading) et sera soit entier (isDead = false) ou détruit (isDead = true).

Paramètres :

| | | |
|--------------------|---------|--|
| countryName | string | Nation de l'objet (cf ci-après) |
| name | string | Nom de l'objet |
| position | table | Point 3D en x, y et z, ou point 2D en x,y. Nom de la zone DCS si tirage aléatoire |
| heading | number | Azimuth de l'objet en degrés |
| isDead | boolean | Puissance de l'émission (en W) |

Valeurs de retour : Aucune

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

Classes ATME

Les objets ATME ont des méthodes liées à leur classe ou à leur instance. Dans ce qui suit, le nom de la classe suivi de « . » sera mentionné pour les méthodes de classe et le mot `Object` suivi de « : » pour les fonctions liées aux instances de cette classe (ie aux objets créés).

Classe ATME.C_AIUnit

Les instances de cette classe sont créées et supprimées par ATME Core.

ATME.C_AIUnit.exists(name)

Description : Fonction vérifiant si une unité est en vie à un instant T.

Paramètres :

| | | |
|-------------|--------|--------------------|
| name | string | Nom d'une unité AI |
|-------------|--------|--------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True si l'instance C_AIUnit existe, false sinon |
| Si erreur : | nil | |

ATME.C_AIUnit.getByName(name)

Description : Fonction permettant de récupérer, à partir de son nom, une unité AI en vie à un instant T.

Paramètres :

| | | |
|-------------|--------|--------------------|
| name | string | Nom d'une unité AI |
|-------------|--------|--------------------|

Valeurs de retour :

| | | |
|---------------|-------|--------------------------------|
| Sans erreur : | table | Instance de la classe C_AIUnit |
| Si erreur : | nil | |

Object:crossAxisFromLeftToRight(pA, pB)

Description : Cette méthode vérifie le franchissement par l'unité AI de la droite horizontale (droite 2D sur les axes x,z) représentée par les points A et B. Le sens AB est orienté et permet de définir le sens du franchissement, ici de gauche à droite.

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True le franchissement dans le sens souhaité a eu lieu, false sinon |
| Si erreur : | nil | |

Object:crossAxisFromRightToLeft(pA, pB)

Description : Cette méthode vérifie le franchissement par l'unité AI de la droite horizontale (droite 2D sur les axes x,z) représentée par les points A et B. Le sens AB est orienté et permet de définir le sens du franchissement, ici de droite à gauche.

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True le franchissement dans le sens souhaité a eu lieu, false sinon |
| Si erreur : | nil | |

Object:disable()

Description : Cette méthode désactive une unité AI. Pour rappel, désactiver une unité revient à la supprimer du jeu sans destruction visible donc sans événement DCS S_EVENT_DEAD.

Les handlers de modules suivants sont activés :

- onDisableAIUnitHandler
- onDeleteAIUnitHandler avec le paramètre isEnabled = false

Paramètres : Aucun

Valeurs de retour : Aucune

Object:explode(power)

Description : Cette méthode déclenche une explosion d'une force définie sur l'unité AI.

Paramètres :

| | | |
|--------------|--------|-----------------------|
| power | number | Force de l'explosion. |
|--------------|--------|-----------------------|

Valeurs de retour : Aucune

Object:fireFlare(color)

Description : Cette méthode déclenche le tir d'un flare à partir de l'unité AI.

Paramètres :

| | | |
|--------------|--------|---|
| color | string | Couleur du flare "GREEN", "RED", "WHITE" et "YELLOW". La valeur "RANDOM" engendre une couleur aléatoire prise parmi les couleurs existantes. |
|--------------|--------|---|

Valeurs de retour : Aucune

Object:fireIlluminationBomb(power)

Description : Cette méthode déclenche le tir d'une bombe lumineuse à partir de l'unité AI.

Paramètres :

| | | |
|--------------|--------|------------------------------|
| power | number | Puissance de la bombe tirée. |
|--------------|--------|------------------------------|

Valeurs de retour : Aucune

Object:fireSmoke(color)

Description : Cette méthode déclenche une fumée à proximité de l'unité AI.

Paramètres :

| | | |
|--------------|--------|---|
| color | string | Couleur de la fumée "BLUE" , "GREEN", "RED", "WHITE" et "ORANGE". La valeur "RANDOM" engendre une couleur aléatoire prise parmi les couleurs existantes. |
|--------------|--------|---|

Valeurs de retour : Aucune

Object:getAGLAltitude()

Description : Cette méthode permet de récupérer l'altitude en mètres de l'unité AI par rapport au niveau du sol.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------|
| number | altitude en m |
|--------|---------------|

Object: getAzimuth()

Description : Cette méthode permet de récupérer l'azimuth (par rapport au Nord) de l'unité AI. Ceci correspond au cap vrai sans prendre en compte la déviation magnétique.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------------------|
| number | Azimuth en degrés (0-360) |
|--------|---------------------------|

Object: getCallsign()

Description : Cette méthode permet de récupérer le nom de la coalition de l'unité AI. Le callsign est composé de « nom id1-id2 », exemple « enfield 1-1 »

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------|
| string | Nom du callsign |
| string | Id1 du callsign |
| string | Id2 du callsign |

Object: getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_AIUnit ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object:getCoalitionName()

Description : Cette méthode permet de récupérer le nom de la coalition de l'unité AI.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | string | Nom de la coalition de l'unité : "RED", "BLUE" ou "NEUTRAL" |
|---------------|--------|---|

Object:getCountryName()

Description : Cette méthode permet de récupérer le nom du pays de l'unité AI.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|------------------------|
| | string | Nom du pays de l'unité |
|--|--------|------------------------|

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

Object: getDCSUnit()

Description : Cette méthode permet de récupérer l'unité AI telle que gérée par DCS (Objet de classe **Unit**).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---------------------------------------|
| table | Instance de la classe DCS Unit |
|-------|---------------------------------------|

Object: getFuelRatio()

Description : Cette méthode permet de connaître le ratio de fuel restant par rapport au plein.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------|
| number | Ratio restant |
|--------|---------------|

Object: getGroup()

Description : Cette méthode permet de récupérer l'instance **ATME.C_Group** correspondant à l'unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Instance de la classe ATME.C_Group liée à l'unité AI. |
|-------|--|

Object : getGroupsNameOnBoard()

Description : Cette méthode permet de récupérer la liste des noms de groupes d'infanterie embarqués dans l'unité AI. L'unité AI doit être un véhicule habilité, c'est à dire avoir une capacité de transport de troupes (personnel carrier).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Table indexée avec les noms des groupes à bord. Le premier groupe embarqué est à l'index 1. |
|-------|---|

Object : getHSpeed()

Description : Cette méthode permet de récupérer la vitesse horizontale de l'unité AI en m/s (calculée sur les 2 axes x et z).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| number | Vitesse horizontale de l'unité AI en m/s |
|--------|--|

Object : getLife()

Description : Cette méthode permet de récupérer l'information de « vie » d'une unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------|
| number | Vie de l'unité |
|--------|----------------|

Object:getLifeRatio()

Description : Cette méthode permet de récupérer le ratio de vie. Ce ratio se calcule à partir de la vie de l'unité à sa création et son état actuel.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--------------|
| number | Ratio de vie |
|--------|--------------|

Object:getMSLAltitude()

Description : Cette méthode permet de récupérer l'altitude en mètres de l'unité AI par rapport au niveau de la mer.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------------|
| number | Altitude en m de l'unité AI |
|--------|-----------------------------|

Object:getName()

Description : Cette méthode permet de récupérer le nom d'une unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-------------------|
| string | Nom de l'unité AI |
|--------|-------------------|

Object:getNbUnitsOnBoard()

Description : Cette méthode permet de récupérer le nombre d'infanteries à bord.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------------|
| number | Nombre d'infanteries à bord |
|--------|-----------------------------|

Object: getNearestReadyToBoardGroups(radius)

Description : Cette méthode permet de rechercher le groupe d'infanterie prêt à être embarqué le plus proche dans le rayon précisé (radius). Si aucune unité ne répond ne peut être embarquée, nil sera retourné.

Paramètres :

| | | |
|---------------|--------|-------------------------|
| radius | number | Rayon pour la recherche |
|---------------|--------|-------------------------|

Valeurs de retour :

| | | |
|--|-------|---|
| | table | Groupe d'infanterie le plus proche pouvant être embarqué (ATME.C_Group) |
| | nil | Si aucun groupe trouvé |

Object: getPosition()

Description : Cette méthode permet de récupérer la position de l'unité AI (Point 3D).

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|--|
| | table | Point 3D en x, y et z représentant la position de l'unité AI |
|--|-------|--|

Object: getRollAxisVector()

Description : Cette méthode retourne un vecteur correspondant à l'axe longitudinal de l'unité AI, orienté vers l'avant.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|---------------------------------|
| | table | Vecteur de type ATME.C_Vector3D |
|--|-------|---------------------------------|

Object:getSpeed()

Description : Cette méthode permet de récupérer la vitesse de l'unité AI en m/s (calculée sur les 3 axes).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|------------------------------|
| number | Vitesse de l'unité AI en m/s |
|--------|------------------------------|

Object:getSpeedAzimuth()

Description : Cette méthode permet de récupérer la route vraie (cap du vecteur vitesse sur les axes x et z) de l'unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-------------------------|
| number | Route en degrés (0-360) |
|--------|-------------------------|

Object:getTypeName()

Description : Cette méthode permet de récupérer le nom du type d'une unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------------------|
| string | Nom du type de l'unité AI |
|--------|---------------------------|

Object:getVelocityVector()

Description : Cette méthode permet de récupérer le vecteur vitesse de l'unité AI

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Instance de C_Vector3D représentant la vitesse |
|-------|--|

Object: getVSpeed()

Description : Cette méthode permet de récupérer la vitesse verticale de l'unité AI en m/s (prise sur l'axe y).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| number | Vitesse verticale de l'unité AI en m/s |
|--------|--|

Object: inAir()

Description : Cette méthode permet de savoir si l'unité AI est en l'air ou non.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est en l'air. False sinon |
|---------|---|

Object: isAAA()

Description : Cette méthode permet de savoir si l'unité AI est une unité de canon anti-aérien.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est un canon anti-aérien. False sinon |
|---------|---|

Object:isAirDefence()

Description : Cette méthode permet de savoir si l'unité AI est une unité de défense aérienne.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | Si true, l'unité AI est une unité de défense aérienne. False sinon |
|---------|--|

Object:isGroundVehicle()

Description : Cette méthode permet de savoir si l'unité AI est un véhicule terrestre.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | Si true, l'unité AI est un véhicule terrestre. False sinon |
|---------|--|

Object:isHelicopter()

Description : Cette méthode permet de savoir si l'unité AI est un hélicoptère.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est un hélicoptère. False sinon |
|---------|---|

Object:isInfantry()

Description : Cette méthode permet de savoir si l'unité AI est une unité d'infanterie.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|---------|---|
| | boolean | Si true, l'unité AI est une unité d'infanterie. False sinon |
|--|---------|---|

Object:isInDCSZone(zoneName)

Description : Cette méthode permet de savoir si l'unité AI est dans une zone DCS définie dans l'éditeur de mission.

Paramètres : Aucun

| | | |
|-----------------|--------|-----------------|
| zoneName | string | Nom de la zone. |
|-----------------|--------|-----------------|

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | Si true, l'unité AI est dans la zone zoneName . False sinon |
| Si erreur : | nil | |

Object:isInZone2D(reference, radius)

Description : Cette méthode permet de savoir si l'unité AI est dans une zone horizontale définie par un cercle ayant pour centre une référence et un rayon radius.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|--------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon du cercle représentant la zone horizontale en m |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, l'unité AI est dans la zone horizontale. False sinon |
| Si erreur : | nil | |

Object:isInZone3D(reference, radius)

Description : Cette méthode permet de savoir si l'unité AI est dans une zone sphérique définie par une boule ayant pour centre une référence et un rayon radius.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|--------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon de la boule représentant la zone sphérique en m |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, l'unité AI est dans la zone. False sinon |
| Si erreur : | nil | |

Object:isNear(reference, radius, deltaAltitude)

Description : Cette méthode permet de savoir si une référence est proche d'une unité AI. La zone est définie par un cylindre représenté par un cercle horizontal de rayon radius, les deux côtés du cylindre correspondent à une différence d'altitude en prenant l'unité AI pour centre.

Une référence peut être :

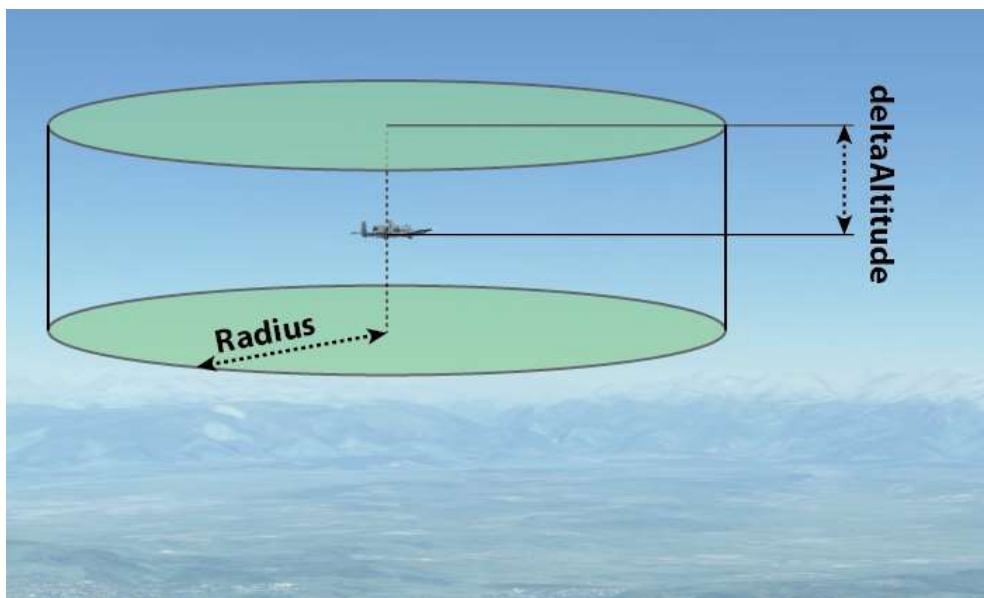
- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|----------------------|--------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon du cercle représentant la zone cylindrique en m |
| deltaAltitude | number | Ecart d'altitude en m |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, la référence est dans la zone cylindrique. False sinon |
| Si erreur : | nil | |



Object:isManPad()

Description : Cette méthode permet de savoir si l'unité AI est une infanterie équipée de missiles anti-aérien.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est une infanterie équipée de missiles anti-aérien. False sinon |
|---------|---|

Object:isPersonnelCarrier()

Description : Cette méthode permet de savoir si l'unité AI est transport de troupes.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | Si true, l'unité AI est un transport de troupes. False sinon |
|---------|--|

Object:isPlane()

Description : Cette méthode permet de savoir si l'unité AI est un avion.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est un avion. False sinon |
|---------|---|

Object:isSAMVehicle()

Description : Cette méthode permet de savoir si l'unité AI est une unité de missiles Sol/Air autoportés.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est une unité de missiles Sol/Air autoportés. False sinon |
|---------|---|

Object:isSAMSiteCommandCenter()

Description : Cette méthode permet de savoir si l'unité AI est une unité de commandement d'un site Sol/Air.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | Si true, l'unité AI est une unité de commandement. False sinon |
|---------|--|

Object:isSAMSiteLauncher()

Description : Cette méthode permet de savoir si l'unité AI est une unité de tir d'un site Sol/Air.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité AI est une unité de tir. False sinon |
|---------|---|

Object:isSAMSiteRadar()

Description : Cette méthode permet de savoir si l'unité AI est une unité radar d'un site Sol/Air.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | Si true, l'unité AI est une unité radar. False sinon |
|---------|--|

Object:load(groupToBoard, radius)

Description : Cette méthode permet d'embarquer un groupe d'infanterie. L'unité AI doit être un véhicule habilité, c'est à dire avoir une capacité de transport de troupes (personnel carrier). Le radius précise le rayon pris en compte pour l'embarquement. Le groupe à embarquer doit se situer dans le rayon d'embarquement de l'unité AI. L'unité AI de transport de troupes doit aussi être posée et avoir une vitesse horizontale inférieure à 1m/s.

Un véhicule de transport de troupes a une capacité d'embarquement maximale en terme d'unités d'infanterie. Il peut embarquer plusieurs groupes d'infanteries tant que cette capacité maximale n'est pas dépassée. La masse totale du transport n'est actuellement pas changée.

L'événement ATME Core "TRANSPORT_END_OF_BOARDING" est émis à la fin de l'embarquement.

Paramètres :

| | | |
|---------------------|--------|--|
| groupToBoard | table | Groupe d'infanterie à embarquer (ATME.C_Group) |
| radius | number | Rayon limite pour l'embarquement |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | False si pas d'erreur, pas d'autres retour. |
| Si erreur : | boolean | True en cas d'erreur, le deuxième retour indique l'erreur |
| | string | Codes d'erreur : |
| | | "LOAD_UNIT_TOO_FAR" |
| | | "LOAD_UNIT_IN_AIR" |
| | | "LOAD_UNIT_BAD_SPEED" |
| | | "LOAD_TOO_MUCH_INFANTRY" |
| | | "LOAD_INFANTRY_GROUP_NOT_READY" |
| | | "LOAD_NO_INFANTRY_GROUP_AVAILABLE" |

Liste des types d'hélicoptères habilités :

"UH-1H" limité à 8 infanteries, "Mi-8MT" limité à 16 infanteries, "SA342M" limité à 2 infanteries, "SA342L" limité à 2 infanteries, "SA342Mistral" limité à 2 infanteries

Liste des types de véhicules terrestres habilités :

"AAV7" limité à 25 infanteries, "M-113" limité à 11 infanteries, "LAV-25" limité à 6 infanteries, "M1126 Stryker ICV" limité à 9 infanteries, "M-2 Bradley" limité à 6 infanteries, "BTR-80" limité à 9 infanteries, "BTR_D" limité à 12 infanteries, "MTLB" limité à 10 infanteries, "BMD-1" limité à 4 infanteries, "BMP-1" limité à 8 infanteries, "BMP-2" limité à 7 infanteries, "BMP-3" limité à 7 infanteries, "UAZ-469" limité à 6 infanteries, "Tigr_233036" limité à 6 infanteries, "GAZ-3307" limité à 16 infanteries, "GAZ-3308" limité à 16 infanteries, "Ural-4320T" limité à 20 infanteries, "Ural-4320-31" limité à 20 infanteries, "M 818" limité à 20 infanteries, "KAMAZ Truck" limité à 20 infanteries.

