

Des Robots et Des Dalles (DR2D)

- La console de commande -

(et son protocole utilisant les WebSocket HTML5)

Table des matières

Préambule.....	1
Quelques définitions.....	2
Organigramme.....	3
Le protocole / les trames.....	4
Les séquences.....	5
ShortMessageService.....	5
PlayersDiscovery.....	5
GameInitialisation.....	6
InitDeclareBoard.....	6
InitPlayerFeatures.....	7
InitPlayerLocalization.....	7
InitStatusOfCompetitors.....	8
InitLocateArtefacts.....	8
QuestStart.....	9
RoundRequireCartouche.....	9
RoundRobotMovement.....	10
RoundBoardMovement.....	10
RoundComputedMovements [non implémenté, réservé pour le jeu physique].....	11
RoundStatusOfCompetitors.....	12
RoundBoardAction.....	12
QuestAchieved.....	13
GameFinalisation.....	13
Annexe 1, Les différents types de dalles.....	14
Annexe 2, Sous divisions des mouvements.....	15

Préambule

Le dialogue se fait en encodage JSON sur WebSocket HTML5.

Pour s'aider à mettre en forme: <http://www.jsoneditoronline.org/>

Quelques définitions

Le plateau de jeu se compose de 244 dalles (12x12). Certaines sont neutres, d'autres portent une contrainte (convoyeur, laser, mur, etc).

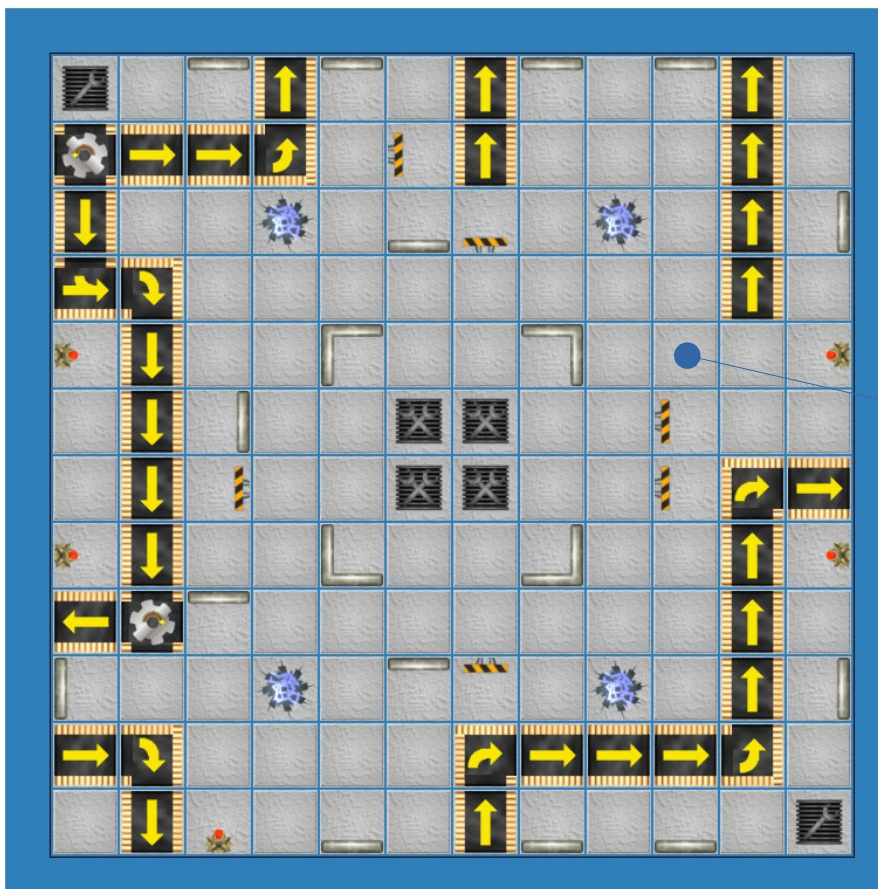
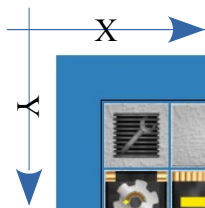
Le repérage se fait à partir du point en haut à gauche sur le dessin ci dessous.

Les X vont vers la droite, les Y vers le bas.

