

Game Master 1.7.5

Manuel de Référence

Document de travail

Jean-Jacques Girardot
Mai 2024

*J.J. Girardot jj@girardot.name
25 Rue Pierre Bérard
42000 Saint-Etienne Cédex*

Document de travail. Release 141.

Copyright (c) J.J.Girardot, 2020, 2021, 2022, 2023, 2024.

Using :

pdfTeX 3.1415926-1.40.9-2.2 (Web2C 7.5.7)

LyX Version 2.3.0

PGF 3.1.8b

Date d'impression 19 mai 2024

Table des matières

Table des Matières	3
I Le Game Master	11
1 Introduction	13
1.1 Préambule :Le <i>Game Master</i> , les concepts	13
1.2 Nouveautés	14
1.2.1 Version 1.7.5	14
1.2.2 Version 1.7.4	14
1.2.3 Version 1.7.3	15
1.2.4 Version 1.7.2	15
1.2.5 Version 1.7.1	15
1.2.6 Version 1.7.0	15
1.2.7 Version 1.6.16 du 31/08/2023	15
1.2.8 Version 1.6.15 du 10/08/2023	15
1.2.9 Version 1.6.14 du 04/07/2023	16
1.2.10 Version 1.6.12r1 du 10/05/2023	16
1.2.11 Version 1.6.12 du 05/05/2023	16
1.2.12 Version 1.6.11 du 26/04/2023	16
1.2.13 Version 1.6.10 du 04/04/2023	16
1.2.14 Version 1.6.9 du 30/03/2023	17
1.2.15 Version 1.6.8 du 03/03/2023	17
1.2.16 Version 1.6.6 du 20/02/2023	17
1.2.17 Version 1.6.5 du 20/01/2023	17
1.2.18 Version 1.6.0 du 14/05/2022	17
1.2.19 Version 1.5.5 du 18/01/2022	18
1.2.20 Version 1.5.4b du 28/12/2021	18
1.2.21 Version 1.5.4 du 19/9/2021	18
1.2.22 Version 1.5.3 du 4/8/2021	18
1.3 Note aux β -testeurs	19
1.4 Installation	19

1.4.1	REAPER et ses extensions	19
1.4.2	Le Game Master	20
1.4.2.1	Mise en oeuvre [Juin 2023]	21
1.4.2.2	Le Game Master pour tous	21
1.4.2.3	Le Game Master intégré au projet	22
1.5	Premier contact	22
1.5.1	Mise en œuvre d'une nouvelle version	22
1.5.2	Et ensuite ?	25
1.6	Les mécanismes	26
1.6.1	Le mode de jeu	27
1.6.2	Le mode d'espace	28
1.6.3	Les partiels, banques et groupes	29
1.6.4	Les « images-de-sons »	30
1.6.5	Les algorithmes	30
1.6.6	Les ClipSets	30
1.7	L'interface utilisateur	31
1.7.1	Le plug-in	32
1.7.2	Onglets, Modules, Widgets et interactions	32
1.7.3	Les éléments graphiques	33
1.7.4	Note	35
1.8	Un exemple de mise en œuvre : mode « génératif » ou mode « live »	35
1.8.1	Préférences	35
1.8.2	Pistes	36
1.8.2.1	Master Track	36
1.8.2.2	Game Player	37
1.8.2.3	File Players	37
1.8.2.4	Génération multiphonique, écoute stéréo.	37
1.8.2.5	Unités MIDI	38
1.8.3	Utilisation du Game Master	38
1.9	Mise en œuvre de contrôleurs MIDI	39
1.10	Caveat	40
1.11	Caveat 2 : IASIAS	40
2	Modules de commande standard	43
2.1	Commandes globales	43
2.1.1	Souris	43
2.1.2	Clavier	44
2.1.3	Interactions génériques typiques	44
2.1.3.1	Cellules	44
2.1.3.2	Sliders	45
2.1.3.3	Notes	46