Object:loopTransmission(file, modulation, frequency, power)

Description : Cette méthode déclenche l'émission en boucle d'un fichier son à la fréquence radio choisie. Le fichier est donc rejoué systématiquement dès que sa lecture se termine. Pour stopper l'émission, il convient d'utiliser la fonction **Object:stopTransmission**. Le fichier son doit être définie dans la mission. La source d'émission correspond à la position de l'unité au moment de l'appel de la fonction. Si une unité déclenche plusieurs appels à cette fonction, le dernier appel remplace les précédents.

Paramètres :

| | | |
|-------------------|--------|--------------------------------|
| file | string | Nom du fichier son |
| modulation | string | "AM" ou "FM" |
| frequency | number | Fréquence d'émission en Hz |
| power | number | Puissance de l'émission (en W) |

Valeurs de retour :

| | | |
|---------------|--------|--|
| Sans erreur : | number | Identifiant utile à Object:stopTransmission |
| Si erreur : | nil | |

Note : une puissance de 5W ou 25W est classique

Object:sendTransmissionOnce(file, modulation, frequency, power)

Description : Cette méthode déclenche l'émission unique d'un fichier son à la fréquence radio choisie. La source d'émission correspond à la position de l'unité au moment de l'appel de la fonction.

Paramètres :

| | | |
|-------------------|--------|--------------------------------|
| file | string | Nom du fichier son |
| modulation | string | "AM" ou "FM" |
| frequency | number | Fréquence d'émission en Hz |
| power | number | Puissance de l'émission (en W) |

Valeurs de retour : Aucune

Object:startTransmission(file, modulation, frequency, power, interval)

Description : Cette méthode déclenche l'émission à intervalle de temps régulier d'un fichier son à la fréquence radio choisie. Si l'intervalle de temps est inférieur à la durée de lecture, la lecture sera interrompue pour reprendre au début. Si l'intervalle de temps est supérieur à la durée de lecture, aucun son ne sera émis après la lecture jusqu'au redéclenchement. Pour stopper l'émission, il convient d'utiliser la fonction **Object:stopTransmission**. Le fichier son doit être définie dans la mission. La source d'émission correspond à la position de l'unité au moment de l'appel de la fonction. Si une unité déclenche plusieurs appels à cette fonction, le dernier appel remplace les précédents.

Paramètres :

| | | |
|-------------------|--------|--------------------------------|
| file | string | Nom du fichier son |
| modulation | string | "AM" ou "FM" |
| frequency | number | Fréquence d'émission en Hz |
| power | number | Puissance de l'émission (en W) |

Valeurs de retour :

| | | |
|---------------|--------|--|
| Sans erreur : | number | Identifiant utile à Object:stopTransmission |
| Si erreur : | nil | |

Note : une puissance de 5W ou 25W est classique

Object:stopAllTransmissions()

Description : Cette méthode arrête une transmission en boucle ou une transmission à intervalle régulier.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:stopTransmission(id)

Description : Cette méthode arrête une transmission en boucle ou une transmission à intervalle régulier. Id correspond à l'identifiant retourné par :

- **Object:startTransmission**
- **Object:loopTransmission**

Paramètres :

| | | |
|-----------|--------|--|
| id | number | Identifiant représentant la transmission à stopper |
|-----------|--------|--|

Valeurs de retour : Aucune

Object:unload(id)

Description : Cette méthode permet de débarquer un groupe d'infanterie présent dans un transport de troupes. Id correspond à l'index dans l'ordre d'embarquement, si besoin, il faudra utiliser **getGroupsNameOnBoard** afin de récupérer la liste indexée des troupes à bord. Id devra correspondre à l'un des index de cette liste de noms.

L'événement ATME Core "TRANSPORT_END_OF_DISEMBARK" est émis à la fin du débarquement.

Paramètres :

| | | |
|-----------|--------|----------------------|
| id | number | Index d'embarquement |
|-----------|--------|----------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | False si pas d'erreur |
| | table | Instance du groupe débarqué |
| | number | Nombre d'unités perdues lors du débarquement |
| Si erreur : | boolean | True en cas d'erreur, le deuxième retour indique l'erreur |
| | string | Codes d'erreur : |
| | | "UNLOAD_UNIT_IN_AIR" |
| | | "UNLOAD_UNIT_BAD_SPEED" |
| | | "UNLOAD_BAD_ID" |

Object:whichSide(reference)

Description : Cette fonction permet de connaître la position relative (droite ou gauche) de la référence par rapport à l'unité AI.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres :

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | number | -1 : la référence est à gauche de l'unité AI 1 : la référence est à droite de l'unité AI 0 : la référence est dans l'axe longitudinal de l'unité AI |
| Si erreur : | nil | |

Classe ATME.C_AirBase

Les instances de cette classe sont créées et supprimées par ATME Core.

Toutes les bases sont référencées qu'elles soient liées à la carte ou qu'elles soient créées dans la mission (FARP ou Airfield)

Object:getName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Airbase ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object:getName()

Description : Cette méthode permet de récupérer le nom d'une base.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------|
| string | Nom de la base |
|--------|----------------|

Classe ATME.C_EventMgr

La classe **C_EventMgr** est la classe de gestion des événements ATME. Les instances de cette classe sont gérées directement par ATME et passées en parameter (**events**) aux handlers de modules (voir classe **C_Module** pour plus d'informations) suivants :

- **onUpdatePlayerHandler(player, events)**
- **onTimerHandler(events)**

Un module utilisateur ne peut créer une instance de cette classe.

Il existe deux type d'événements gérés par cette classe :

- Les événements core : Ils sont générés par ATME Core et **envoyés à tous les modules actifs dans la mission**. Ces événements concernent notamment les entrées/sorties de zones de surveillance, les fins d'embarquement/débarquement et les événements de courses.
- Les événements associés aux triggers utilisateur : Ces événements sont générés par ATME sur la base de l'état individuel des triggers utilisateur et **envoyé au module ayant créé ce trigger utilisateur**. Les autres modules ne recevront pas l'événement.

Un événement n'est envoyé qu'une seule fois à un module. S'il n'est pas traité par le module, il sera donc perdu. Seul les événements issus de trigger utilisateur ayant une durée longue d'activation pourront être générés et envoyés plusieurs fois jusqu'à la fin d'activation.

L'identifiant (**id**) d'un événement est :

- Un nombre si l'événement est un événement Core
- Une chaîne de caractères (string) si l'événement est généré par un trigger utilisateur. Il correspond au nom du trigger utilisateur associé

Object: getCoreEventType(id)

Description : Cette méthode permet de récupérer le type d'un événement Core . **id** est donné par la boucle en utilisant la fonction **pairs** associée.

Chaque événement Core dispose de données (datas) associées variables selon le type de l'événement. Il est donc nécessaire d'utiliser cette fonction avec de lire les données de l'événement.

Paramètres :

| | | |
|-----------|--------|----------------------------|
| id | number | Identifiant de l'événement |
|-----------|--------|----------------------------|

Valeurs de retour :

| | |
|--------|---|
| string | Type de l'événement core permettant sa caractérisation. |
|--------|---|

Types d'événements Core associé aux courses (voir aussi **C Race** pour plus d'informations) :

- **"RACE_PLAYER_TOO_HIGH"** : Généré si le joueur est trop haut en franchissant une porte
- **"RACE_FINAL_TIME"** : Généré lorsqu'un joueur franchi la porte d'arrivée. La course est finie pour ce joueur.
- **"RACE_BEST_FINAL_TIME"** : Généré lorsqu'un joueur réalise le meilleur temps en course
- **"RACE_PLAYER_BEST_FINAL_TIME"** : Généré lorsqu'un joueur réalise son meilleur temps
- **"RACE_BEST_LAP_TIME"** : Généré lorsqu'un joueur réalise le meilleur temps intermédiaire en course à une porte donnée.
- **"RACE_LAP_TIME"** : Généré lorsqu'un joueur franchi une porte avec une mesure de temps intermédiaire
- **"RACE_BEST_TURN_TIME"** : Généré lorsqu'un joueur réalise le meilleur tour en course. Ceci n'est valable que pour les courses sur plusieurs tours.
- **"RACE_PLAYER_BEST_TURN_TIME"** : Généré lorsqu'un joueur réalise son meilleur tour en course. Ceci n'est valable que pour les courses sur plusieurs tours.
- **"RACE_TURN_TIME"** : Généré lorsqu'un joueur franchi la porte de départ après un tour complet. Ceci n'est valable que pour les courses sur plusieurs tours.
- **"RACE_DOOR_MISSED"** : Généré lorsqu'un joueur loupe une porte.
- **"RACE_START"** : Généré lorsqu'un joueur franchit la porte de départ et débute sa course. Pour les courses avec plusieurs tours, cet événement ne sera généré qu'une seule fois, en début de course.

Types d'événements Core associé entrée/sortie de zone de surveillance :

- **"SIGNAL_UNIT_IN_ZONE"** : Généré lorsqu'une unité de transport de troupe entre dans une zone de surveillance.
- **"SIGNAL_UNIT_OUT_OF_ZONE"** : Généré lorsqu'une unité de transport de troupe sort d'une zone de surveillance.

Types d'événements Core associé aux transports de troupes :

- **"TRANSPORT_END_OF_BOARDING"** : Généré lorsqu'une unité de transport de troupe a terminé l'embarquement d'un groupe d'infanterie.
- **"TRANSPORT_END_OF_DISEMBARK"** : Généré lorsqu'une unité de transport de troupe a terminé le débarquement d'un groupe d'infanterie.

Object: `getCoreEventDatas(id)`

Description : Cette méthode permet de récupérer le type d'un événement Core . **id** est donné par la boucle en utilisant la fonction **pairs** associée.

Chaque événement Core dispose de données (datas) associées variables selon le type de l'événement.

Paramètres :

| | | |
|-----------|--------|----------------------------|
| id | number | Identifiant de l'événement |
|-----------|--------|----------------------------|

Valeurs de retour :

| | |
|-------|--|
| table | Datas associées à l'événement Core et dépendantes du type. |
|-------|--|

Ci-dessous, la valeur de retour est supposée être associé à la variable **datas** :

```
local datas = Object:getCoreEventDatas(id)
```

Datas d'événements Core associé aux courses (voir aussi **C_Race** pour plus d'informations) :

- **"RACE_PLAYER_TOO_HIGH"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.penalty** : Cumul de pénalités pour le joueur.
- **"RACE_FINAL_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.penalty** : Cumul de pénalités pour le joueur.
 - **datas.totalPlayTime** : Temps total
- **"RACE_BEST_FINAL_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.totalPlayTime** : Temps total qui devient le meilleur temps de course
- **"RACE_PLAYER_BEST_FINAL_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.totalPlayTime** : Temps total qui devient le meilleur temps du joueur
- **"RACE_BEST_LAP_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.playTime** : Temps du tour.

- **datas.turnNumber** : Numéro du tour.
- **datas.doorNumber** : Numéro de la porte mesurant le temps intermédiaire.
- **"RACE_LAP_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.penalty** : Cumul de pénalités pour le joueur.
 - **datas.playTime** : Temps du tour.
 - **datas.totalPlayTime** : Temps total depuis le début de course
 - **datas.turnNumber** : Numéro du tour.
 - **datas.doorNumber** : Numéro de la porte mesurant le temps intermédiaire.
- **"RACE_BEST_TURN_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.playTime** : Temps du tour qui devient le meilleur temps en course
 - **datas.turnNumber** : Numéro du tour.
- **"RACE_PLAYER_BEST_TURN_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.playTime** : Temps du tour qui devient le meilleur temps du joueur
 - **datas.turnNumber** : Numéro du tour.
- **"RACE_TURN_TIME"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.penalty** : Cumul de pénalités pour le joueur.
 - **datas.playTime** : Temps du tour.
 - **datas.totalPlayTime** : Temps total depuis le début de course
 - **datas.turnNumber** : Numéro du tour.
- **"RACE_DOOR_MISSED"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement
 - **datas.doorNumber** : Numéro de la porte ayant été loupée.
- **"RACE_START"** :
 - **datas.race** : Course à l'origine de l'événement
 - **datas.unit** : Joueur (Instance **C_Player**) ayant causé l'événement

Types d'événements Core associé entrée/sortie de zone de surveillance :

- **"SIGNAL_UNIT_IN_ZONE"** :
 - **datas.unit** : Unité (Instance **C_Player** ou **C_AIUnit**) ayant causé l'événement
 - **datas.linkedGroup** : Groupe (Instance **C_Group**) en surveillance
- **"SIGNAL_UNIT_OUT_OF_ZONE"** :
 - **datas.unit** : Unité (Instance **C_Player** ou **C_AIUnit**) ayant causé l'événement
 - **datas.linkedGroup** : Groupe (Instance **C_Group**) en surveillance

Types d'événements Core associé embarquement/débarquement de groupes d'infanterie :

- " **TRANSPORT_END_OF_BOARDING**" :
 - **datas.unit** : Unité (Instance **C_Player** ou **C_AIUnit**) ayant causé l'événement
 - **datas.onBoardGroupName** : Nom du groupe venant d'embarquer
 - **datas.nbUnitsOnBoard** : Nombre total d'unités d'infanterie à bord
 - **datas.nbDeadUnits** : Nombre d'unités perdues au cours de l'embarquement
- " **TRANSPORT_END_OF_DISEMBARK**" :
 - **datas.unit** : Unité (Instance **C_Player** ou **C_AIUnit**) ayant causé l'événement
 - **datas.index** : Index d'embarquement du groupe venant de débarquer
 - **datas.linkedGroup** : Groupe (Instance **C_Group**) venant de débarquer
 - **datas.nbUnitsOnBoard** : Nombre total d'unités d'infanterie restant à bord
 - **datas.nbDeadUnits** : Nombre d'unités perdues au cours du débarquement

Object:isCoreEvent(id)

Description : Cette méthode permet de savoir si un événement est un événement Core. **id** est donné par la boucle en utilisant la fonction **pairs** associée.

Paramètres :

| | | |
|-----------|--------|----------------------------|
| id | number | Identifiant de l'événement |
|-----------|--------|----------------------------|

Valeurs de retour :

| | |
|---------|--|
| boolean | True si l'événement est un événement Core, false sinon |
|---------|--|

Object:isUserTriggerEvent(id)

Description : Cette méthode permet de savoir si un événement est un événement engendré par un trigger utilisateur. **id** est donné par la boucle en utilisant la fonction **pairs** associée et correspond au nom du trigger utilisateur.

Paramètres :

| | | |
|-----------|--------|---|
| id | string | Identifiant de l'événement, ici le nom du trigger utilisateur |
|-----------|--------|---|

Valeurs de retour :

| | |
|---------|---|
| boolean | True si l'événement est un événement issu d'un trigger utilisateur, false sinon |
|---------|---|

Object:isUserTriggerEventToggle(id)

Description : Cette méthode permet de savoir si un événement engendré par un trigger utilisateur vient de changer d'état c'est-à-dire que le trigger associé vient d'être activé. **id** est donné par la boucle en utilisant la fonction **pairs** associée et correspond au nom du trigger utilisateur.

Cette fonction est utile si un trigger utilisateur a une durée longue. Dans ce cas, **isUserTriggerEventToggle** ne retournera vrai qu'au premier appel de handler faisant suite à son activation, et ce contrairement à **isUserTriggerEvent**.

Paramètres :

| | | |
|-----------|--------|---|
| id | string | Identifiant de l'événement, ici le nom du trigger utilisateur |
|-----------|--------|---|

Valeurs de retour :

| | |
|---------|---|
| boolean | True si l'événement issu d'un trigger utilisateur vient d'être activé. Il sera à false si il n'y a pas eu de changement d'état, false sinon |
|---------|---|

Object: pairs()

Description : Cette méthode redéfinit la méthode pairs permettant de balayer les entrées dans une table. Elle s'utilise dans une boucle visant à traiter successivement tous les événements reçus à un instant T.

Paramètres : Aucun

Valeurs de retour : Associé à pairs (lua)

Classe ATME.C_Flare

ATME.C_Flare(colorName, point)

Description : Cette fonction crée un flare sur un point de la carte.

Paramètres :

| | | |
|------------------|--------|--|
| colorName | string | Couleur du flare: "GREEN", "RED", "WHITE" et "YELLOW". La valeur "RANDOM" engendre une couleur aléatoire à prendre parmi celles existantes. |
| point | table | Localisation sur la carte, point 3D en x, y et z. |

Valeurs de retour :

| | | |
|-------------|-------|-------------------------------|
| Sans erreur | table | Nouvelle instance de C_Flare. |
| Si erreur | nil | |

Object: fire(duration)

Description : Cette méthode déclenche des tirs aléatoires de flares pendant une période donnée en secondes.

Paramètres :

| | | |
|-----------------|--------|-------------------|
| duration | number | Durée en secondes |
|-----------------|--------|-------------------|

Valeurs de retour : Aucune

Object:fireOnce()

Description : Cette méthode déclenche un tir de flare.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Flare ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object:getColor()

Description : Cette méthode retourne la couleur du flare. Celle-ci peut varier dans le temps si random.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| string | Couleur du flare : "GREEN", "RED", "WHITE" ou "YELLOW". |
|--------|---|

Object:getDuration()

Description : Cette méthode retourne la durée en secondes d'un tir (fonction fire).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-------------------|
| number | Durée en secondes |
|--------|-------------------|

Object:getPoint()

Description : Cette méthode retourne le point d'où le flare est tiré.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Point 3D en x, y et z, ou point 2D en x,y |
|-------|---|

Object:getTimeStart()

Description : Cette méthode retourne l'heure de début du tir exprimé en secondes depuis le début de la mission.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| number | Nombre de secondes depuis le début de la mission |
|--------|--|

Object:stop()

Description : Cette méthode stoppe un tir avec une durée (fonction **fire**).