La dalle en haut à gauche aura pour coordonnée [1,1] et la dalle en bas à droite [12,12].

Lors d'un listage des dalles, elles seront numérotés de 0 à 243, de gauche à droite puis de haut en bas.

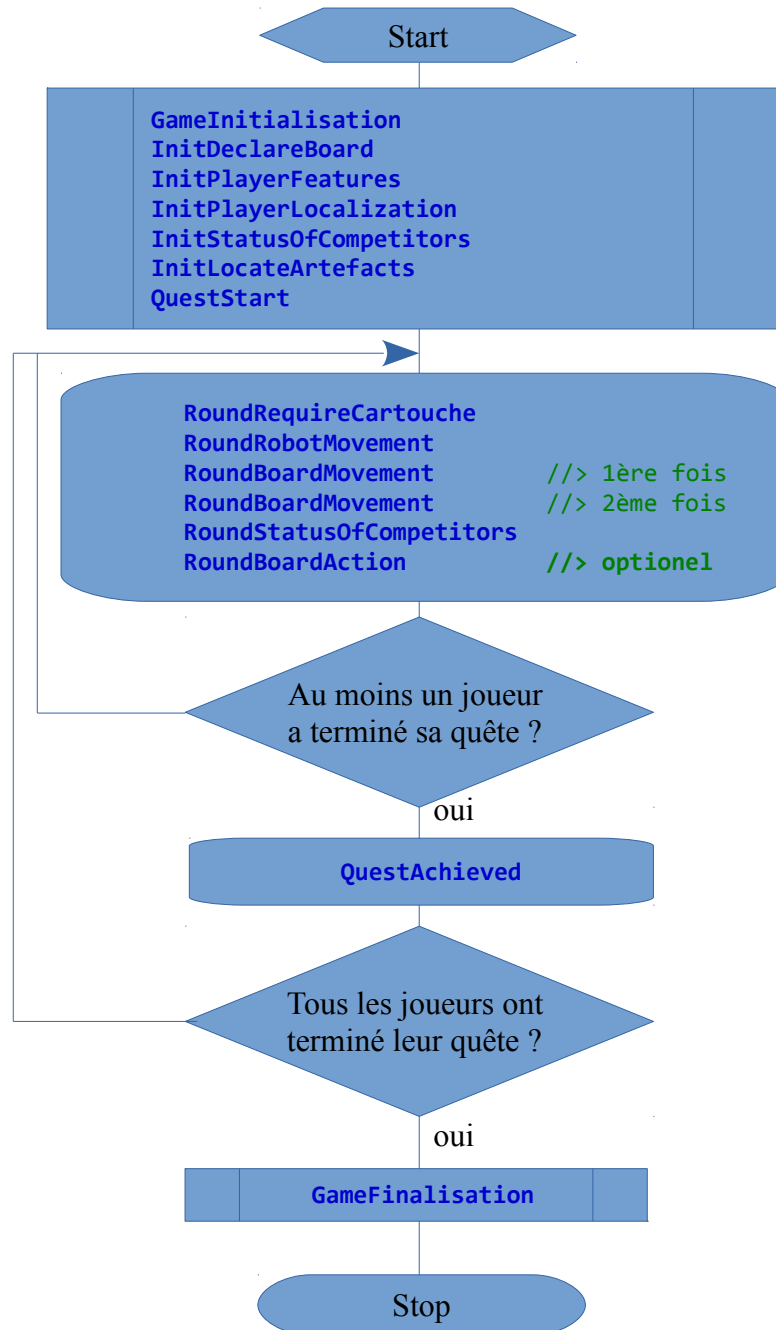
Pour la géolocalisation des mécanoïds, on utilisera la rose des vents. Une orientation vers le haut corespondra au nord, vers la droite à l'est et ainsi de suite.



Par exemple,
cette dalle à pour
coordonnées [10,5]
et pour n° 54

Organigramme

Ci dessous une vue globale du déroulement d'une partie avec les différentes étapes impliquées. A chaque étape correspond une séquence qui fera l'objet d'un échange de trames entre le jeu maître et la console.



Le protocol / les trames

Trois type de trame:

- les trames de service, de ou vers la console, identifiées par le nom "ShortMessageService"
- les trames vers la console identifiées par le nom "stgyCall"
- les trames de réponse vers le jeu identifiées par le nom "stgyReply".