2.1.4	Glisser-déposer	47
2.2	Configuration globale	47
2.3	Module « Main Infos »	49
2.4	Module « System Log »	49
2.5	Modules « Parameters »	50
2.6	Module « Play Control »	51
2.7	Module « Studio Play Pad »	54
2.7.1	Commandes globales	55
2.7.2	Commandes locales	56
2.8	Module « Sound Units »	56
2.8.1	Commandes globales	57
2.8.2	Commandes locales	58
2.9	Module « Displayer »	58
2.9.1	Configuration des afficheurs	60
2.9.2	Configuration des contrôleurs	61
2.9.3	Réglages globaux	62
2.10	Module « Clips Settings »	62
2.11	Module « HP Configuration »	64
2.12	Module « Space Modes »	65
2.13	Module « Play Modes »	66
2.14	Module « Banks Definition »	67
2.15	Module « Scheduler »	68
2.16	Module « Links Manager »	69
2.16.1	Nature des sources	70
2.16.2	Natures des cibles	70
2.16.3	Trace	71
2.16.4	Utilisation de périphériques MIDI	71
2.16.5	Connexion d'un contrôleur MIDI	72
2.16.5.1	Sélectivité	73
2.16.6	Connexion d'une note MIDI	74
2.16.7	Utilisation d'un slider JSFX	74
2.16.8	Raccourcis clavier	75
2.16.9	Événement interne.	76
2.17	Module « Sensors »	76
2.18	Module « Random States »	77
2.19	Module « System Settings »	78
2.19.1	Script auxiliaire	79
2.19.2	Commande du script auxiliaire depuis le module « System Settings »	80
2.19.3	Commande du script auxiliaire depuis un script « <i>mSL</i> » ou depuis un pad...	81

2.20	Module « Script Manager »	81
2.21	Module « Memory »	82
3	Le module « Clips Selection »	85
3.1	Introduction	85
3.2	Description	86
3.3	Interactions	87
3.4	Glisser-déposer	88
4	Le playlog	91
4.1	Concepts généraux	91
4.2	Interface utilisateur	92
4.3	Commandes d'enregistrement et de rejouage	92
4.4	Réglages de position et de vitesse	94
4.4.1	Le temps virtuel	94
4.4.2	La vitesse temporelle	95
4.5	Les marqueurs	95
4.5.1	Opérations	96
4.5.2	Edition des marqueurs	98
4.6	Autres commandes	100
5	Les Paramètres Dynamiques de Jeu	101
5.1	Note sur les « grains »	101
5.2	Note sur les modes de jeu	102
5.3	Les paramètres et leurs actions	104
5.4	Le module « Dynamic Play Parameters »	106
5.4.1	Manipulation des préréglages	107
5.4.2	Randomisation	107
5.4.3	Interpolation	108
5.4.4	Visualisation et édition de préréglages	108
5.4.4.1	Utilisation	108
5.4.4.2	Récapitulatif des commandes	111
II	Le langage de script	113
6	Programmer avec mSL	115
6.1	Introduction	115
6.2	Premiers éléments	118
6.3	Premiers essais	121
6.4	Opérateurs, fonctions, expressions et instructions	126

6.5	Recherche et Tri d'éléments	128
6.6	Tableaux	133
6.7	Fonctions définies	136
6.8	Chaînes de caractères	146
6.9	Erreurs	147
6.10	Une première conclusion	147
7	Présentation détaillée du langage mSL	149
7.1	Concepts généraux	149
7.1.1	Le langage	150
7.1.1.1	Le langage, et ses différences avec <i>eel2</i>	150
7.2	Quelques caractéristiques de l'implémentation	157
7.2.1	Gestion de la mémoire	157
7.2.2	Gestion des chaînes de caractères	157
7.2.3	Compilation dynamique	157
7.2.4	Blocs	157
7.2.5	Tables des symboles	159
7.2.6	Processus	159
7.3	<i>mSL</i> , en résumé	160
8	Integration de mSL au sein du Game Master	163
8.1	Fonctions spécifiques	163
8.1.1	La procédure « action »	163
8.1.2	La procédure « table »	163
8.1.2.1	Macro commandes	165
8.1.3	Opérations « get/set »	166
8.1.4	Opération « call »	167
8.1.5	Extensions	167
8.2	Lancement d'un script	168
9	Processus	169
9.1	Introduction	169
9.1.1	REAPER	169
9.1.2	Le GameMaster	170
9.1.3	Processus légers	171
9.1.4	Le Scheduler au sein du GameMaster	172
9.1.5	Opérations sur threads : « thread »	173
9.1.6	Opérations sur événements : « event »	174

10	Jeu des clips depuis mSL	177
10.1	Scénario	177
10.2	Variables et structures de données	177
10.3	Opérations disponibles	177
10.3.1	Jeu de clips : « play »	177
10.3.2	Gestion des players : « player »	178
III	Configuration et paramétrisation	181
11	Liste des paramètres et indicateurs	183
11.1	Paramètres	183
11.2	Indicateurs	188
12	Actions	189
12.1	Liste des actions prédéfinies	189
12.2	Séquences d'actions	197
12.3	Utilisation des actions	197
13	Fichiers de configuration	199
13.1	Les listes de fichiers de configuration	199
13.2	Syntaxe de la commande « data »	200
13.3	Configuration des canaux/haut-parleurs	200
13.4	Configuration des clips	203
13.5	Configuration des ClipSets	204
13.6	Modes d'espace	208
13.7	Modes de jeu	210
13.8	Banques	213
13.9	Senseurs	216
13.9.1	Instructions	217
13.9.2	Exemples	217
13.9.3	Notes	218
13.9.4	Identification des commandes	218
13.10	Le fichier de configuration général	218
14	Utilitaires	221
14.1	Smooth Compressor	221
14.2	Multi Channel VU Meter	222
14.3	Cheap Channel Reducer	222
14.4	MIDIduino	222