Paramètres : Aucun

Valeurs de retour : Aucune

Classe ATME. C_Group

Les instances de cette classe sont créées et supprimées par ATME Core.

ATME.C_Group.exists(name)

Description : Fonction vérifiant si un groupe est en vie à un instant T (au moins une unité en vie dans le groupe)

Paramètres :

| | | |
|-------------|--------|---------------|
| name | string | Nom du groupe |
|-------------|--------|---------------|

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | True si l'instance C_Group existe, false sinon |
| Si erreur : | nil | |

ATME.C_Group.getByName(name)

Description : Fonction permettant de récupérer, à partir de son nom, un groupe en vie à un instant T.

Paramètres :

| | | |
|-------------|--------|---------------|
| name | string | Nom du groupe |
|-------------|--------|---------------|

Valeurs de retour :

| | | |
|---------------|-------|-------------------------------|
| Sans erreur : | table | Instance de la classe C_Group |
| Si erreur : | nil | |

ATME.C_Group.getGroupsInDCSZoneForAll(zoneName, allGroup)

Description : Cette méthode permet de retourner tous les groupes présents dans une zone DCS définie dans l'éditeur de mission. Il est possible de définir si seuls les groupes entièrement et/ou partiellement dans la zone sont pris en compte.

Paramètres :

| | | |
|-----------------|---------|--|
| zoneName | string | Nom de la zone. |
| allGroup | boolean | Si true, les groupes doivent être complètement dans la zone. Si false, il suffit qu'une unité des groupes soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Table avec les groupes (ATME.C_Group) répondant aux critères. La table est référencée par les noms de groupes. La table sera vide si aucun groupe ne répond aux critères. |
| Si erreur : | nil | |

ATME.C_Group.getGroupsInDCSZoneForCoalition(coalitionName, zoneName, allGroup)

Description : Cette méthode permet de retourner tous les groupes d'une coalition particulière présents dans une zone DCS définie dans l'éditeur de mission. Il est possible de définir si seuls les groupes entièrement et/ou partiellement dans la zone sont pris en compte.

Paramètres :

| | | |
|----------------------|---------|--|
| coalitionName | string | Nom de la coalition : "RED", "BLUE" ou "NEUTRAL" |
| zoneName | string | Nom de la zone. |
| allGroup | boolean | Si true, les groupes doivent être complètement dans la zone. Si false, il suffit qu'une unité des groupes soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Table avec les groupes (ATME.C_Group) répondant aux critères. La table est référencée par les noms de groupes. La table sera vide si aucun groupe ne répond aux critères. |
| Si erreur : | nil | |

ATME.C_Group.getGroupsInZone2DForAll(reference, radius, allGroup)

Description : Cette méthode permet de retourner tous les groupes présents dans une zone horizontale (2D) définie par une référence et le rayon radius. Il est possible de définir si seuls les groupes entièrement et/ou partiellement dans la zone sont pris en compte.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres :

| | | |
|------------------|---------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon pour la zone 2D en m. |
| allGroup | boolean | Si true, les groupes doivent être complètement dans la zone. Si false, il suffit qu'une unité des groupes soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Table avec les groupes (ATME.C_Group) répondant aux critères. La table est référencée par les noms de groupes. La table sera vide si aucun groupe ne répond aux critères. |
| Si erreur : | nil | |

| | |
|--|---------------------------------------|
| ATME.C_Group.getGroupsInZone2DForCoalition(coalitionName, | reference, allGroup) |
|--|---------------------------------------|

Description : Cette méthode permet de retourner tous les groupes d'une coalition particulière présents dans une zone horizontale (2D) définie par une référence et le rayon radius. dans une zone DCS définie dans l'éditeur de mission. Il est possible de définir si seuls les groupes entièrement et/ou partiellement dans la zone sont pris en compte.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres :

| | | |
|----------------------|---------|--|
| coalitionName | string | Nom de la coalition : "RED", "BLUE" ou "NEUTRAL" |
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon pour la zone 2D en m. |
| allGroup | boolean | Si true, les groupes doivent être complètement dans la zone. Si false, il suffit qu'une unité des groupes soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|-------|---|
| Sans erreur : | table | Table avec les groupes (ATME.C_Group) répondant aux critères. La table est référencée par les noms de groupes. La table sera vide si aucun groupe ne répond aux critères. |
| Si erreur : | nil | |

ATME.C_Group.getReadyToBoardGroupsForAll()

Description : Cette méthode permet de retourner tous les groupes prêts à être embarqués.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Table avec les groupes (ATME.C_Group) répondant aux critères. La table est référencée par les noms de groupes. La table sera vide si aucun groupe ne répond aux critères. |
| Si erreur : | nil | |

ATME.C_Group.getReadyToBoardGroupsForCoalition(coalitionName, zoneName, allGroup)

Description : Cette méthode permet de retourner tous les groupes prêts à être embarqués pour une coalition donnée.

Paramètres :

| | | |
|----------------------|--------|--|
| coalitionName | string | Nom de la coalition : "RED", "BLUE" ou "NEUTRAL" |
|----------------------|--------|--|

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Table avec les groupes (ATME.C_Group) répondant aux critères. La table est référencée par les noms de groupes. La table sera vide si aucun groupe ne répond aux critères. |
| Si erreur : | nil | |

Object:changeReadyToBoard(value)

Description : Cette méthode permet de rendre un groupe d'infanterie prêt à être embarqué dans une unité de transport de troupes. C'est une condition préalable à l'embarquement. Le groupe est stoppé s'il était en mouvement.

Paramètres :

| | | |
|--------------|---------|---|
| value | boolean | Si true, le groupe est prêt à être embarqué, false pour qu'il reprenne ses activités. |
|--------------|---------|---|

Valeurs de retour : Aucune

Object:activate()

Description : Cette méthode active un groupe complet. L'activation d'un groupe est nécessaire pour les groupes définis avec une activation retardée. Lors de l'activation, le groupe devient visible. Le groupe est déjà créé dans ce cas.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:disable()

Description : Cette méthode désactive un groupe complet. Pour rappel, désactiver un groupe revient à le supprimer du jeu sans destruction visible donc sans événement DCS S_EVENT_DEAD pour les unités associées.

Les handlers de modules suivants sont activés :

- onDisableGroupHandler
- onDeleteGroupHandler avec le paramètre isEnabled = false

Paramètres : Aucun

Valeurs de retour : Aucune

Object:freeze(on)

Description : Cette méthode permet de figer ou non un groupe composé d'unités AI. Dès que figé, le groupe ne fera plus aucune action et sera non contrôlé par DCS.

Paramètres :

| | | |
|-----------|---------|---|
| on | boolean | Si true, le groupe est mis hors contrôle (figé), si false est à nouveau contrôlé par DCS. |
|-----------|---------|---|

Valeurs de retour : Aucune

Object:getBarycentre()

Description : Cette méthode permet de retourner le barycentre du groupe

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|-----------------------|
| | table | Point 3D en x, y et z |
|--|-------|-----------------------|

Object:getCategoryName()

Description : Cette méthode permet de retourner le nom de la catégorie d'un groupe.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|---|
| | string | Nom de la catégorie : "AIRPLANE", "HELICOPTER", "GROUND" ou "SHIP" |
|--|--------|---|

Object:getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Group ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------------|
| | string | Nom de la classe ATME |
|--|--------|-----------------------|

Object:getCoalitionName()

Description : Cette méthode permet de récupérer le nom de la coalition du groupe.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|--|
| Sans erreur : | string | Nom de la coalition du groupe : "RED", "BLUE" ou "NEUTRAL" |
|---------------|--------|--|

Object: getDCSGroup()

Description : Cette méthode permet de récupérer le groupe tel que géré par DCS (Objet de classe **Group**).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Instance de la classe DCS Group |
|---------------|-------|--|

Object: getFirstUnit()

Description : Cette méthode permet de récupérer la première unité en vie d'un groupe.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|---|
| | table | Instance de type C_AIUnit ou C_Player |
|--|-------|---|

Object: getLastUnit()

Description : Cette méthode permet de récupérer la dernière unité en vie d'un groupe.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|---|
| | table | Instance de type C_AIUnit ou C_Player |
|--|-------|---|

Object: getMaxDistance(reference)

Description : Cette méthode permet de calculer la distance maximale d'un groupe par rapport à une référence. C'est en fait la distance de l'unité en vie la plus éloignée. Le calcul se fait sur les axes x, y et z.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position

- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | |
|--------|---------------|
| number | Distance en m |
|--------|---------------|

Object: getMaxHDistance(reference)

Description : Cette méthode permet calculer la distance horizontale maximale d'un groupe par rapport à une référence. C'est en fait la distance de l'unité en vie la plus éloignée. Le calcul se fait sur les axes x et z.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | |
|--------|---------------|
| number | Distance en m |
|--------|---------------|

Object: getMinDistance(reference)

Description : Cette méthode permet calculer la distance minimale d'un groupe par rapport à une référence. C'est en fait la distance de l'unité en vie la plus proche. Le calcul se fait sur les axes x, y et z.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | |
|--------|---------------|
| number | Distance en m |
|--------|---------------|

Object: getMinHDistance(reference)

Description : Cette méthode permet calculer la distance horizontale minimale d'un groupe par rapport à une référence. C'est en fait la distance de l'unité en vie la plus proche. Le calcul se fait sur les axes x et z.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | |
|--------|---------------|
| number | Distance en m |
|--------|---------------|

Object: getName()

Description : Cette méthode permet de récupérer le nom du groupe.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---------------|
| Sans erreur : | string | Nom du groupe |
| Si erreur : | nil | |

Object:getNbUnits()

Description : Cette méthode permet de récupérer le nombre d'unités en vie du groupe.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------------------------------|
| number | Nombre d'unités en vie dans le groupe |
|--------|---------------------------------------|

Object:getNextUnit()

Description : Cette méthode permet de récupérer l'unité suivante en vie. Pour être utilisée, il faut avoir au préalable initialisé un balayage du groupe avec `getFirstUnit` ou `getLastUnit`.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Instance de type C_AIUnit ou C_Player |
| nil | Si aucune unité trouvée ou si aucun balayage du groupe n'est en cours |

Object:getPickupZone()

Description : Cette méthode permet de récupérer la zone DCS définissant une zone d'embarquement pour un groupe d'infanterie.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| string | Nom de la zone DCS |
| nil | Si aucune zone n'est affectée au groupe. |

Object: getPreviousUnit()

Description : Cette méthode permet de récupérer l'unité précédente en vie. Pour être utilisée, il faut avoir au préalable initialisé un balayage du groupe avec getFirstUnit ou getLastUnit.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Instance de type C_AIUnit ou C_Player |
| nil | Si aucune unité trouvée ou si aucun balayage du groupe n'est en cours |

Object: getUnitByName(name)

Description : Cette méthode permet de récupérer l'unité du groupe ayant pour nom **name**. L'unité doit être en vie

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Instance de type C_AIUnit ou C_Player |
| nil | Si aucune unité trouvée |

Object: getUnits()

Description : Cette méthode permet de récupérer une table avec toutes les unités en vie du groupe au moment de l'appel

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Table contenant toutes les instances de type C_AIUnit ou C_Player |
|-------|---|

Object:hasPickupZone()

Description : Cette méthode permet de savoir si une zone DCS définissant une zone d'embarquement a été affectée à un groupe d'infanterie.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le groupe d'infanterie a une zone DCS d'embarquement associée. False sinon. |
|---------|---|

Object:isActivated()

Description : Cette méthode retourne l'état d'activation d'un groupe défini comme étant à activation retardée dans l'éditeur de mission.

Paramètres : Aucun

Valeurs de retour : Aucune

| | |
|---------|--|
| boolean | True si le groupe est activé, false sinon. |
|---------|--|

Note importante :

Attention, la fonction **disable** détruit le groupe. Aussi, dans ce cas, le groupe n'existe plus pour ATME après l'utilisation de cette fonction.

Par ailleurs, actuellement, le test utilise une fonction DCS qui se base sur l'unité.

Object:isBoardingStarted()

Description : Cette méthode permet de savoir si un groupe d'infanterie est en cours d'embarquement dans une unité de transport de troupes.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le groupe est en cours d'embarquement, false sinon. |
|---------|---|

Object:isInDCSZone(zoneName, allGroup)

Description : Cette méthode permet de savoir si le groupe est dans une zone DCS définie dans l'éditeur de mission. Il est possible de savoir si tout le groupe ou seulement une partie du groupe répond à cette contrainte.

Paramètres : Aucun

| | | |
|-----------------|---------|--|
| zoneName | string | Nom de la zone. |
| allGroup | boolean | Si true, vérifie si tout le groupe est dans la zone. Si false, il suffit qu'une unité du groupe soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, en fonction de allGroup , tout ou partie du groupe est dans la zone zoneName . False sinon |
| Si erreur : | nil | |

Object:isInZone2D(reference, radius, allGroup)

Description : Cette méthode permet de savoir si le groupe est dans une zone horizontale définie par un cercle ayant pour centre une référence et un rayon radius. Il est possible de savoir si tout le groupe ou seulement une partie du groupe répond à cette contrainte.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|---------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon du cercle représentant la zone horizontale en m |
| allGroup | boolean | Si true, vérifie si tout le groupe est dans la zone 2D. Si false, il suffit qu'une unité du groupe soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | Si true, en fonction de allGroup , tout ou partie du groupe est dans la zone horizontale. False sinon |
| Si erreur : | nil | |

Object:isInZone3D(reference, radius, allGroup)

Description : Cette méthode permet de savoir si le groupe est dans une zone sphérique définie par une boule ayant pour centre une référence et un rayon radius. Il est possible de savoir si tout le groupe ou seulement une partie du groupe répond à cette contrainte.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|---------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon de la boule représentant la zone sphérique en m |
| allGroup | boolean | Si true, vérifie si tout le groupe est dans la zone 3D. Si false, il suffit qu'une unité du groupe soit dans la zone. |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, en fonction de allGroup , tout ou partie du groupe est dans la zone 3D. False sinon |
| Si erreur : | nil | |

Object:isNear(reference, radius, deltaAltitude, allGroup)

Description : Cette méthode permet de savoir si la référence est proche de tout ou partie d'un groupe. La zone est définie par un cylindre représenté par un cercle horizontal de rayon radius, les deux côtés du cylindre correspondent à une différence d'altitude.

Une référence peut être :

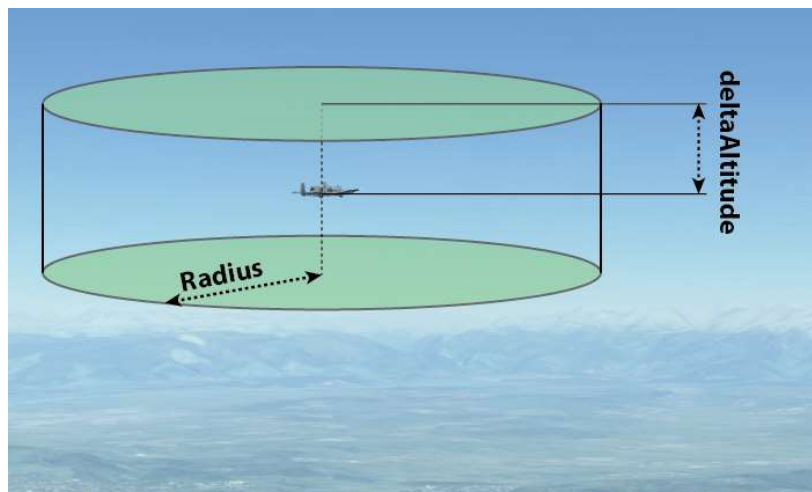
- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|----------------------|---------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon du cercle représentant la zone cylindrique en m |
| deltaAltitude | number | Ecart d'altitude en m |
| allGroup | boolean | Si true, vérifie si la référence est dans la zone de chaque unité du groupe. Si false, il suffit que la référence soit proche d'au moins une unité du groupe. |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, en fonction de allGroup , la référence est de la zone de chaque unité du groupe ou d'au moins une unité. False sinon |
| Si erreur : | nil | |



Object:isOnlyInfantry()

Description : Cette méthode permet de savoir si un groupe ne contient que des unités d'infanterie à un instant T.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le groupe ne contient que des unités d'infanterie, false sinon. |
|---------|---|

Object:isOnlyAI()

Description : Cette méthode permet de savoir si un groupe ne contient que des unités AI à un instant T.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le groupe ne contient que des unités AI, false sinon. |
|---------|---|

Object:isReadyToBoard()

Description : Cette méthode permet de savoir si un groupe d'infanterie est prêt à être embarqué dans une unité de transport de troupes. Quand un groupe est prêt à être embarqué, il peut ensuite être embarqué. C'est une condition préalable à l'embarquement. Le groupe est également stoppé s'il était en mouvement.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | True si le groupe est prêt, false sinon. |
|---------|--|

Object:isSignalSet()

Description : Cette méthode permet de savoir le groupe est en surveillance de survol.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le groupe d'infanterie est en surveillance de survol. False sinon. |
|---------|---|

Object:isStopped()

Description : Cette méthode permet de savoir est en mouvement ou stoppé. Son état dépend de l'utilisation des fonctions **move** ou **stop**.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | True si le groupe a été stoppé, false sinon. |
|---------|--|

Object:move()

Description : Cette méthode permet de mettre en route un groupe composé d'unités AI après un maintien en position

Paramètres : Aucun

Valeurs de retour : Aucune

Object:resetPickupZone()

Description : Cette méthode supprimer le rattachement d'un groupe d'infanterie à une zone DCS d'embarquement. Le groupe ne change pas d'état prêt à être embarqué.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:resetSignal()

Description : Cette méthode supprimer la surveillance de la zone. Dès lors, le groupe n'émettra plus de signaux en cas de survol par une unité de transport de troupes.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:setPickupZone(zoneName, random)

Description : Cette méthode permet de rattacher un groupe d'infanterie à une zone d'embarquement. Cette zone est définie par une zone DCS de nom **zoneName** définie dans l'éditeur de mission. Le paramètre **random** permet de mettre de l'aléatoire dans l'arrêt du groupe et son passage à l'état prêt à embarquer.