Composition d'une trame de service:

```
// Un message envoyé à la console ou reçu de la console. Pour info/debug.
// msgCode: un code quelconque, peut être nul,
// msgDesc: une description, un message, une erreur,
// msgArgs: libre
// (pas de réponse attendue)
{
  "ShortMessageService": {
    "msgCode": "0",
    "msgDesc": "Bonjour",
    "msgArgs": null
  }
}
```

Composition d'une trame de jeu:

```
// Un message de communication lors du jeu. Nécessite une réponse.
// seq:      un nom de séquence, non nul, non modifiable
// token:    un jeton unique, fourni lors de l'appel 'stgyCall' et réutilisé par 'stgyReply'
// argsIn:   une série d'arguments en lecture seule (seulement avec "stgyCall")
// argsRef   une série d'arguments modifiables

{
  "stgyCall": {
    "seq": "GameInitialisation", // ou "stgyReply"
    "token": "123456789", // un identifiant de séquence
    "argsIn": {
      "gameContext": null
    },
    "argsRef": null
  }
}
```

Remarque 1: argsIn et argsRef peuvent être nuls.

Remarque 2: Une réponse valide, la plus élémentaire, consiste à renvoyer le contenu de l'appel en changeant "stgyCall" par "stgyReply" et en supprimant le champ argsIn.

Les séquences

ShortMessageService

```
// Un message envoyé à la console ou reçu de la console. Pour info/debug.  
// msgCode: un code quelconque, peut être nul,  
// msgDesc: une description, un message, une erreur,  
// msgArgs: libre  
Message type: (pas de réponse attendue)  
{  
  "ShortMessageService": {  
    "msgCode": "0",  
    "msgDesc": "Bonjour",  
    "msgArgs": null  
  }  
}
```

```
// découverte des joueurs présents
```

PlayersDiscovery

```
// Un message envoyé à la console pour valider l'inscription du joueur (un SMS en fait)  
Message type: (pas de réponse attendue)  
{  
  "ShortMessageService": {  
    "msgCode": "PlayersDiscovery",  
    "msgDesc": "Bienvenue!",  
    "msgArgs": null  
  }  
}
```

GameInitialisation

```
// Premier appel du jeu.
// gameContext => fournit le contexte de jeu (non utilisé pour le moment).
// playerUID => un jeton (différent pour chaque joueur) et unique pour la session de jeu
Appel type:
{
  "stgyCall": {
    "seq": "GameInitialisation",
    "token": "123456789",
    "argsIn": {
      "gameContext": null,
      "playerUID": "1234567890" // à mémoriser si on veut pouvoir se reconnecter
    },
    "argsRef": null
  }
}
Réponse type:
{
  "stgyReply": {
    "seq": "GameInitialisation",
    "token": "123456789",
    "argsRef": null
  }
}
```

InitDeclareBoard

```
// Reçoit la configuration d'un plateau, c.a.d. les dimensions (12x12) et
// une liste de définition des dalles actives, par exemple, laser en 2:3, presse en 4:3, etc
// Les dalles non listées sont les dalles neutres.
// Peut être appelé plus d'une fois s'il y a plusieurs plateaux.
Appel type:
{
  "stgyCall": {
    "seq": "InitDeclareBoard",
    "token": "123456789",
    "argsIn": {
      "board": {
        "boardIdx": 0,
        "boardWidth": 12,
        "boardHeight": 12,
        "boardActiveTiles": [
          {
            "tileType": "0x16, murLaser_NS",
            "tileLocX": 2,
            "tileLocY": 3
          },
          { . . . }
        ]
      }
    },
    "argsRef": null
  }
}
Réponse type:
{
  "stgyReply": {
    "seq": "InitDeclareBoard",
    "token": "123456789",
    "argsRef": null
  }
}
```

InitPlayerFeatures

// Le joueur peut modifier son avatar. Le nom est tronqué après 11 caractères. Pour la couleur de
// fond, les champs A, R, G, B sont en hexa. (D'autres définitions viendront plus tard.)