15 Beta testing	223
15.1 Outils de « mise au point » disponibles	223
15.2 Modules spécialisés	224
15.2.1 Players	225
15.3 Et aussi	227
16 Installation ex-nihilo - v1.6.16	229
16.1 Projet autonome contenant le Game Master	229
16.1.1 Création du projet lui-même	229
16.1.2 Décisions, décisions...	233
16.1.3 Description de l'installation	233
16.1.4 Création des clips...	234
16.1.4.1 Création d'un répertoire spécial pour le projet . .	235
16.1.4.2 Seconde approche : le glisser-déposer	245
16.1.5 Autres fichiers de configuration	245
17 Erreurs détectées par le Game Master	249
 IV Annexes	 255
18 Glossaire	257
Récapitulatif	259
Références	259

Première partie

Le Game Master

Chapitre 1

Introduction

1.1 Préambule :Le *Game Master*, les concepts

La suite « *Game Master* » est destinée à la génération semi-automatique de séquences électroacoustiques multicanales à partir de « clips », échantillons sonores de courtes ou moyennes durées, typiquement de une seconde à quelques minutes.

On peut considérer le « *Game Master* » comme un *échantillonneur logiciel*, pouvant manipuler simultanément un grand nombre d'échantillons (jusqu'à 10000), et disposant à l'heure actuelle de deux algorithmes de jeux :

- le *jeu continu*, qui va jouer un *segment* du clip originel. Différents paramètres permettent de choisir la longueur et le début du segment, sa vitesse et son sens de lecture, ses enveloppes, etc.
- le *jeu de fragments*, qui va extraire des *fragments* de l'échantillon, et les restituer avec différentes variations. Pour des fragments longs (quelques secondes à quelques dizaines de secondes), l'algorithme s'apparente à un *looper* lorsque ces fragments sont de même taille, débutent au même endroit dans le clip, et sont synchronisés. Pour des fragments courts (de quelques millisecondes à une seconde), on utilisera le terme de *grains*, et l'on se retrouve alors dans le domaine de la *synthèse granulaire*.

Ces deux algorithmes sont dits, pour des raisons historiques, mode « *Play* » et mode « *Loop* » – c'est cette terminologie qui est utilisée dans ce document.

Sans entrer dans les débats, et en référence à un article de Pierre Couprie aisément consultable sur internet [6], on peut dire que les clips fournis au « *Game Master* » sont, idéalement, des « objets sonores fixés » au sens Schaefferien du terme, voire des « objets convenables » ou des « objets musicaux ». En s'appuyant sur des configurations de modes de fonctionnement choisies par l'utilisateur, le logiciel va transformer ces « objets sonores » en « images-de-sons » (terme dû à François Bayle), au travers de différents algorithmes de placement en espace et en temps.

La suite « *Game Master* » peut fonctionner de manière entièrement automatique, à partir de préréglages enregistrés et de fichiers de configuration, ou encore travailler en mode interactif, voir en « live ». Elle constitue un « studio » particulièrement adapté au « design sonore », qui permet d'expérimenter au travers de très nombreux réglages de paramètres.

Cette suite se compose de plug-ins de type « JSFX[3, 4] », c'est-à dire qu'elle ne va fonctionner *que* sous le logiciel « REAPER[2, 1] ». Les différentes interactions s'effectuent au travers d'un plug-in unique, le « *Game Master* » et de son interface graphique, lui-même communiquant avec d'autres plug-ins grâce à une mémoire partagée commune.

Cette première partie décrit l'installation et la mise en œuvre de la suite « *Game Master* ». Une seconde partie s'intéresse à « *mSL* », le langage de script incorporé à la suite. La troisième partie aborde en détail les spécifications internes de l'outil, permettant à un utilisateur compétent dans le langage « JS » d'effectuer des modifications du logiciel.

Le *Game Master* est le résultat de plusieurs années de développements. Il doit son existence et nombre d'idées originales à *Jean-François Minjard*, qui a accompagné ses développements depuis le début. Qu'il en soit ici tout particulièrement remercié.