Dès que le groupe a été affecté à une zone d'embarquement, s'il entre dans la zone un tirage du sort aura lieu toutes les secondes pour le rendre prêt à être embarqué. Si le groupe ressort alors qu'il n'est toujours pas prêt à être embarqué, il sera immédiatement mis dans cet état. Tout est géré automatiquement. Plusieurs groupes peuvent avoir une même zone d'embarquement.

Une zone d'embarquement influe sur les moyens de signalement d'un groupe d'infanterie prêt à embarquer, si ils ont été définis. Dans ce cas, un seul groupe de la zone se signalera. On sera reportera à la fonction **setSignal** pour connaître le fonctionnement d'un signalement.

Paramètres :

| | | |
|-----------------|--------|---|
| zoneName | string | Nom de la zone DCS |
| random | number | Valeur décimale entre 0 et 1 : 1 : le groupe passe immédiatement en prêt à embarquer dès qu'il entre dans la zone. 0 : le groupe ne passe jamais en prêt à embarquer. Dans ce cas, c'est lorsqu'il quittera la zone qu'il sera forcé à l'état prêt à embarquer. |

Valeurs de retour : Aucune

Object : setRoute(...)

Description : Cette méthode permet d'associer tout ou partie des waypoints d'un autre groupe défini dans la mission. Les paramètres de cette fonction sont variables et listés ci-après. Il est ainsi possible de faire changer de route un groupe quelconque, voire également de demander à un groupe de reparcourir ses propres waypoints.

Paramètres cas 1 : Aucun paramètre. Les waypoints repris sont ceux du groupe tels que défini dans l'éditeur de mission. Si le groupe est une copie d'un groupe initial (spawn), les waypoints seront ceux du groupe initial.

Paramètres cas 2 : Deux paramètres. Les waypoints repris sont ceux du groupe dont le nom est passé en paramètre. Ce nom doit exister dans l'éditeur de mission. Le deuxième paramètre permet de fixer à partir d'où les waypoints sont repris :

- Un numéro de waypoint : Les waypoints antérieurs seront ignorés.
- Un nom de waypoint : S'il existe, les waypoints antérieurs seront ignorés. S'il n'existe pas, seul le dernier waypoint sera repris. Une référence : Tous les waypoints définis après le waypoint le plus proche de la référence seront repris. Une référence peut être :
 - Un point 2D ou 3D,
 - Une unité AI représentée par sa position
 - Une unité joueur représentée par sa position
 - Un objet statique représenté par sa position

| | | |
|------------------|--------|--|
| groupName | string | Nom du groupe défini dans l'éditeur de mission. |
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |

Valeurs de retour : Aucune

Object:setSignal(radius, signalType)

Description : Cette méthode permet à un groupe de passer en surveillance de zone. Dès qu'une unité de transport de troupes aérienne (actuellement uniquement des hélicoptères) débute le survol de la zone dont le centre est la première unité en vie du groupe et le rayon, celui passé en paramètre de cette fonction, cette dernière déclenche un signal. Si le groupe est rattaché à une zone d'embarquement où sont présents d'autres groupes eux même en surveillance, un seul d'entre eux enclenchera le signal. En cas de destruction du groupe, un autre groupe (s'il existe) prendra immédiatement le relais. Dans ce cas précis, et si le signal est une fumée, il peut y avoir plusieurs fumées actives simultanément, celle des groupes détruits et celle du groupe toujours actif.

Le groupe émet les signaux à intervalle régulier tant qu'une unité de transport de troupes survole la zone :

- Si le signal est un flare, le tir est régulier avec toutefois un facteur aléatoire de temps
- Si le signal est une fumée, la fumée sera régénérée régulièrement avec un intervalle de temps aléatoire.

La fonction **resetSignal** arrêtera la surveillance et l'émission de signaux.

Paramètres :

| | | |
|-------------------|--------|---|
| radius | number | Rayon de la zone de surveillance |
| signalType | string | Chaine de caractère ayant un format spécifique (cf ci-dessous). |

Valeurs de retour : Aucune

Valeurs pour le paramètre signalType :

- La chaine est composée de deux mots : "**TypeSignal Color**".
- "**TypeSignal**" correspond à "**SIGNAL_FLARE**" pour un flare et "**SIGNAL_SMOKE**" pour une fumée
- "**Color**" doit correspondre à une des couleurs possibles, y compris "**RANDOM**". On se reportera à **ATME.C_Flare** ou **ATME.C_Smoke** pour plus d'informations.
- Exemples :
 - "**SIGNAL_SMOKE BLUE**" déclenchera une fumée bleue
 - "**SIGNAL_FLARE RANDOM**" un tir régulier de flare de couleur aléatoire.

Object:stop()

Description : Cette méthode permet un maintien en position du groupe composé d'unités AI.

Paramètres : Aucun

Valeurs de retour : Aucune

Classe ATME. C_GroupSpawnDatas

ATME.C_GroupSpawnDatas.duplicateFromMissionDatas(groupName, newGroupName)

Description : Cette fonction crée une instance permettant de récupérer les données nécessaires à la création d'un nouveau groupe par rapport à un groupe défini dans l'éditeur de mission.

Paramètres :

| | | |
|---------------------|--------|---|
| groupName | string | Nom du groupe tel que défini dans la mission. |
| newGroupName | string | Nom du nouveau groupe. |

Valeurs de retour :

| | | |
|---------------|-------|---|
| Sans erreur : | table | Nouvelle instance de données avec le nouveau nom de groupe. |
| Si erreur | nil | |

Object:getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_GroupSpawnDatas ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------------|
| | string | Nom de la classe ATME |
|--|--------|-----------------------|

Object : getName()

Description : Cette méthode permet de récupérer le nom du groupe de données qui correspond au nouveau nom.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---------------|
| Sans erreur : | string | Nom du groupe |
| Si erreur : | nil | |

Object : spawn(...)

Description : Cette méthode crée un nouveau groupe d'unités à partir des données présentes dans l'instance courante. Le nom du nouveau groupe ne doit pas être utilisé au moment de la création. Il en va de même pour les unités du nouveau groupe dont le nom a été modifié (cf ci-après). Cette fonction peut avoir plusieurs paramètres variables, décrits ci-après. Ces paramètres concernent la reprise des waypoints du groupe initial ou d'un autre groupe ainsi que la position du nouveau groupe. Cette fonction déclenche les handlers suivant dans les modules utilisateurs présents :

- Appel de **onCreateGroupHandler**
- Pour chaque unité du nouveau groupe : Appel de **onCreateAIUnitHandler**

Les nouvelles unités portent les noms suivant :

- Nom d'origine précédé de "**SpawnXXXXXX**", **XXXXXX** étant un nombre unique précédé si besoin par des 0.

Paramètres cas 1 : Aucun paramètre. Les waypoints repris sont ceux du groupe tels que défini dans l'éditeur de mission. La position du nouveau groupe et celle du groupe initial.

Paramètres cas 2 : un paramètre. La position est précisée. Les waypoints repris sont ceux du groupe initial, utilisé lors du **duplicateFromMissionDatas**.

| | | |
|--------------|-------|---|
| point | table | Point 3D en x, y et z définissant la position du groupe |
|--------------|-------|---|

Paramètres cas 3 : Deux paramètres. La position doit être précisée. Les waypoints repris sont ceux du groupe dont le nom est passé en paramètre (ce peut être le même nom que le groupe initial). Tous les waypoints sont repris.

| | | |
|--------------------|--------|---|
| point | table | Point 3D en x, y et z définissant la position du groupe |
| groupNameWP | string | Nom du groupe source des waypoints à reprendre |

Paramètres cas 4 : Trois paramètres. La position doit être précisée. Les waypoints repris sont ceux du groupe dont le nom est passé en paramètre. Ce nom doit exister dans l'éditeur de mission. Le troisième paramètre permet de fixer à partir d'où les waypoints sont repris :

- Un numéro de waypoint : Les waypoints antérieurs seront ignorés.
- Un nom de waypoint : S'il existe, les waypoints antérieurs seront ignorés. S'il n'existe pas, seul le dernier waypoint sera repris. Une référence : Tous les waypoints définis après le waypoint le plus proche de la référence seront repris. Une référence peut être :
 - Un point 2D ou 3D,
 - Une unité AI représentée par sa position
 - Une unité joueur représentée par sa position
 - Un objet statique représenté par sa position

| | | |
|--------------------|--------|--|
| point | table | Point 3D en x, y et z définissant la position du groupe |
| groupNameWP | string | Nom du groupe source des waypoints à reprendre |
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |

Valeurs de retour : Aucune

Classe ATME.C_IndexList

Cette classe permet de liste indexée pouvant contenir des valeurs nil. L'index sera forcément numérique. En cas de suppression d'une entrée, l'index des entrées suivantes ne change pas.

ATME.C_IndexList()

Description : Cette fonction crée une nouvelle liste.

Paramètres : Aucun

Valeurs de retour :

| | | |
|-------------|-------|-----------------------------------|
| Sans erreur | table | Nouvelle instance de C_IndexList. |
| Si erreur | nil | |

Object:add(item)

Description : Cette méthode permet d'ajouter un élément à la liste.

Paramètres :

| | | |
|-------------|-------|------------------------------|
| item | table | Élément à ajouter à la liste |
|-------------|-------|------------------------------|

Valeurs de retour : Aucune

Object:get(index)

Description : Cette méthode permet de récupérer l'élément positionné à l'index passé en paramètre

Paramètres :

| | | |
|--------------|--------|-----------------------------|
| index | number | Index de l'élément souhaité |
|--------------|--------|-----------------------------|

Valeurs de retour :

| | | |
|--|-------|-----------------------------|
| | table | Élément à la position index |
|--|-------|-----------------------------|

Object:getName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Line2D ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------------|
| | string | Nom de la classe ATME |
|--|--------|-----------------------|

Object:getCount()

Description : Cette méthode permet de récupérer le nombre d'éléments dans la liste.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|---------------------------------|
| | number | Nombre d'éléments dans la liste |
|--|--------|---------------------------------|

Object:remove(index)

Description : Cette méthode permet de supprimer l'élément présent à l'index passé en paramètre.

Paramètres :

| | | |
|--------------|--------|--------------------------------|
| index | number | Index de l'élément à supprimer |
|--------------|--------|--------------------------------|

Valeurs de retour : Aucune

Object:removeAll()

Description : Cette méthode permet de supprimer tous les éléments de la liste.

Paramètres : Aucun

Valeurs de retour : Aucune

Object: pairs()

Description : Cette méthode redéfinit la méthode pairs permettant de balayer les entrées dans une liste. Elle s'utilise dans une boucle.

Paramètres : Aucun

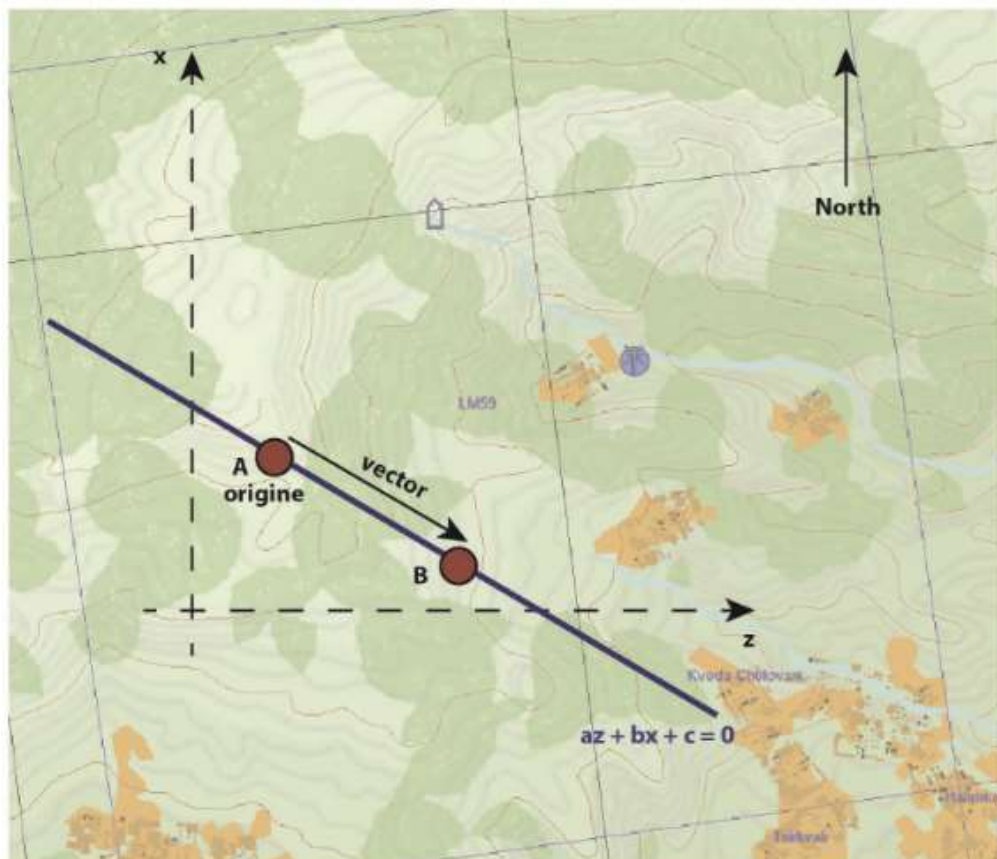
Valeurs de retour : Associé à pairs (lua)

Classe ATME.C_Line2D

Cette classe permet de gérer des droites sur le plan horizontal de DCS défini par les axes x et z.

Le format est : $az + bx + c = 0$.

Une droite dispose également d'un point d'origine, nommé origine ci-dessous.



ATME.C_Line2D(...)

Description : Cette fonction crée une droite sur le plan horizontal, sur les axes x et z.

Paramètres cas 1 : Crée la copie d'une autre droite

| | | |
|-------------|-------|----------------------|
| line | table | Instance de C_Line2D |
|-------------|-------|----------------------|

Paramètres cas 2 : Crée une droite à partir de deux points A et B, en se basant sur les coordonnées x et z. Le point A est l'origine de la droite.

| | | |
|-----------|-------|---|
| pA | table | Point A - Point 3D en x, y et z, ou point 2D en x,y |
|-----------|-------|---|

| | | |
|-----------|-------|---|
| pB | table | Point B - Point 3D en x, y et z, ou point 2D en x,y |
|-----------|-------|---|

Paramètres cas 3 : Crée une droite à partir d'un vecteur et d'un point qui sera l'origine, en se basant sur les coordonnées x et z.

| | | |
|----------------|-------|---|
| vector | table | Instance de C_Vector3D |
| origine | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|-------------|-------|--------------------------------|
| Sans erreur | table | Nouvelle instance de C_Line2D. |
| Si erreur | nil | |

Object:get()

Description : Cette méthode permet de récupérer les coordonnées a, b et c d'une droite horizontale sur les axes x et z.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | a |
| number | b |
| number | c |

Object:getA()

Description : Cette méthode permet de récupérer a.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | a |
|--------|---|

Object:getB()

Description : Cette méthode permet de récupérer b.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | b |
|--------|---|

Object:getC()

Description : Cette méthode permet de récupérer c.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | c |
|--------|---|

Object:getName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Line2D ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object:getOrigine()

Description : Cette méthode permet de récupérer le point d'origine de la droite horizontale

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Point 3D en x, y et z représentant l'origine |
|-------|--|

Object: getPerpendicular(offset)

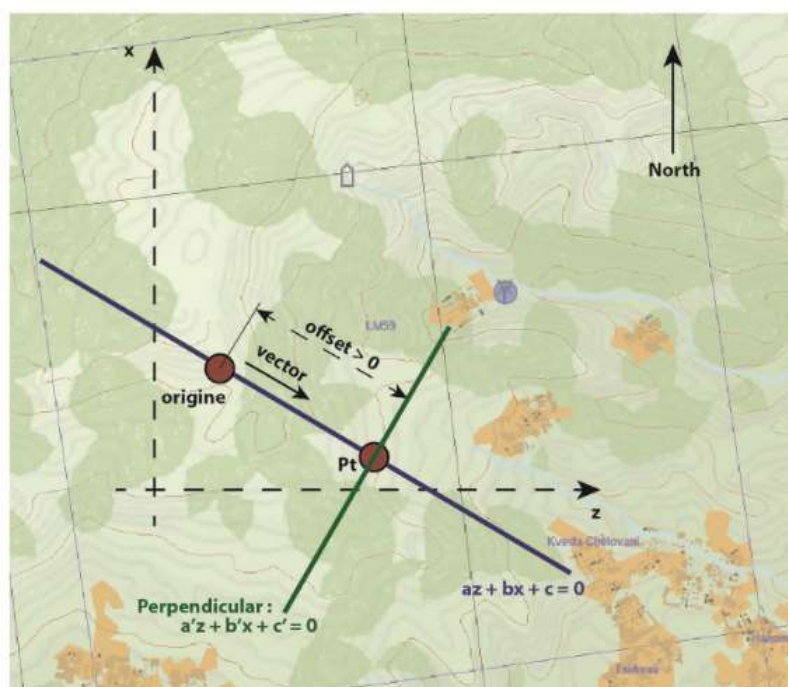
Description : Cette méthode permet de récupérer la droite perpendiculaire passant au point distant de offset par rapport à l'origine. Le point d'origine de cette nouvelle droite sera le point d'intersection (Pt sur le schéma). Si offset est négatif, le point sera à l'opposé de l'exemple ci-dessous pour une même valeur donnée.