Appel type:

```
{
  "stgyCall": {
    "seq": "InitPlayerFeatures",
    "token": "123456789",
    "argsIn": {"mcndId": "03"},
    "argsRef": {
      "avatar": {
        "name": "mécanoïd03",
        "argbBackColor": {"A":FF, "R":1E, "G":90, "B":FF}
      }
    }
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "InitPlayerFeatures",
    "token": "123456789",
    "argsRef": {
      "avatar": {
        "name": "Terminator",
        "argbBackColor": {"A":FF, "R":1E, "G":90, "B":FF}
      }
    }
  }
}
```

InitPlayerLocalization

// Cette fonction est appelée une fois our déterminer la position initiale. S'il y a des
// conflits de position, elle peut être rapellée jusqu'à huit fois.

Appel type:

```
{
  "stgyCall": {
    "seq": "InitPlayerLocalization",
    "token": "123456789",
    "argsIn": {
      "availableTiles": [1,25,33,89,143] // availableTiles ∈ [0..143]
    },
    "argsRef": {
      "locX": 10, // locX et locY ∈ [1..12]
      "locY": 3,
      "azimuth": "South" // azimuth ∈ ["North","East","South","West"]
    }
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "InitPlayerLocalization",
    "token": "123456789",
    "argsRef": {
      "locX": 1,
      "locY": 2,
      "azimuth": "West"
    }
  }
}
```

InitStatusOfCompetitors

// Reçoit la liste des concurrents: un identifiant de #1 à #8, leur identité,
// leur position et orientation de départ. La liste peut contenir de 2 à 8 descriptions.

Appel type:

```
{
  "stgyCall": {
    "seq": "InitStatusOfCompetitors",
    "token": "123456789",
    "argsIn": {
      "competitors": [
        {
          "mcndId": "01",
          "mcndLocX": 10,
          "mcndLocY": 7,
          "azimuth": "South",
          "energy": 7
        },
        { . . . }
      ]
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "InitStatusOfCompetitors",
    "token": "123456789",
    "argsRef": null
  }
}
```

InitLocateArtefacts

// Reçoit la position des dalles sur lesquelles se trouvent chacun des artéfacts
// (les trois lois de la robotique) qu'il faut atteindre pour terminer le jeu.

Appel type:

```
{
  "stgyCall": {
    "seq": "InitLocateArtefacts",
    "token": "123456789",
    "argsIn": {
      "artefacts": [
        { "locX": 11, "locY": 8 },
        { "locX": 5, "locY": 7 },
        { "locX": 8, "locY": 9 }
      ]
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "InitLocateArtefacts",
    "token": "123456789",
    "argsRef": null
  }
}
```


QuestStart

// C'est parti! C'est à ce moment que la partie débute ...

Appel type:

```
{
  "stgyCall": {
    "seq": "QuestStart",
    "token": "123456789",
    "argsIn": null,
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "QuestStart",
    "token": "123456789",
    "argsRef": null
  }
}
```

RoundRequireCartouche

// Le joueur doit retourner un cartouche composé de zéro à cinq mouvements

// qu'il souhaite voir se réaliser.

Appel type:

```
{
  "stgyCall": {
    "seq": "RoundRequireCartouche",
    "token": "123456789",
    "argsIn": null,
    "argsRef": {
      "cartouche": []
    }
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "RoundRequireCartouche",
    "token": "123456789",
    "argsRef": {
      "cartouche": [
        [
          "0x245A, Move_90Right",
          "0x2100, Move_Ahead"
        ],
        [
          "0x245C, Move_90Left",
          "0x2100, Move_Ahead"
        ],
        [
          "0x8822, Enrichment_Standby"
        ],
        [],
        []
      ]
    }
  }
}
```

RoundRobotMovement

// Les mouvements (des robots) effectivement réalisés (classés par ordre 1,2,3,4,5,6,7,8).
// Ils peuvent être différents de ceux prévus lors de heurts, de nid de poule, de mur etc ...

Appel type:

```
{
  "stgyCall": {
    "seq": "RoundRobotMovement",
    "token": "123456789",
    "argsIn": {
      "mcnd01": ["0x245A, Move_90Right", "0x2100, Move_Ahead"],
      "mcnd02": ["0x245C, Move_90Left", "0x2100, Move_Ahead"],
      "mcnd03": ["0x8822, Enrichment_Standby"],
      ...
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "RoundRobotMovement",
    "token": "123456789",
    "argsRef": null
  }
}
```

RoundBoardMovement

// 1^{er} appel: Les mouvements (des robots) provoqués par les réactions du plateau.
// 2^{ème} appel: Les mouvements (des robots) consécutifs aux réactions du plateaus.