Paramètres :

| | | |
|---------------|--------|---------------------------------------|
| offset | number | Offset par rapport au point d'origine |
|---------------|--------|---------------------------------------|

Valeurs de retour :

| | |
|-------|-------------------|
| table | Instance C_Line2D |
|-------|-------------------|



Object: getPointFromOrigine(offset)

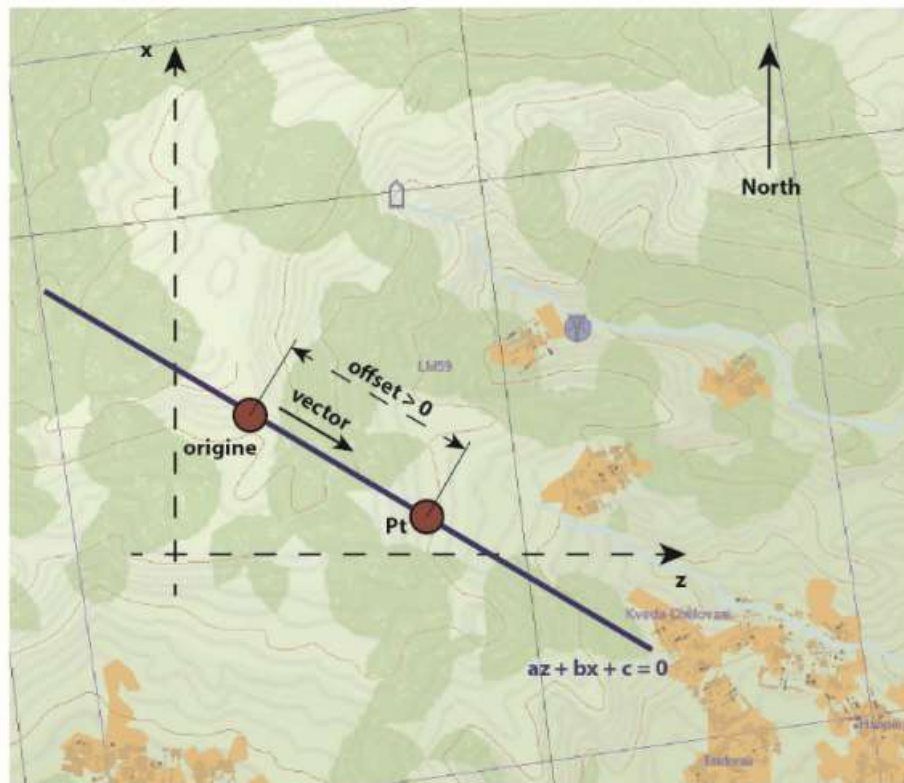
Description : Cette méthode permet de récupérer le point sur la droite distant de offset par rapport à l'origine. Attention, offset peut être positif ou négatif en fonction de la demi-droite concernée par rapport à l'origine. Le signe dépend du vecteur directeur de la droite ; dans le même sens, offset sera positif.

Paramètres :

| | | |
|---------------|--------|---------------------------------------|
| offset | number | Offset par rapport au point d'origine |
|---------------|--------|---------------------------------------|

Valeurs de retour :

| | |
|-------|-----------------------|
| table | Point 3D en x, y et z |
|-------|-----------------------|



Object:getX()

Description : Cette méthode permet de calculer x avec z connu (si possible).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | Coordonnée calculée en x ou nil si impossible |
|--------|---|

Object:getZ()

Description : Cette méthode permet de calculer z avec x connu (si possible).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | Coordonnée calculée en z ou nil si impossible |
|--------|---|

Object:isNSAxis()

Description : Cette méthode permet de savoir si la droite est parallèle à l'axe NS (coordonnée x pour DCS)

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | True si la droite est parallèle, false sinon |
|---------|--|

Object:isWEAxis()

Description : Cette méthode permet de savoir si la droite est parallèle à l'axe WE (coordonnée z pour DCS)

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | True si la droite est parallèle, false sinon |
|---------|--|

Classe ATME.C_MenuF10

Cette classe gère les menus joueurs. Elle s'occupe de l'affichage ou non des items de menu et de leur classement à l'affichage. Le classement à l'affichage est déterminé par les fonctions `ATME.setF10groupIDAlphaOrder()` et `ATME.setF10AlphaOrder()`. Par défaut, il s'agit d'un classement alphabétique.

ATME.C_MenuF10(player, label, parent)

Description : Cette fonction crée un nouveau menu accessible depuis F10 et l'associe à un joueur. Le menu, affiché avec le libellé **label**, est attaché à un menu **parent** ; il s'agit du menu de base (root) si parent vaut **nil**.

Paramètres :

| | | |
|---------------|--------|---|
| player | table | Instance de C_Player |
| label | string | Libellé affiché du menu. |
| parent | table | Instance de C_F10Menu, parent du menu à créer |

Valeurs de retour : Aucune

Object:append(groupId, menuLabel, radioHandler, argsRadioHandler)

Description : Cette méthode ajoute une entrée dans le menu F10 souhaité. Le **groupId** permet d'identifier les entrées dans l'ensemble des menus. Il agit sur le classement à l'affichage si l'option a été activée par la fonction `ATME.setF10groupIDAlphaOrder()`. Cette valeur est également utile à la suppression d'une ou plusieurs entrées. Le **groupId** peut être le même pour plusieurs entrées.

Le menu ne sera affiché que s'il existe au moins une entrée, y compris dans ses sous-menus.

Paramètres :

| | | |
|-------------------------|----------------|---|
| groupId | number | Nombre entier permettant d'identifier une ou plusieurs entrées de menu F10. |
| menuLabel | string | Texte affiché au niveau du menu F10 |
| radioHandler | | Handler de la fonction appelée lors du choix du menu radio |
| argsRadioHandler | table autre | Paramètres passés au handler du menu F10 (table ou autre type de variable) |

Valeurs de retour : Aucune

Object : remove(groupId)

Description : Cette méthode supprime toutes les entrées de commandes radio correspondantes au **groupId** indiqué. La suppression impacte également les sous menus associés.

Paramètres :

| | | |
|----------------|--------|--|
| groupId | number | Nombre entier permettant d'identifier une ou plusieurs entrées de menus F10. |
|----------------|--------|--|

Valeurs de retour : Aucune

Object : removeAll()

Description : Cette méthode supprime toutes les entrées du menu F10 et de ses sous-menus.

Paramètres : Aucun

Valeurs de retour : Aucune

Classe ATME.C_Module

ATME.C_Module(name, handlers, debugOn)

Description : Cette fonction déclare et crée un nouveau module utilisateur. Ce module doit être créé avant que la fonction **ATME.run** soit appelée dans l'éditeur de mission. Le nom du module doit être unique.

Les **handlers** défini dans le nouveau module seront mémorisé par ATME Core et appelés lorsqu'il sera nécessaire.

debugOn permet d'indiquer si le module est ou non en mode debug.

Paramètres :

| | | |
|-----------------|---------|--|
| name | string | Nom du module |
| handlers | table | Liste des handlers définis dans le nouveau module |
| debugOn | boolean | True si le module est en mode debug, false sinon.. |

Valeurs de retour :

| | | |
|-------------|-------|--------------------------------|
| Sans erreur | table | Nouvelle instance de C_Module. |
| Si erreur | nil | |

Handlers liés aux joueurs :

- **onCreatePlayerHandler(player)** : Appelé par ATME Core lors de la création d'un nouveau joueur. Le paramètre **player** correspond à l'instance **C_Player** du nouveau joueur
- **onDeletePlayerHandler(player)** : Appelé par ATME Core lors de la destruction (ou de la désactivation) d'un joueur. Le paramètre **player** correspond à l'instance **C_Player** du joueur.
- **onUpdatePlayerHandler(player, events)** : Appelé par ATME Core toutes les secondes pour tous les joueurs actifs. Le paramètre **player** correspond à l'instance **C_Player** du joueur. **events** correspond à la liste des événements actifs au moment de l'appel (voir classe **C_EventMgr**).
- **onTakeoffPlayerHandler(player)** : Appelé par ATME Core lorsque le joueur décolle. Le paramètre **player** correspond à l'instance **C_Player** du joueur.
- **onLandingPlayerHandler(player)** : Appelé par ATME Core lorsque le joueur atterrit. Le paramètre **player** correspond à l'instance **C_Player** du joueur.
- **onStartEnginePlayerHandler(player)** : Appelé par ATME Core lorsque le joueur démarre ses moteurs. Le paramètre **player** correspond à l'instance **C_Player** du joueur.
- **onStopEnginePlayerHandler(player)** : Appelé par ATME Core lorsque le joueur arrête ses moteurs. Le paramètre **player** correspond à l'instance **C_Player** du joueur.

Handlers liés aux unités AI :

- **onCreateAIUnitHandler(AIUnit)** : Appelé par ATME Core lors de la création d'une nouvelle unité AI. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de la nouvelle unité AI.
- **onDeleteAIUnitHandler(AIUnit, isEnabled)** : Appelé par ATME Core lors de la destruction (ou de la désactivation) d'une unité AI. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de l'unité AI. **isEnabled** est à true si l'unité AI est active, false si elle est désactivée.
- **onDisableAIUnitHandler(AIUnit)** : Appelé par ATME Core lors de la désactivation d'une unité AI. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de l'unité AI. Le handler **onDeleteAIUnitHandler** sera ensuite appelé avec le paramètre **isEnabled** à false.
- **onTakeoffAIUnitHandler(AIUnit)** : Appelé par ATME Core lorsque l'unité AI décolle. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de l'unité AI.
- **onLandingAIUnitHandler(AIUnit)** : Appelé par ATME Core lorsque l'unité AI atterrit. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de l'unité AI.
- **onStartEngineAIUnitHandler(AIUnit)** : Appelé par ATME Core lorsque l'unité AI démarre ses moteurs. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de l'unité AI.
- **onStopEngineAIUnitHandler(AIUnit)** : Appelé par ATME Core lorsque l'unité AI arrête ses moteurs. Le paramètre **AIUnit** correspond à l'instance **C_AIUnit** de l'unité AI.

Handlers liés aux groupes :

- **onCreateGroupHandler(group)** : Appelé par ATME Core lors de la création d'un nouveau groupe. Le paramètre **group** correspond à l'instance **C_Group** du nouveau joueur.
- **onDeleteGroupHandler(group, isEnabled)** : Appelé par ATME Core lors de la destruction (ou de la désactivation) d'un groupe. Le paramètre **group** correspond à l'instance **C_Group** du joueur. **isEnabled** est à true si le groupe est actif, false s'il est désactivé.
- **onDisableGroupHandler(group)** : Appelé par ATME Core lors de la désactivation d'un groupe. Le paramètre **group** correspond à l'instance **C_Group** du groupe. Le handler **onDeleteGroupHandler** sera ensuite appelé avec le paramètre **isEnabled** à false.

Handlers liés aux objets statiques :

- **onCreateStaticObjectHandler(object)** : Appelé par ATME Core lors de la création d'un nouvel objet statique. Le paramètre **object** correspond à l'instance **C_StaticObject** du nouvel objet statique.
- **onDeleteStaticObjectHandler(object)** : Appelé par ATME Core lors de la destruction d'un objet statique. Le paramètre **object** correspond à l'instance **C_StaticObject** de l'objet statique.

Handlers généraux :

- **onTimerHandler(events)** : Appelé par ATME Core une fois par seconde. **events** correspond à la liste des événements actifs au moment de l'appel (voir classe **C_EventMgr**).
- **onModuleStartHandler(initOk)** : Appelé deux fois par ATME Core lors du démarrage d'ATME, dans la fonction **ATME.run**. Le paramètre **initOk** prend les deux valeurs successives suivantes :
 - Au premier appel, **initOk** vaut false. Les unités AI et joueurs n'ont pas été initialisées à ce stade.
 - Au second appel, **initOk** vaut true. Les unités AI et joueurs sont initialisées.

Variables globales et extension des instances d'objets de type C_AIUnit, C_Player, C_Group ou C_StaticObject :

Lors de la création d'un module, une entrée spécifique au module est créée dans la table **ATME.modules**. Le nom de cette entrée est **ATME.modules["nom du module"]**. Cette entrée peut être utilisée librement par le concepteur du module pour y déclarer des données ou fonctions utiles à d'autres modules ou devant être accessible à partir de l'éditeur de mission.

De plus, les classes **C_AIUnit**, **C_Player**, **C_Group** et **C_StaticObject** disposent également d'un nouveau champ dédié au module, **modules["nom du module"]**.

Object:error(text)

Description : Cette fonction génère un message d'erreur qui sera affiché dans une fenêtre et bloquera le script. On notera que tous les messages d'erreur sont enregistrés dans dcs.log et préfixés par "[ATME-ModuleName]".

Paramètres :

| | | |
|-------------|--------|--------------------------------------|
| text | string | Texte du message d'erreur à afficher |
|-------------|--------|--------------------------------------|

Valeurs de retour : Aucune

Object:getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Module ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------------|
| | string | Nom de la classe ATME |
|--|--------|-----------------------|

Object:isDebugEnabled()

Description : Cette méthode permet retourner si un module est ou non en mode debug.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|---------|--|
| | boolean | True si le module est en mode debug. False sinon |
|--|---------|--|

Object:output(text, level)

Description : Cette fonction permet d'afficher un message de debug à niveau défini et de l'enregistrer dans dcs.log. Tous les messages de debug sont préfixés par "[ATME-ModuleName]". La **ATME.setDebugLevel** permet de définir le niveau maximal des messages affichés. Cette fonction peut être utilisée directement par un script dans l'éditeur de mission.

Le message de debug n'est affiché que si le mode debug a été activé pour le module concerné par la méthode **C_Module.setDebug**

Paramètres :

| | | |
|--------------|--------|---|
| text | string | Texte du message de debug |
| level | number | Niveau du message de debug pour affichage |

Valeurs de retour : Aucune

Object:createAbsoluteUserTrigger(name, condition)

Description : Cette méthode crée un nouveau trigger utilisateur basé sur une condition de temps lié au temps de la mission. Un trigger utilisateur est identifié par son nom et toujours lié au module appelant cette fonction afin d'éviter les doublons. Le trigger sera supprimé automatiquement après sa désactivation.

Paramètres :

| | | |
|------------------|--------|---|
| name | string | Nom du trigger utilisateur |
| condition | string | Définition des conditions temporelles d'activation du trigger |

Valeurs de retour :

| | | |
|---------------|-------|---------------------------|
| Sans erreur : | table | Instance de C_UserTrigger |
| Si erreur : | nil | |

Explications sur le paramètre **condition** :

condition est une chaîne définissant le déclenchement et l'arrêt d'un trigger en secondes.

Le format est strict et suit la règle "(+)XXX" ou "(+)XXX (+)YYY"

- "(+)XXX" indique un trigger qui ne se déclenchera qu'une seule fois au temps **XXX**, relatif au début de la mission en l'absence du +, relatif à la création du trigger si + est indiqué.
- "(+)XXX (+)YYY" indique un trigger qui se déclenchera dans une période définie par les bornes **XXX** et **YYY**. Pour **XXX**, relatif au début de la mission en l'absence du +, relatif à la

création du trigger si + est indiqué. Pour **YYY**, relatif au début de la mission en l'absence du +, relatif à XXX en présence du + (soit **XXX+YYY** secondes)

"XXX" et "YYY" se représentent sous les formes :

- sous la forme d'un entier positif représentant le nombre de secondes,
- sous la forme **HH:MM:SS** représentant les heures, minutes et secondes,

Exemples :

- **"10"** le trigger se déclenchera une fois, 10 secondes après le début de la mission
- **"+4200"** le trigger se déclenchera 4200 secondes après sa création.
- **"10 20"** le trigger se déclenchera sur la période allant de 10 secondes à 20 secondes après le début de la mission
- **"00:10:00 +20"** le trigger se déclenchera sur la période allant de 10 minutes à 10 minutes et 20 secondes après le début de la mission
- **"600 +20"** même cas que le précédent écrit différemment.
- **"600 +00:20:00"** le trigger se déclenchera sur la période allant de 10 minutes à 30 minutes après le début de la mission
- **"+00:00:10 20"** le trigger se déclenchera 10 secondes après sa création et se terminera 20 secondes après le début de la mission (attention à la cohérence)
- **"+00:10:00 +1805"** le trigger se déclenchera 10 minutes après sa création et terminera à 10 minutes + 1805 secondes.

Object:createFlagRelativeUserTrigger(name, flagNumber, condition)

Description : Cette méthode crée un nouveau trigger utilisateur basé sur une condition de temps liée à l'activation d'un flag. Un trigger utilisateur est identifié par son nom et toujours lié au module appelant cette fonction afin d'éviter les doublons. Le trigger utilisateur sera supprimé automatiquement après sa désactivation.

Paramètres :

| | | |
|-------------------|--------|---|
| name | string | Nom du trigger utilisateur |
| flagNumber | number | Numéro du flag, qui activé, déclenchera le décompte temporel |
| condition | string | Définition des conditions temporelles d'activation du trigger |

Valeurs de retour :

| | | |
|---------------|-------|---------------------------|
| Sans erreur : | table | Instance de C_UserTrigger |
| Si erreur : | nil | |

Explications sur le paramètre **condition** :

condition est une chaîne définissant le déclenchement et l'arrêt d'un trigger en secondes à partir de l'activation d'un flag.