Appel type:

```
{
  "stgyCall": {
    "seq": "RoundBoardMovement",
    "token": "123456789",
    "argsIn": {
      "mcnd01": ["0x8220, Damage_LaserFront"],
      "mcnd02": [],
      "mcnd03": [],
      ...
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "RoundBoardMovement",
    "token": "123456789",
    "argsRef": null
  }
}
```

RoundComputedMovements [non implémenté, réservé pour le jeu physique]

// Retourne, pour chaque mécanoïd, l'ensemble des mouvements effectués.

Appel type:

```
{
  "stgyCall": {
    "seq": "RoundComputedMovements",
    "token": "123456789",
    "argsIn": {
      "mcnd01": [],
      "mcnd02": [],
      "mcnd03": [],
      "mcnd04": [],
      "mcnd05": [],
      "mcnd06": [],
      "mcnd07": [],
      "mcnd08": []
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "RoundComputedMovements",
    "token": "123456789",
    "argsRef": null
  }
}
```

RoundStatusOfCompetitors

// Reçoit la liste de l'ensemble des joueurs avec leur position, leur orientation
// et le nombre de points de capacité (PC (energie)) qu'ils leur reste.

Appel type:

```
{
  "stgyCall": {
    "seq": "RoundStatusOfCompetitors",
    "token": "123456789",
    "argsIn": {
      "competitors": [
        {
          "mcndId": "01",          // in ["01".."08"]
          "mcndLocX": 10,         // in [1..12]
          "mcndLocY": 7,         // in [1..12]
          "azimuth": "South",    // in ["North","East","South","West"]
          "energy": 6            //
        },
        { . . . }
      ]
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "RoundStatusOfCompetitors",
    "token": "123456789",
    "argsRef": null
  }
}
```

RoundBoardAction

// Reçoit la définition de l'action qui s'est produite sur le plateau.

// Cet appel est facultatif. Diffusé seulement lors des parties où la dalle action est activée.

Appel type:

```
{
  "stgyCall": {
    "seq": "RoundBoardAction",
    "token": "123456789",
    "argsIn": {
      "action": {
        "name": "BoardSwap",    // la seule action actuellement définie
        "args": "1=>0",        // du plateau boardIndex °1 vers les plateau boardIndex °0
      }
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "RoundBoardAction",
    "token": "123456789",
    "argsRef": null
  }
}
```

QuestAchieved

// Le jeu est fini pour les mécanoïds listés, les joueurs ayant achevé leur quête.

Appel type:

```
{
  "stgyCall": {
    "seq": "QuestAchieved",
    "token": "123456789",
    "argsIn": {
      "completed": [
        "01"
      ]
    },
    "argsRef": null
  }
}
```

Réponse type:

```
{
  "stgyReply": {
    "seq": "QuestAchieved",
    "token": "123456789",
    "argsRef": null
  }
}
```

GameFinalisation

// Dernier appel du jeu. Permet de libérer toutes les ressources utilisées.

// A partir de là, la communication est rompue.

Appel type:







```
{
  "stgyCall": {
    "seq": "GameFinalisation",
    "token": "123456789",
    "argsIn": null,
    "argsRef": null
  }
}
```







Réponse type:

```
{
  "stgyReply": {
    "seq": "Finalisation",
    "token": "123456789",
    "argsIn": null,
    "argsRef": null
  }
}
```

Annexe 1, Les différents types de dalles

L'ensemble de définition des dalles se décompose comme suit:

<i>exemple de rendu</i>	<i>description</i>	<i>id</i>	<i>valeur</i>
	Dalle neutre.	neutre	0x0000
	Convoyeurs linéaires (par ex. eOsE => entrée Ouest , sortie Est)	conv_eNsS conv_eEsO conv_eSsN conv_eOsE	0x0014 0x0028 0x0041 0x0082
	Convoyeur double entrée (eNOsE => entrée nord et ouest, sortie est)	conv_eNOsE	0x0092
	Convoyeurs rotatifs (par ex. eSsO => entrée Sud , sortie Ouest)	conv_eOsS conv_eOsN conv_eSsO conv_eSsE conv_eEsN conv_eEsS conv_eNsE conv_eNsO	0x0084 0x0081 0x0048 0x0042 0x0021 0x0024 0x0012 0x0018
	Tourniquets droite et gauche	pt_babord pt_tribord	0x00F0 0x000F
	Murs mur_O => mur à l'ouest mur_NO => mur au nord et à l'ouest, etc	mur_N mur_E mur_S mur_O mur_NE mur_NO mur_SE mur_SO mur_NS mur_EO mur_NEO mur_NES mur_NS0 mur_ESO mur_NESO	0x0110 0x0120 0x0140 0x0180 0x0130 0x0190 0x0160 0x01C0 0x0150 0x01A0 0x01B0 0x0170 0x01D0 0x01E0 0x01F0

	LASER, en fait mur/laser (murLaser_OE => tire d'ouest en est)	murLaser_NS murLaser_EO murLaser_SN murLaser_OE	0x0214 0x0228 0x0241 0x0282
	Pousseur	pous_babord pous_poupe pous_proue pous_tribord	0x0428 0x0414 0x0441 0x0482
	Action	action	0x0800
	Nid de poule	nid2poule	0x1001
	Presse (non utilisé)	presse	0x1002
	Réparation simple et double	clef_1rep clef_2rep	0x2001 0x2002
	Artéfacts /Lois	loi_r1 loi_r2 loi_r3	0x8001 0x8010 0x8100

Annexe 2, Sous divisions des mouvements

Un mouvement se subdivise en actions et réactions. Ci dessous on trouve par catégories:

0x10.. => signalements
0x21.. => déplacements
0x22.. 0x24.. 0x28.. => rotations
0x8... => réactions

```
None = 0x0000, // ni action ni réaction
//
Light_Off = 0x1010, // éteint sa DEL de positionnement
Light_On = 0x1011, // allume sa DEL de positionnement
//
Move_Ahead = 0x2100, // 00 move 000 north
Move_Back = 0x21B4, // B4 move 180 south
//
Move_180Right = 0x22B4, // B4+0 move 180 east
Move_180Left = 0x22B6, // B4+2
Move_90Right = 0x245A, // 5A+0 move 90 east
Move_90Left = 0x245C, // 5A+2
Move_45Right = 0x282D, // 2D+0 move 45 east
Move_45Left = 0x282F, // 2D+2
//
Hustle_McndFront = 0x8110, // heurte un mécanoïd par l'avant
Hustle_McndRight = 0x8111,
Hustle_McndRear = 0x8112,
Hustle_McndLeft = 0x8114,
//
Hustle_Wall = 0x8120, // heurte un mur (par l'avant)
//
Hustle_PusherFront = 0x8140, // se prend un pousseur (ou un mcnd) par l'avant
Hustle_PusherRight,
Hustle_PusherRear,
Hustle_PusherLeft,
//
Damage_LaserFront = 0x8220, // se reçoit le laser par l'avant
Damage_LaserRight,
Damage_LaserRear,
Damage_LaserLeft,
Damage_Press = 0x8240,
Damage_Pothole = 0x8280, // dans un nid de poule
//
Conveyor_MakeMoveFront = 0x8420, // se fait déplacer par l'avant
Conveyor_MakeMoveRight,
Conveyor_MakeMoveRear,
Conveyor_MakeMoveLeft,
Conveyor_Turn45Right = 0x8440, // se fait tourner de 45° vers la droite
Conveyor_Turn45Left,
Conveyor_Turn90Right = 0x8480, // se fait tourner de 90° vers la droite
Conveyor_Turn90Left,
//
Enrichment_EnergyX1 = 0x8811,
Enrichment_EnergyX2 = 0x8812,
Enrichment_Frozen = 0x8820,
Enrichment_Standby = 0x8822,
Enrichment_WakeUp = 0x8824,
Enrichment_LawOne = 0x8841,
Enrichment_LawTwo = 0x8842,
Enrichment_LawThree = 0x8843,
Enrichment_QuestAchieved = 0x8848,
Enrichment_Victory = 0x8880,
```