Le format est strict et suit la règle **"(+)**XXX**"** ou **"(+)**XXX** (+)**YYY**"**

- **"(+)**XXX**"** indique un trigger qui ne se déclenchera qu'une seule fois au temps **XXX**. Le **+** n'a pas de signification car dans tous les cas, le début est relatif au déclenchement du flag.
- **"(+)**XXX** (+)**YYY**"** indique un trigger qui se déclenchera dans une période définie par les bornes **XXX** et **YYY**. Pour **XXX**, le **+** n'a pas de signification car dans tous les cas, le début est relatif au déclenchement du flag. Pour **YYY**, relatif au début de la mission en l'absence du **+**, relatif à **XXX** en présence du **+** (soit **XXX+YYY** secondes)

"XXX" et **"YYY"** se représentent sous les formes :

- sous la forme d'un entier positif représentant le nombre de secondes,
- sous la forme **HH:MM:SS** représentant les heures, minutes et secondes,

Exemples :

- **"10"** le trigger se déclenchera une fois, 10 secondes après l'activation du flag
- **"**+10**"** même cas que le précédent écrit différemment, le **+** n'a pas de signification ici
- **"4200"** même cas que le précédent écrit différemment
- **"10 20"** le trigger se déclenchera sur la période allant de 10 secondes après l'activation du flag à 20 secondes après le début de la mission. Attention au moment où le flag est activé. La période peut avoir une fin précédant son début
- **"**+10 20**"** même cas que le précédent écrit différemment, le **+** n'a pas de signification ici.
- **"00:10:00 **+20**"** le trigger se déclenchera sur la période allant de 10 minutes à 10 minutes et 20 secondes après l'activation du flag
- **"**+600 +20**"** même cas que le précédent écrit différemment. , le **+** n'a pas de signification ici
- **"600 **+00:20:00**"** le trigger se déclenchera sur la période allant de 10 minutes à 30 minutes après l'activation du flag
- **"**+00:10:00 +1805**"** même cas que le précédent écrit différemment

Object:getUserTriggerByName(name)

Description : Cette méthode retourne un trigger utilisateur programmé et/ou activé.

Paramètres :

| | | |
|-------------|--------|----------------------------|
| name | string | Nom du trigger utilisateur |
|-------------|--------|----------------------------|

Valeurs de retour :

| | | |
|---------------|-------|---------------------------|
| Sans erreur : | table | Instance de C_UserTrigger |
| Si erreur : | nil | |

Object:removeUserTrigger(name)

Description : Cette méthode supprime un trigger utilisateur programmé et/ou activé. S'il est activé, il est immédiatement inactif.

Paramètres :

| | | |
|-------------|--------|----------------------------|
| name | string | Nom du trigger utilisateur |
|-------------|--------|----------------------------|

Valeurs de retour : Aucune

Object:userTriggerExists(name)

Description : Cette méthode permet de savoir si un trigger utilisateur existe, c'est-à-dire, qu'il est programmé et/ou activé.

Paramètres :

| | | |
|-------------|--------|----------------------------|
| name | string | Nom du trigger utilisateur |
|-------------|--------|----------------------------|

Valeurs de retour :

| | | |
|---------------|---------|---------------------------------------|
| Sans erreur : | boolean | True si un trigger utilisateur existe |
| Si erreur : | nil | |

Classe ATME. C_Player

Les instances de cette classe sont créées et supprimées par ATME Core.

ATME.C_Player.exists(name)

Description : Fonction vérifiant si l'unité d'un joueur est en vie à un instant T.

Paramètres :

| | | |
|-------------|--------|------------------------|
| name | string | Nom d'une unité joueur |
|-------------|--------|------------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True si l'instance C_Player existe, false sinon |
| Si erreur : | nil | |

ATME.C_Player.getByName(name)

Description : Fonction permettant de récupérer, à partir de son nom, l'unité d'un joueur en vie à un instant T.

Paramètres :

| | | |
|-------------|--------|------------------------|
| name | string | Nom d'une unité joueur |
|-------------|--------|------------------------|

Valeurs de retour :

| | | |
|---------------|-------|--------------------------------|
| Sans erreur : | table | Instance de la classe C_Player |
| Si erreur : | nil | |

Object:crossAxisFromLeftToRight(pA, pB)

Description : Cette méthode vérifie le franchissement par l'unité du joueur de la droite horizontale (droite 2D sur les axes x,z) représentée par les points A et B. Le sens AB est orienté et permet de définir le sens du franchissement, ici de gauche à droite.

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True le franchissement dans le sens souhaité a eu lieu, false sinon |
| Si erreur : | nil | |

Object:crossAxisFromRightToLeft(pA, pB)

Description : Cette méthode vérifie le franchissement par l'unité du joueur de la droite horizontale (droite 2D sur les axes x,z) représentée par les points A et B. Le sens AB est orienté et permet de définir le sens du franchissement, ici de droite à gauche.

Paramètres :

| | | |
|-----------|-------|---|
| pA | table | Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point 3D en x, y et z, ou point 2D en x,y |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True le franchissement dans le sens souhaité a eu lieu, false sinon |
| Si erreur : | nil | |

Object:display(text, duration)

Description : Cette méthode permet d'afficher un message destiné à un joueur particulier. Le message sera affiché pendant la durée indiquée.

Paramètres :

| | | |
|-----------------|--------|--|
| text | string | Texte du message à afficher. |
| duration | number | Durée en secondes de l'affichage du message. |

Valeurs de retour : Aucune

Object:explode(power)

Description Cette méthode déclenche une explosion de force définie sur l'unité d'un joueur.

Paramètres :

| | | |
|--------------|--------|----------------------|
| power | number | Force de l'explosion |
|--------------|--------|----------------------|

Valeurs de retour : Aucune

Object:getAGLAltitude()

Description : Cette méthode permet de récupérer l'altitude en mètres de l'unité d'un joueur par rapport au sol.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---------------|
| Sans erreur : | number | Altitude en m |
|---------------|--------|---------------|

Object: getAzimuth()

Description : Cette méthode permet de récupérer l'azimuth (par rapport au Nord) de l'unité d'un joueur. Ceci correspond au cap vrai sans prendre en compte la variation magnétique.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---------------------------|
| Sans erreur : | number | Azimuth en degrés (0-360) |
|---------------|--------|---------------------------|

Object: getCallsign()

Description : Cette méthode permet de récupérer le nom de la coalition de l'unité AI. Le callsign est composé de « nom id1-id2 », exemple « enfield 1-1 »

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------|
| string | Nom du callsign |
| string | Id1 du callsign |
| string | Id2 du callsign |

Object: getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Player ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object: getCoalitionName()

Description : Cette méthode permet de récupérer le nom de la coalition de l'unité d'un joueur.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | string | Nom de la coalition de l'unité du joueur : "RED", "BLUE" ou "NEUTRAL" |
|---------------|--------|---|

Object: getCountryName()

Description : Cette méthode permet de récupérer le nom du pays de l'unité d'un joueur.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|------------------------------------|
| Sans erreur : | string | Nom du pays de l'unité d'un joueur |
|---------------|--------|------------------------------------|

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

Object:getDCSUnit()

Description : Cette méthode permet de récupérer l'unité d'un joueur telle que gérée par DCS (Objet de classe **Unit**).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|-------|---------------------------------------|
| Sans erreur : | table | Instance de la classe DCS Unit |
|---------------|-------|---------------------------------------|

Object:getF10MenuRoot()

Description : Cette méthode permet de retourner le menu racine du joueur (menu F10 de base).

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|---|
| | table | Menu root de type ATME.C_F10Menu |
|--|-------|---|

Object:getFuelRatio()

Description : Cette méthode permet de connaître le ratio de fuel restant par rapport au plein.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|---------------|
| | number | Ratio restant |
|--|--------|---------------|

Object:addGroup()

Description : Cette méthode permet de récupérer l'instance C_Group correspondant à l'unité d'un joueur.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Instance de la classe C_Group liée à l'unité d'un joueur. |
|---------------|-------|--|

Object:getGroupsNameOnBoard()

Description : Cette méthode permet de récupérer la liste des noms de groupes d'infanterie embarqués dans l'unité du joueur. L'unité du joueur doit être un véhicule habilité, c'est à dire avoir une capacité de transport de troupes (personnel carrier).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Table indexée avec les noms des groupes à bord. Le premier groupe embarqué est à l'index 1. |
|-------|---|

Object:getHSpeed()

Description : Cette méthode permet de récupérer la vitesse horizontale de l'unité d'un joueur en m/s (calculée sur les 2 axes x et z).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | number | Vitesse horizontale de l'unité d'un joueur en m/s |
|---------------|--------|---|

Object: getLife()

Description : Cette méthode permet de récupérer l'information de « vie » d'une unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------|
| number | Vie de l'unité |
|--------|----------------|

Object: getLifeRatio()

Description : Cette méthode permet de récupérer le ratio de vie. Ce ratio se calcule à partir de la vie de l'unité à sa création et son état actuel.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--------------|
| number | Ratio de vie |
|--------|--------------|

Object: getMSLAltitude()

Description : Cette méthode permet récupérer l'altitude en mètres de l'unité d'un joueur par rapport au niveau de la mer.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---------------|
| Sans erreur : | number | Altitude en m |
|---------------|--------|---------------|

Object : getName()

Description : Cette méthode permet de récupérer le nom d'une unité d'un joueur.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|----------------------------|
| Sans erreur : | string | Nom de l'unité d'un joueur |
|---------------|--------|----------------------------|

Object : getNbUnitsOnBoard()

Description : Cette méthode permet de récupérer le nombre d'infanteries à bord.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------------------|
| | number | Nombre d'infanteries à bord |
|--|--------|-----------------------------|

Object : getNearestReadyToBoardGroups(radius)

Description : Cette méthode permet de rechercher le groupe d'infanterie prêt à être embarqué le plus proche dans le rayon précisé (radius). Si aucune unité ne répond ne peut être embarquée, nil sera retourné.

Paramètres :

| | | |
|---------------|--------|-------------------------|
| radius | number | Rayon pour la recherche |
|---------------|--------|-------------------------|

Valeurs de retour :

| | | |
|--|-------|---|
| | table | Groupe d'infanterie le plus proche pouvant être embarqué (ATME.C_Group) |
| | nil | Si aucun groupe trouvé |

Object:getPosition()

Description : Cette méthode permet de récupérer la position de l'unité joueur (Point 3D).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|-------|---|
| Sans erreur : | table | Point 3D en x, y et z représentant la position de l'unité d'un joueur |
|---------------|-------|---|

Object:getPseudo()

Description : Cette méthode permet de récupérer le pseudo d'un joueur, utile en multijoueur. Le pseudo est le nom de connexion. En monoplayer, ce pseudo est fixe.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|------------------|
| Sans erreur : | string | Pseudo du joueur |
|---------------|--------|------------------|

Object:getRollAxisVector()

Description : Cette méthode retourne un vecteur correspondant à l'axe longitudinal de l'unité AI, orienté vers l'avant.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|-------|--|
| | table | Vecteur de type ATME.C_Vector3D |
|--|-------|--|

Object: getSpeed()

Description : Cette méthode permet de récupérer la vitesse de l'unité d'un joueur en m/s (calculée sur les 3 axes).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|----------------------------------|
| Sans erreur : | number | Vitesse de l'unité joueur en m/s |
|---------------|--------|----------------------------------|

Object: getSpeedAzimuth()

Description : Cette méthode permet de récupérer la route vraie (cap du vecteur vitesse sur les axes x et z) de l'unité d'un joueur.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|-------------------------|
| Sans erreur : | number | Route en degrés (0-360) |
|---------------|--------|-------------------------|

Object: getTypeName()

Description : Cette méthode permet de récupérer le nom du type de l'unité d'un joueur.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|------------------------------------|
| Sans erreur : | string | Nom du type de l'unité d'un joueur |
|---------------|--------|------------------------------------|

Object: getVelocityVector()

Description : Cette méthode permet de récupérer le vecteur vitesse de l'unité d'un joueur

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|-------|--|
| Sans erreur : | table | Instance de C_Vector3D représentant la vitesse |
|---------------|-------|--|

Object: getVSpeed()

Description : Cette méthode permet de récupérer la vitesse verticale de l'unité d'un joueur en m/s (prise sur l'axe y).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | number | Vitesse verticale de l'unité d'un joueur en m/s |
|---------------|--------|---|

Object: inAir()

Description : Cette méthode permet de savoir si l'unité d'un joueur est en l'air ou non.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, l'unité AI est en l'air. False sinon |
|---------------|---------|---|

Object:isEngineStarted()

Description : Cette méthode permet de savoir si l'unité du joueur a ses moteurs démarrés.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|--|
| boolean | Si true, l'unité du joueur a ses moteurs démarrés. False sinon |
|---------|--|

Object:isGroundVehicle()

Description : Cette méthode permet de savoir si l'unité du joueur est un véhicule terrestre.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | Si true, l'unité du joueur est un véhicule terrestre. False sinon |
|---------|---|

Object:isHelicopter()

Description : Cette méthode permet de savoir si l'unité d'un joueur est un hélicoptère.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | Si true, l'unité d'un joueur est un hélicoptère. False sinon |
|---------------|---------|--|

Object:isInDCSZone(zoneName)

Description : Cette méthode permet de savoir si l'unité du joueur est dans une zone DCS définie dans l'éditeur de mission.

Paramètres : Aucun

| | | |
|-----------------|--------|-----------------|
| zoneName | string | Nom de la zone. |
|-----------------|--------|-----------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, l'unité du joueur est dans la zone zoneName . False sinon |
| Si erreur : | nil | |

Object:isRouteInDirection(reference)

Description : Cette méthode permet de savoir si l'unité du joueur est dans la bonne direction par rapport à une référence donnée.

Une référence peut être :

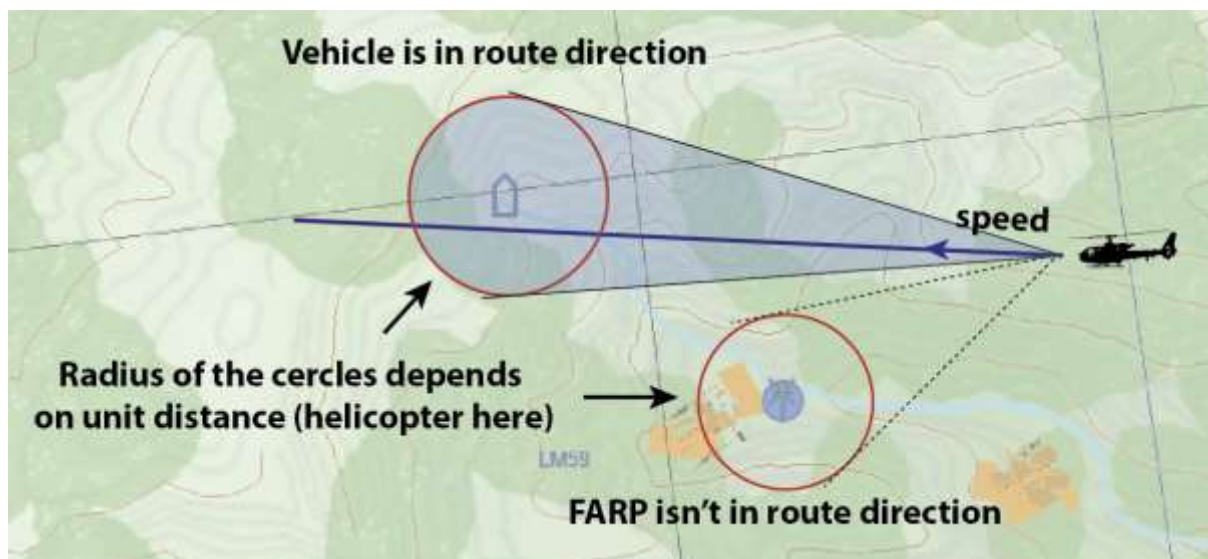
- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, l'unité du joueur est dans la bonne direction. False sinon |
| Si erreur : | nil | |



Object:isInZone2D(reference, radius)

Description : Cette méthode permet de savoir si l'unité du joueur est dans une zone horizontale définie par un cercle ayant pour centre une référence et un rayon radius.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|--------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon du cercle représentant la zone horizontale en m |

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | Si true, l'unité du joueur est dans la zone horizontale. False sinon |
| Si erreur : | nil | |

Object:isInZone3D(reference, radius)

Description : Cette méthode permet de savoir si l'unité du joueur est dans une zone sphérique définie par une boule ayant pour centre une référence et un rayon radius.

Une référence peut être :

- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|------------------|--------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon de la boule représentant la zone sphérique en m |

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | Si true, l'unité du joueur est dans la zone. False sinon |
| Si erreur : | nil | |

Object:isNear(reference, radius, deltaAltitude)

Description : Cette méthode permet de savoir si une référence est proche de l'unité du joueur. La zone est définie par un cylindre représenté par un cercle horizontal de rayon radius, les deux côtés du cylindre correspondent à une différence d'altitude en prenant l'unité du joueur pour centre.

Une référence peut être :

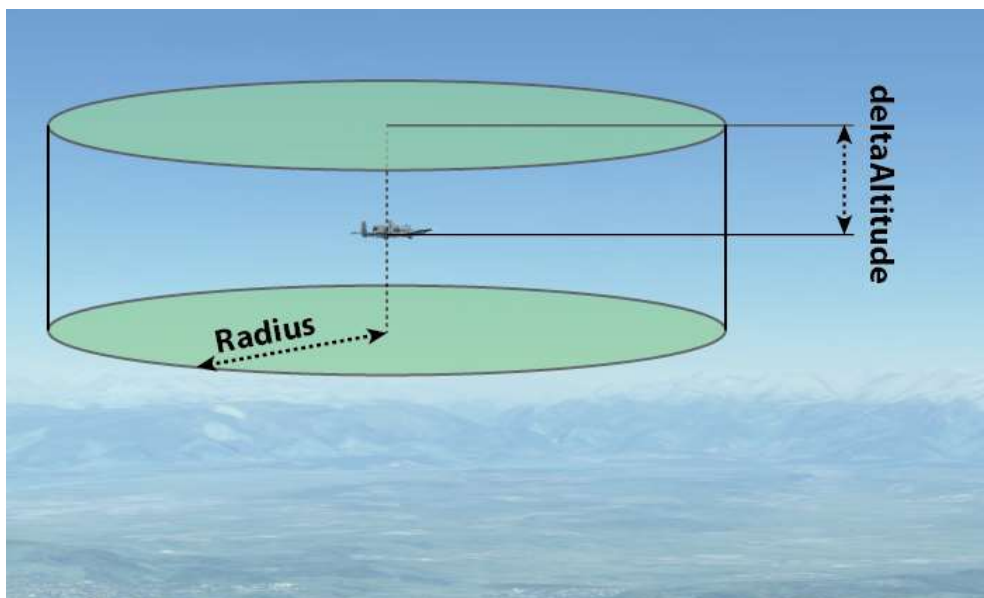
- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres : Aucun

| | | |
|----------------------|--------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
| radius | number | Rayon du cercle représentant la zone cylindrique en m |
| deltaAltitude | number | Ecart d'altitude en m |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | Si true, la référence est dans la zone cylindrique. False sinon |
| Si erreur : | nil | |



Object:isPersonnelCarrier()

Description : Cette méthode permet de savoir si l'unité du joueur est transport de troupes.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|---------|---|
| | boolean | Si true, l'unité du joueur est un transport de troupes. False sinon |
|--|---------|---|

Object:isPlane()

Description : Cette méthode permet de savoir si l'unité d'un joueur est un avion.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|---------|--|
| Sans erreur : | boolean | Si true, l'unité d'un joueur est un avion. False sinon |
|---------------|---------|--|

Object:load(groupToBoard, radius)

Description : Cette méthode permet d'embarquer un groupe d'infanterie. L'unité du joueur doit être un véhicule habilité, c'est à dire avoir une capacité de transport de troupes (personnel carrier). Le radius précise le rayon pris en compte pour l'embarquement. Le groupe à embarquer doit se situer dans le rayon d'embarquement de l'unité du joueur. L'unité du joueur de transport de troupes doit aussi être posée et avoir une vitesse horizontale inférieure à 1m/s.

Un véhicule de transport de troupes a une capacité d'embarquement maximale en terme d'unités d'infanterie. Il peut embarquer plusieurs groupes d'infanteries tant que cette capacité maximale n'est pas dépassée. La masse totale du transport n'est actuellement pas changée.

L'événement ATME Core "TRANSPORT_END_OF_BOARDING" est émis à la fin de l'embarquement.

Paramètres :

| | | |
|---------------------|--------|--|
| groupToBoard | table | Groupe d'infanterie à embarquer (ATME.C_Group) |
| radius | number | Rayon limite pour l'embarquement |

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | False si pas d'erreur, pas d'autres retour. |
| Si erreur : | boolean | True en cas d'erreur, le deuxième retour indique l'erreur |
| | string | Codes d'erreur : |
| | | "LOAD_UNIT_TOO_FAR" |
| | | "LOAD_UNIT_IN_AIR" |
| | | "LOAD_UNIT_BAD_SPEED" |
| | | "LOAD_TOO_MUCH_INFANTRY" |
| | | "LOAD_INFANTRY_GROUP_NOT_READY" |
| | | "LOAD_NO_INFANTRY_GROUP_AVAILABLE" |

Liste des types d'hélicoptères habilités :

"UH-1H" limité à 8 infanteries, "Mi-8MT" limité à 16 infanteries, "SA342M" limité à 2 infanteries, "SA342L" limité à 2 infanteries, "SA342Mistral" limité à 2 infanteries

Liste des types de véhicules terrestres habilités :

"AAV7" limité à 25 infanteries, "M-113" limité à 11 infanteries, "LAV-25" limité à 6 infanteries, "M1126 Stryker ICV" limité à 9 infanteries, "M-2 Bradley" limité à 6 infanteries, "BTR-80" limité à 9 infanteries, "BTR-D" limité à 12 infanteries, "MTLB" limité à 10 infanteries, "BMD-1" limité à 4 infanteries, "BMP-1" limité à 8 infanteries, "BMP-2" limité à 7 infanteries, "BMP-3" limité à 7 infanteries, "UAZ-469" limité à 6 infanteries, "Tigr_233036" limité à 6 infanteries, "GAZ-3307" limité à 16 infanteries, "GAZ-3308" limité à 16 infanteries, "Ural-4320T" limité à 20 infanteries, "Ural-4320-31" limité à 20 infanteries, "M 818" limité à 20 infanteries, "KAMAZ Truck" limité à 20 infanteries.

Object:soundOnce(file)

Description : Cette méthode permet de jouer un fichier son pour un joueur particulier. Le fichier son doit exister dans la mission.

Paramètres :

| | | |
|-------------|--------|----------------------------|
| file | string | Nom du fichier son à jouer |
|-------------|--------|----------------------------|

Valeurs de retour : Aucune

Object:soundChange(file, interval, forced)

Description : Cette méthode permet de modifier le fichier son déjà en diffusion en boucle pour un joueur particulier. Le changement de liste se fait soit immédiatement (forced à true), soit au terme de la lecture du fichier en cours (forced = false). Pour arrêter la lecture, la fonction **soundStop** doit être utilisée. Le fichier son doit exister dans la mission.

Paramètres :

| | | |
|-----------------|---------|--|
| file | string | Nom du fichier son à jouer |
| interval | number | Intervalle de lecture en secondes. |
| forced | boolean | True pour une prise en compte immédiate, false pour laisser la lecture en cours se terminer. |

Valeurs de retour : Aucune

Object:soundPause(duration)

Description : Cette méthode permet de suspendre la lecture active en boucle pour une durée précise. Au terme de la pause, la lecture reprend automatiquement en début de fichier. Le début de fichier sera le fichier suivant en cas de playlist. Cette fonction est utile pour insérer ponctuellement un autre son dans une lecture en boucle.

Paramètres :

| | | |
|-----------------|--------|------------------------------|
| duration | number | Durée de la pause en seconde |
|-----------------|--------|------------------------------|

Valeurs de retour : Aucune

Object:soundStart(file, interval)

Description : Cette méthode permet de jouer à intervalle régulier un fichier son pour un joueur particulier. Le passage se fait de manière forcée par arrêt de la lecture du fichier en cours de diffusion. Pour arrêter la lecture, la fonction **soundStop** doit être utilisée. Le fichier son doit exister dans la mission.

Paramètres :

| | | |
|-----------------|--------|----------------------------------|
| file | string | Nom du fichier son à jouer |
| interval | number | Intervalle de lecture en seconde |

Valeurs de retour : Aucune

Object:soundStop()

Description : Cette méthode arrête définitivement la lecture en boucle en cours de diffusion.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:soundChangePlayList(playList, interval, forced)

Description : Cette méthode permet de modifier une liste de fichiers son déjà en diffusion pour un joueur particulier. Le mode de lecture (séquentiel ou aléatoire) ne change pas. L'intervalle fixe le nombre de secondes de lecture avant de passer au fichier suivant. Le changement de liste se fait soit immédiatement (forced à true), soit au terme de la lecture du fichier en cours (forced = false). Pour arrêter la lecture, la fonction **soundStop** doit être utilisée. Les fichiers son doivent exister dans la mission.

Paramètres :

| | | |
|-----------------|---------|--|
| playList | table | Liste indexée des noms de fichier son à jouer. Le premier fichier est à l'index 1. |
| interval | number | Intervalle de lecture en secondes et unique pour l'ensemble des fichiers. |
| forced | boolean | True pour une prise en compte immédiate, false pour laisser la lecture en cours se terminer avant de basculer de playlist. |

Valeurs de retour : Aucune

Object:soundStartPlayList(playList, randomRead, interval)

Description : Cette méthode permet de jouer une liste de fichiers son pour un joueur particulier. Cette lecture peut être séquentielle ou aléatoire. Dans le cas de lecture séquentielle, après la lecture du dernier fichier de la liste, la lecture reprend au premier fichier. L'intervalle fixe le nombre de secondes de lecture avant de passer au fichier suivant. Le passage se fait de manière forcée par arrêt de la lecture du fichier en cours de diffusion. Pour arrêter la lecture, la fonction **soundStop** doit être utilisée. Les fichiers son doivent exister dans la mission.

Paramètres :

| | | |
|-------------------|---------|--|
| playList | table | Liste indexée des noms de fichier son à jouer. Le premier fichier est à l'index 1. |
| randomRead | boolean | True pour une lecture aléatoire, false pour une lecture séquentielle. |
| interval | number | Intervalle de lecture en secondes et unique pour l'ensemble des fichiers. |

Valeurs de retour : Aucune

Object:unload(id)

Description : Cette méthode permet de débarquer un groupe d'infanterie présent dans un transport de troupes. Id correspond à l'index dans l'ordre d'embarquement, si besoin, il faudra utiliser **getGroupsNameOnBoard** afin de récupérer la liste indexée des troupes à bord. Id devra correspondre à l'un des index de cette liste de noms.

L'événement ATME Core "TRANSPORT_END_OF_DISEMBARK" est émis à la fin du débarquement.

Paramètres :

| | | |
|-----------|--------|----------------------|
| id | number | Index d'embarquement |
|-----------|--------|----------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | False si pas d'erreur |
| | table | Instance du groupe débarqué |
| | number | Nombre d'unités perdues lors du débarquement |
| Si erreur : | boolean | True en cas d'erreur, le deuxième retour indique l'erreur |
| | string | Codes d'erreur : |
| | | "UNLOAD_UNIT_IN_AIR" |
| | | "UNLOAD_UNIT_BAD_SPEED" |
| | | "UNLOAD_BAD_ID" |

Object:whichSide(reference)

Description : Cette fonction permet de connaître la position relative (droite ou gauche) de la référence par rapport à l'unité du joueur. L'orientation est fixé par l'axe longitudinal et le vecteur vitesse (voir schéma ci-dessous avec deux références en exemple).

Une référence peut être :

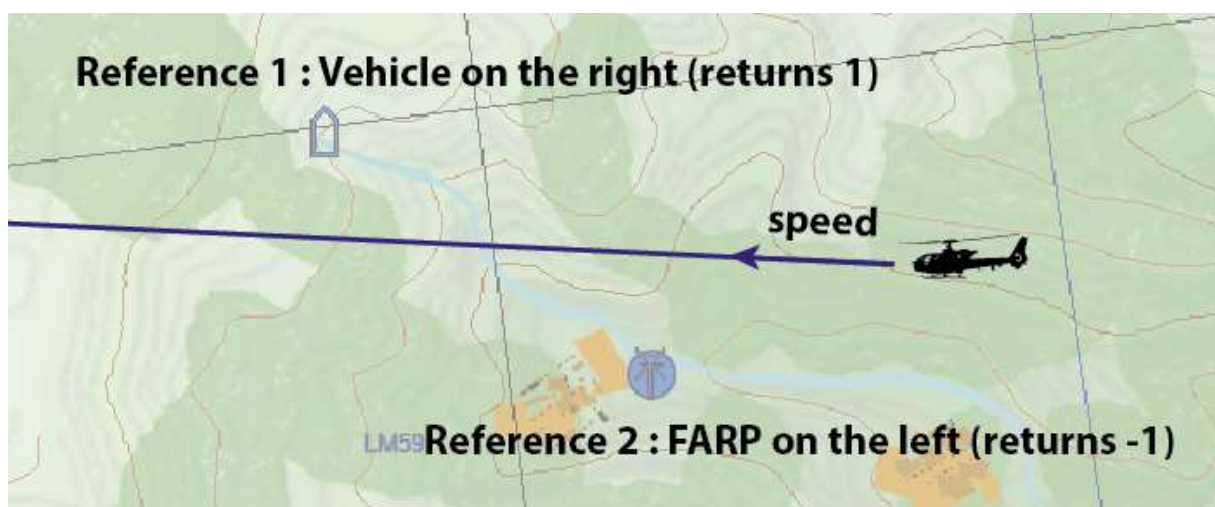
- Un point 2D ou 3D,
- Une unité AI représentée par sa position
- Une unité joueur représentée par sa position
- Un objet statique représenté par sa position

Paramètres :

| | | |
|------------------|-------|--|
| reference | table | Point 3D en x, y et z, ou point 2D en x,y Unité AI. Type ATME.C_AIUnit Unité joueur. Type ATME.C_Player Objet statique. Type ATME.C_StaticObject |
|------------------|-------|--|

Valeurs de retour :

| | | |
|---------------|--------|--|
| Sans erreur : | number | -1 : la référence est à gauche de l'unité du joueur 1 : la référence est à droite de l'unité du joueur 0 : la référence est dans l'axe longitudinal de l'unité du joueur |
| Si erreur : | nil | |



Classe ATME.C_Race

Cette classe permet de définir et de gérer des courses entre joueurs. Une course est définie par un ensemble de portes avec un piquet gauche et un piquet droit. Les piquets gauches et droits doivent être des objets statiques avec un nom unique suivi de « # » et d'un numéro incrémental allant de 1 à 100. Ces noms seront définis directement dans l'éditeur de mission. Le numéro de piquet droit et du piquet gauche doivent être identiques et définit le numéro de porte. La distance entre les deux piquets d'une même porte doit être inférieure à 500m.

La porte ayant le numéro le plus petit est la porte de départ. La porte d'arrivée a le numéro le plus grand. Les portes se passent dans un ordre précis, de la plus porte ayant le numéro le plus petit à la porte ayant le numéro le plus grand.

Le numéro n'ont pas à être consécutifs. Aussi, une porte aura un numéro d'index correspondant au numéro incrémental défini dans l'éditeur de mission. Le numéro de passage pourra être différent et correspondra au numéro de porte dans la course. Exemple :

- S'il manque la porte #003 dans l'éditeur de mission, porte constituée de "**piquetG #003**" et "**piquetD #003**", alors la troisième porte de la course sera la porte #004.

Attention, dans l'éditeur de mission il ne devra y avoir qu'un seul espace entre le nom et le #.

Les temps de course s'expriment en seconde avec 2 décimales soit une résolution au 1/100^{ème} de seconde. Une course génère des événements Core pour chaque joueur inscrit.

| |
|--|
| ATME.C_Race(name, leftSideName, rightSideName, nbTurns) |
|--|

Description : Cette fonction crée une course.

Paramètres :

| | | |
|----------------------|--------|--|
| name | string | Nom de la course. |
| leftSideName | string | Nom général sans #XXX ni l'espace des objets statiques représentant les piquets gauches. |
| leftRightName | string | Nom général sans #XXX ni l'espace des objets statiques représentant les piquets droits. |
| NbTurns | number | Nombre de tours pour la course. |

Valeurs de retour :

| | | |
|-------------|-------|------------------------------|
| Sans erreur | table | Nouvelle instance de C_Race. |
| Si erreur | nil | |

ATME.C_Race.exists(name)

Description : Fonction vérifiant si une course existe.

Paramètres :

| | | |
|-------------|--------|------------------|
| name | string | Nom de la course |
|-------------|--------|------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True si l'instance C_Race existe, false sinon |
| Si erreur : | nil | |

ATME.C_Race.getByName(name)

Description : Fonction permettant de récupérer une course à partir de son nom.

Paramètres :

| | | |
|-------------|--------|------------------|
| name | string | Nom de la course |
|-------------|--------|------------------|

Valeurs de retour :

| | | |
|---------------|-------|---|
| Sans erreur : | table | Instance de la classe C_Race ou nil si inexistante. |
| Si erreur : | nil | |

Object:addPlayer(player)

Description : Cette méthode inscrit un joueur comme participant à la course.

Paramètres :

| | | |
|---------------|-------|--|
| player | table | Instance C_Player représentant le joueur |
|---------------|-------|--|

Valeurs de retour : Aucune

Object:delete()

Description : Cette méthode supprime course. Ceci engendre la suppression dans ATME Core de tous les éléments de cette course. Elle n'existera plus vue d'ATME. Il convient mettre à nil les tables pointant sur cette course dans le ou les modules concernés car l'instance détruite ne sera plus utilisable.

Paramètres : Aucun

Valeurs de retour : Aucun

Object:displayForAllPlayers(text, duration)

Description : Cette méthode permet d'afficher un message destiné à tous les joueurs participant à une course. Le message sera affiché pendant la durée indiquée.

Paramètres :

| | | |
|-----------------|--------|--|
| text | string | Texte du message à afficher. |
| duration | number | Durée en secondes de l'affichage du message. |

Valeurs de retour : Aucune

Object:isPlayerInRace(player)

Description : Cette méthode vérifie si un joueur participe à la course.

Paramètres :

| | | |
|---------------|-------|--|
| player | table | Instance C_Player représentant le joueur |
|---------------|-------|--|

Valeurs de retour :

| | | |
|--|---------|--|
| | boolean | True si le joueur participe, false sinon |
|--|---------|--|

Object:getName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Race ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object:getDoorIndex(offset)

Description : Cette méthode retourne l'index d'une porte dont le numéro est connu. Le numéro de porte est le numéro dans la course. Si les index sont consécutifs, le numéro de porte sera équivalent à l'index. Si les numéros d'index ne sont pas consécutifs, le numéro de porte sera différent. Dans une course les numéros de porte sont toujours consécutifs allant de la première porte (offset = 1) à la dernière porte (offset correspondant au nombre de portes définies) .

Paramètres :

| | | |
|---------------|--------|-----------------------------------|
| offset | number | Numéro de la porte dans la course |
|---------------|--------|-----------------------------------|

Valeurs de retour :

| | |
|--------|--|
| number | Index de la porte tel que défini dans l'éditeur de mission |
|--------|--|

Object:getDoorSides(index)

Description : Cette méthode retourne les objets statiques gauche et droits formant la porte d'index passé en paramètre. Pour rappel, l'index est le numéro fixé dans l'éditeur de mission.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Instance C_StaticObject définissant le piquet gauche |
| table | Instance C_StaticObject définissant le piquet droit |

Object: getLapAtDoor(index)

Description : Cette méthode indique si la porte, dont l'index est passé en paramètre, est une mesure de temps de temps intermédiaire ou pas.

Paramètres :

| | | |
|--------------|--------|--|
| index | number | Index de la porte tel que défini dans l'éditeur de mission |
|--------------|--------|--|

Valeurs de retour :

| | |
|---------|--|
| boolean | True si la porte est une mesure de temps intermédiaire, false sinon |
|---------|--|

Object: getMaxAltitude()

Description : Cette méthode permet de récupérer l'altitude sol (AGL) maximale autorisée lors du franchissement d'une porte.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------------------------|
| number | Altitude AGL maximale autorisée |
|--------|---------------------------------|

Object: getMaxAltitudePenalty()

Description : Cette méthode permet de récupérer la pénalité en secondes appliquée en cas de dépassement de l'altitude sol (AGL) maximale autorisée lors du franchissement d'une porte.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------------|
| number | Pénalité en secondes |
|--------|----------------------|

Object: getMissedDoorPenalty()

Description : Cette méthode permet de récupérer la pénalité en secondes appliquée en cas de porte loupée.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------------|
| number | Pénalité en secondes |
|--------|----------------------|

Object: getName()

Description : Cette méthode permet de récupérer le nom de la course.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|------------------|
| string | Nom de la course |
|--------|------------------|

Object: getNbDoors()

Description : Cette méthode retourne le nombre de portes de la course.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-------------------------------|
| number | Nombre de portes de la course |
|--------|-------------------------------|

Object: getNbTurns()

Description : Cette méthode retourne le nombre de tours de la course.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|------------------------------|
| number | Nombre de tours de la course |
|--------|------------------------------|

Object:getPlayerNextDoorIndex(player)

Description : Cette méthode retourne l'index de la prochaine porte à franchir par un joueur donné.

Paramètres :

| | | |
|---------------|-------|--|
| player | table | Instance C_Player représentant le joueur |
|---------------|-------|--|

Valeurs de retour :

| | |
|--------|--|
| number | Index de la prochaine porte à franchir par le joueur |
|--------|--|

Object:getRanking()

Description : Cette méthode retourne le classement des joueurs inscrits à la course. C'est une table indexée avec comme champs pour chaque entrée :

- player : définissant le joueur (instance C_Player)
- bestTime : son meilleur temps en secondes (avec 2 décimales).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|----------------------------------|
| table | Table indexée pour le classement |
|-------|----------------------------------|

Exemple :

Object:removeAllPlayers()

Description : Cette méthode annule l'inscription de tous les joueurs inscrits à la course.

Paramètres :

| | | |
|---------------|-------|--|
| player | table | Instance C_Player représentant le joueur |
|---------------|-------|--|

Valeurs de retour : Aucune

Object:removePlayer(player)

Description : Cette méthode annule l'inscription d'un joueur à la course.

Paramètres :

| | | |
|---------------|-------|--|
| player | table | Instance C_Player représentant le joueur |
|---------------|-------|--|

Valeurs de retour : Aucune

Object:resetMaxAltitudeRule()

Description : Cette méthode supprime le contrôle d'altitude maximum autorisé et la pénalité associée.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:resetMissedDoorRule()

Description : Cette méthode supprime la pénalité en seconde appliquée si un joueur loupe une porte.

Paramètres : Aucun

Valeurs de retour : Aucune

Object:setLapAtDoor(index)

Description : Cette méthode définit la porte, dont l'index est passé en paramètre, comme une mesure de temps de temps intermédiaire.

Paramètres :

| | | |
|--------------|--------|--|
| index | number | Index de la porte tel que défini dans l'éditeur de mission |
|--------------|--------|--|

Valeurs de retour : Aucun

Object:setMaxAltitudeRule(maxAltitude, penalty)

Description : Cette méthode définit l'altitude sol (AGL) maximale autorisée pour passer une porte et la pénalité en seconde appliquée si un joueur dépasse cette altitude.

Cette pénalité ne s'applique pas si le joueur loupe la porte.

Paramètres :

| | | |
|--------------------|--------|--|
| maxAltitude | number | Altitude AGL maximale autorisée en m pour passer une porte |
| penalty | number | Nombre de secondes de pénalité |

Valeurs de retour : Aucune

Object:setMissedDoorRule(penalty)

Description : Cette méthode définit la pénalité en seconde appliquée si un joueur loupe une porte.

Paramètres :

| | | |
|----------------|--------|--------------------------------|
| penalty | number | Nombre de secondes de pénalité |
|----------------|--------|--------------------------------|

Valeurs de retour : Aucune

Object : soundForAllPlayers(file)

Description : Cette méthode permet de jouer un fichier son à tous les joueurs participant à une course.

Paramètres :

| | | |
|-------------|--------|----------------------------|
| file | string | Nom du fichier son à jouer |
|-------------|--------|----------------------------|

Valeurs de retour : Aucune

Classe ATME.C_Smoke

ATME.C_Smoke(colorName, point)

Description : Cette fonction crée une fumée sur un point de la carte qui sera activée ultérieurement.

Paramètres :

| | | |
|------------------|--------|--|
| colorName | string | Couleur de la fumée : "BLUE", "GREEN", "RED", "WHITE" et "ORANGE". La valeur "RANDOM" engendre une couleur aléatoire à prendre parmi celles existantes. |
| point | table | Localisation sur la carte, point 3D en x, y et z. |

Valeurs de retour :

| | | |
|-------------|-------|-------------------------------|
| Sans erreur | table | Nouvelle instance de C_Smoke. |
| Si erreur | nil | |

Object:activate(restart)

Description : Cette méthode active la fumée. Si restart est à true, la fumée sera réactivée automatiquement au bout de 300s (5 minutes) et un stop sera nécessaire pour l'arrêter.

Paramètres :

| | | |
|----------------|---------|---|
| restart | boolean | Si true, la fumée continuera après 5min. Si false, s'arrête après 5 min |
|----------------|---------|---|

Valeurs de retour : Aucune

Object:getName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Smoke ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object:getColor()

Description : Cette méthode retourne la couleur de la fumée.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| string | Couleur de la fumée : "BLUE", "GREEN", "RED", "WHITE" ou "ORANGE" |
|--------|---|

Object:getPoint()

Description : Cette méthode retourne le point d'où la fumée est déclenchée.

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---|
| table | Point 3D en x, y et z, ou point 2D en x,y |
|-------|---|

Object:stop()

Description : Cette méthode supprime les réactivations ultérieures. La fumée s'arrêtera au terme de l'activation courante, soit au plus 5 min.

Paramètres : Aucun

Valeurs de retour : Aucune

Classe ATME.C_StaticObject

Les instances de cette classe sont créées et supprimées par ATME Core.

ATME.C_StaticObject.exists(name)

Description : Fonction vérifiant si un objet static existe (est en vie) à un instant T.

Paramètres :

| | | |
|-------------|--------|-------------------------|
| name | string | Nom d'un objet statique |
|-------------|--------|-------------------------|

Valeurs de retour :

| | | |
|---------------|---------|---|
| Sans erreur : | boolean | True si l'instance C_StaticObject existe, false sinon |
| Si erreur : | nil | |

ATME.C_StaticObject.getByName(name)

Description : Fonction permettant de récupérer, à partir de son nom, un objet statique (en vie) à un instant T.

Paramètres :

| | | |
|-------------|--------|-------------------------|
| name | string | Nom de l'objet statique |
|-------------|--------|-------------------------|

Valeurs de retour :

| | | |
|---------------|-------|--------------------------------------|
| Sans erreur : | table | Instance de la classe C_StaticObject |
| Si erreur : | nil | |

Object:explode(power)

Description : Cette méthode déclenche une explosion d'une force définie sur l'objet statique.

Paramètres :

| | | |
|--------------|--------|-----------------------|
| power | number | Force de l'explosion. |
|--------------|--------|-----------------------|

Valeurs de retour : Aucune

Object:fireFlare(color)

Description : Cette méthode déclenche le tir d'un flare à partir de l'objet statique.

Paramètres :

| | | |
|--------------|--------|---|
| color | string | Couleur du flare "GREEN", "RED", "WHITE" et "YELLOW". La valeur "RANDOM" engendre une couleur aléatoire prise parmi les couleurs existantes. |
|--------------|--------|---|

Valeurs de retour : Aucune

Object:fireIlluminationBomb(power)

Description : Cette méthode déclenche le tir d'une bombe lumineuse à partir de l'objet statique.

Paramètres :

| | | |
|--------------|--------|------------------------------|
| power | number | Puissance de la bombe tirée. |
|--------------|--------|------------------------------|

Valeurs de retour : Aucune

Object:fireSmoke(color)

Description : Cette méthode déclenche une fumée à proximité de l'objet statique.

Paramètres :

| | | |
|--------------|--------|---|
| color | string | Couleur de la fumée "BLUE" , "GREEN", "RED", "WHITE" et "ORANGE". La valeur "RANDOM" engendre une couleur aléatoire prise parmi les couleurs existantes. |
|--------------|--------|---|

Valeurs de retour : Aucune

Object: getAzimuth()

Description : Cette méthode permet de récupérer l'azimuth (par rapport au Nord) de l'unité AI. Ceci correspond au cap vrai sans prendre en compte la déviation magnétique.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---------------------------|
| number | Azimuth en degrés (0-360) |
|--------|---------------------------|

Object: getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_AIUnit ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object: getCoalitionName()

Description : Cette méthode permet de récupérer le nom de la coalition de l'unité AI.

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---|
| Sans erreur : | string | Nom de la coalition de l'unité : "RED", "BLUE" ou "NEUTRAL" |
|---------------|--------|---|

Object: getCountryName()

Description : Cette méthode permet de récupérer le nom du pays de l'unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|------------------------|
| string | Nom du pays de l'unité |
|--------|------------------------|

Liste des nations (issue de DCS)

"RUSSIA", "UKRAINE", "USA", "TURKEY", "UK", "FRANCE", "GERMANY", "AGGRESSORS", "CANADA", "SPAIN", "THE_NETHERLANDS", "BELGIUM", "NORWAY", "DENMARK", "ISRAEL", "GEORGIA", "INSURGENTS", "ABKHAZIA", "SOUTH_OSETIA", "ITALY", "AUSTRALIA", "SWITZERLAND", "AUSTRIA", "BELARUS", "BULGARIA", "CHEZH_REPUBLIC", "CHINA", "CROATIA", "EGYPT", "FINLAND", "GREECE", "HUNGARY", "INDIA", "IRAN", "IRAQ", "JAPAN", "KAZAKHSTAN", "NORTH_KOREA", "PAKISTAN", "POLAND", "ROMANIA", "SAUDI_ARABIA", "SERBIA", "SLOVAKIA", "SOUTH_KOREA", "SWEDEN", "SYRIA"

Object: getDCSStaticObject()

Description : Cette méthode permet de récupérer l'objet statique tel que géré par DCS (Objet de classe **Unit**).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|---------------------------------------|
| table | Instance de la classe DCS Unit |
|-------|---------------------------------------|

Object:getLife()

Description : Cette méthode permet de récupérer l'information de « vie » d'une unité AI.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------|
| number | Vie de l'unité |
|--------|----------------|

Object:getName()

Description : Cette méthode permet de récupérer le nom de l'objet statique.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-------------------------|
| string | Nom de l'objet statique |
|--------|-------------------------|

Object:getPosition()

Description : Cette méthode permet de récupérer la position de l'objet statique (Point 3D).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|--|
| table | Point 3D en x, y et z représentant la position de l'objet statique |
|-------|--|

Object : getTypeName()

Description : Cette méthode permet de récupérer le nom du type d'un objet statique.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| string | Nom du type de l'objet statique : "Ships", "Planes", "Helicopters", "Warehouses", "Cargos", "Unarmed", "Fortifications" |
|--------|---|

Classe ATME.C_UserTrigger

Les instances de cette classe sont créées à partir des deux fonctions suivantes de la classe ATME.C_Module :

- **createAbsoluteUserTrigger**
- **createFlagRelativeUserTrigger**

Un trigger utilisateur est en effet associé à un module unique, celui qui l'a créé. Il est par ailleurs défini par son nom qui peut donc exister dans deux modules différents sans interaction. A un instant donné, il ne peut y avoir deux triggers créés avec un même nom dans un module donné.

Ces triggers fonctionnent sur la base d'un temps relatif ou absolu. Ils se déclencheront donc sur la période définie et seront ensuite automatiquement détruits, permettant ainsi de créer un nouveau trigger du même nom ultérieurement.

Object : getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_UserTrigger ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------------|
| string | Nom de la classe ATME |
|--------|-----------------------|

Object: getFlag()

Description : Cette méthode permet de récupérer le flag associé à un trigger utilisateur créé avec la fonction **createFlagRelativeUserTrigger**.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|---|
| number | Numéro du flag associé, -1 si pas de flag associé |
|--------|---|

Object: getName()

Description : Cette méthode permet de récupérer le nom d'un trigger programmé et/ou activé.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|----------------|
| string | Nom du trigger |
|--------|----------------|

Object: getTimeEnd()

Description : Cette méthode permet de récupérer le nombre de secondes correspondant à la fin de l'activation du trigger utilisateur.

Si le trigger utilisateur a été créé avec la fonction **createFlagRelativeUserTrigger** de C_Module, cette valeur ne sera significative que si le trigger a été armé (voir fonction **isArmed**).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| number | Nombre de secondes correspondant à la fin de l'activation du trigger utilisateur |
|--------|--|

Object:getTimeStart()

Description : Cette méthode permet de récupérer le nombre de secondes correspondant à l'activation du trigger utilisateur.

Si le trigger utilisateur a été créé avec la fonction **createFlagRelativeUserTrigger** de C_Module, cette valeur ne sera significative que si le trigger a été armé (voir fonction **isArmed**).

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|--|
| number | Nombre de secondes correspondant à l'activation du trigger utilisateur |
|--------|--|

Object:isActivated()

Description : Cette méthode permet de savoir si un trigger utilisateur est actif c'est-à-dire que le temps de mission est compris entre le time start et le time end du trigger.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le trigger a été armé, false sinon. |
|---------|---|

Object:isArmed()

Description : Cette méthode permet de savoir si un trigger utilisateur défini relativement à un flag a été armé, c'est-à-dire que le flag associé a été activé.

Paramètres : Aucun

Valeurs de retour :

| | |
|---------|---|
| boolean | True si le trigger a été armé, false sinon. |
|---------|---|

Classe ATME.C_Vector3D

Cette classe définit la gestion des vecteurs en trois dimensions.

Il est aussi possible d'additionner ou soustraire deux vecteurs.

ATME.C_Vector3D(...)

Description : Cette fonction crée un vecteur 3D à partir des données passées en paramètres.

Paramètres cas 1 : Aucun paramètre – Crée un vecteur 3D nul (0,0,0)

Paramètres cas 2 : Crée la copie d'un vecteur

| | | |
|---------------|-------|------------------------|
| vector | table | Instance de C_Vector3D |
|---------------|-------|------------------------|

Paramètres cas 3 : Crée un vecteur à partir de deux points A et B (vecteur AB)

| | | |
|-----------|-------|---|
| pA | table | Point A - Point 3D en x, y et z, ou point 2D en x,y |
| pB | table | Point B - Point 3D en x, y et z, ou point 2D en x,y |

Paramètres cas 4 : Crée un vecteur à partir de trois coordonnées en x, y et z

| | | |
|----------|--------|-----------------|
| x | number | Coordonnée en x |
| y | number | Coordonnée en y |
| z | number | Coordonnée en z |

Valeurs de retour :

| | | |
|-------------|-------|----------------------------------|
| Sans erreur | table | Nouvelle instance de C_Vector3D. |
| Si erreur | nil | |

Object:get()

Description : Cette méthode permet de récupérer les coordonnées x, y et z d'un vecteur 3D.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------|
| | number | Coordonnée en x |
| | number | Coordonnée en y |
| | number | Coordonnée en z |

Object:getAzimuth()

Description : Cette méthode permet de récupérer l'azimuth du vecteur (par rapport au Nord).

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|---------------------------|
| Sans erreur : | number | Azimuth en degrés (0-360) |
| Si erreur : | nil | |

Object:getClassName()

Description : Cette méthode permet de récupérer le nom de la classe ATME de l'objet, dans ce cas précis « C_Vector3D ». Cette méthode existe pour toutes les classes.

Paramètres : Aucun

Valeurs de retour :

| | | |
|--|--------|-----------------------|
| | string | Nom de la classe ATME |
|--|--------|-----------------------|

Object: getHModule()

Description : Cette méthode permet de récupérer le module horizontal du vecteur, calculé comme suit : $\sqrt{x^2 + z^2}$

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|-----------------------------------|
| Sans erreur : | number | Module horizontal du vecteur en m |
| Si erreur : | nil | |

Object: getModule()

Description : Cette méthode permet de récupérer le module du vecteur, calculé comme suit : $\sqrt{x^2 + y^2 + z^2}$

Paramètres : Aucun

Valeurs de retour :

| | | |
|---------------|--------|------------------------|
| Sans erreur : | number | Module du vecteur en m |
| Si erreur : | nil | |

Object: getX()

Description : Cette méthode permet de récupérer les coordonnées x 3D.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------|
| number | Coordonnée en x |
|--------|-----------------|

Object:getY()

Description : Cette méthode permet de récupérer les coordonnées y 3D.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------|
| number | Coordonnée en y |
|--------|-----------------|

Object:getZ()

Description : Cette méthode permet de récupérer les coordonnées z 3D.

Paramètres : Aucun

Valeurs de retour :

| | |
|--------|-----------------|
| number | Coordonnée en z |
|--------|-----------------|

Object:toUnitVector()

Description : Cette méthode retourne un vecteur unitaire colinéaire à ce vecteur 3D. Le vecteur sera un vecteur nul si l'objet courant est un vecteur nul (module = 0).

Paramètres : Aucun

Valeurs de retour :

| | |
|-------|------------------------|
| table | Instance de C_Vector3D |
|-------|------------------------|

Object:scalaireH(vector)

Description : Cette méthode retourne le produit scalaire horizontal, calculé sur les axes x et z, entre le vecteur courant et le vecteur passé en paramètre.

Paramètres :

| | | |
|---------------|-------|------------------------|
| vector | table | Instance de C_Vector3D |
|---------------|-------|------------------------|

Valeurs de retour :

| | |
|--------|-----------------------------|
| number | Produit scalaire horizontal |
|--------|-----------------------------|