1.2 Nouveautés

Les paragraphes qui suivent décrivent les évolutions récentes du logiciel. Si c'est votre premier contact avec celui-ci, vous pouvez directement sauter à la seconde partie, 1.5.1 page 22.

Pour ceux qui utilisent déjà le « *Game Master* », voici une description brève des modifications récentes du logiciel. Tous ces aspects sont mis à jour dans la documentation elle-même. Notons que la numérotation des versions du *Game Master* est constituée de trois nombres de deux chiffres, séparés par des points. La version « 1.6.4 » doit être comprise comme « 01.06.04 », et est antérieure à « 1.6.15 ».

1.2.1 Version 1.7.5

Diverses corrections dans les modes de jeu. Ajoût d'une nouvelle fonction dans les scripts « mSL » : « *gui* » (non encore documentée :) qui permet à un script de créer de nouveaux modules affichables. A noter que les dernières versions du « *Game Master* » ne sont *plus* opérationnelles sous Windows. L'enquête est ouverte.

1.2.2 Version 1.7.4

Début de la mise en place des déplacements dynamiques dans l'espace.

1.2.3 Version 1.7.3

Introduction des paramètres dynamiques, pour les modifications en cours de jeu.

1.2.4 Version 1.7.2

Disponible fin Janvier 2024, cette version introduit quelques nouveaux aspects dans la définition des modes de jeu et des modes d'espace.

1.2.5 Version 1.7.1

Disponible en Décembre 2023, cette version propose l'import direct de fichiers par « glisser-déposer » (c.f. § 2.1.4 page 47), et permet la lecture d'audio de grande taille.

1.2.6 Version 1.7.0

Disponible depuis fin Novembre 2023. Elle a consisté en une grosse réécriture des *FileReaders*, qui introduit la notion de « paramètres dynamiques » permettant d'intervenir durant le jeu même des clips, et propose un nouveau module pour la gestion de ces paramètres (c.f. le chapitre 5 page 101).

1.2.7 Version 1.6.16 du 31/08/2023

La distribution de cette version a été retardée du fait de problèmes d'implantation de JSFX dans REAPER (aujourd'hui corrigés) ne permettant pas son utilisation sur Mac à processeurs M1. Cette version contourne ces problèmes, avec quelques améliorations dans la gestion des *clipsets* (c.f. § 13.5 page 204), du « Clip Selection » et des lecteurs. Le manuel a aussi subi relectures et corrections.

1.2.8 Version 1.6.15 du 10/08/2023

- Intégration dans le manuel d'un chapitre consacré à la programmation en mSL.
- Intégration de la notion de « ClipSet », ensemble de clips.
- Quelques modifications dans la gestion du Play Log.
- Modification des File Readers, permettant d'éviter les RTxruns et drop-out du son. Les lectures de fichiers, antérieurement situées dans la section « @block » qui est une section temps réel audio, ont été déplacées dans la section « @gfx » ce qui fait que les lectures n'interfèrent plus avec la création de l'audio.

1.2.9 Version 1.6.14 du 04/07/2023

- Intégration du Play Log (c.f. 4 page 91), permettant d'enregistrer et de rejouer les séquences créées depuis le début de la session.

1.2.10 Version 1.6.12r1 du 10/05/2023

- Nouvelle version du compactage des identificateurs longs. Evite les cas (rares) où deux identificateurs longs (+ 20 caractères) étaient reconnus identiques s'ils ne différaient que d'une lettre.
- Le menu de configuration du GUI peut être activé par « shift-clic droit » ou « alt-clic droit ».

1.2.11 Version 1.6.12 du 05/05/2023

- Intégration de la nouvelle gestion de mémoire dynamique. Quelques optimisations. Demande possible de blocs alignés sur des frontières de 64k.
- Les tâches, listes et messages sont maintenant représentés par des blocs de mémoire pouvant être statiques ou dynamiques.
- Commande pour enregistrer « syslog » sous forme d'un fichier texte.

1.2.12 Version 1.6.11 du 26/04/2023

- Intégration dans mSL des accès aux événements et messages, pour la communication avec des tâches du GM ou avec d'autres scripts mSL.
- Intégration dans «get» et «set» de l'accès semi-efficace aux variables des scripts JSFX. Il suffit d'ajouter les noms des variables que l'on veut pouvoir lire et écrire dans une liste, et d'exécuter un utilitaire annexe qui va générer automatiquement le code JSFX pour l'accès à ces variables.
- Intégration au moyen de la fonction « call » de l'accès (plus simple) à des fonctions JSFX du GM d'une importance réduite, ne nécessitant pas de « gaspiller » un des rares codes machine encore libres dans l'implantation.
- Premier usage réellement pertinent d'un script mSL complexe pour la gestion de l'installation du « cube » à « Chazelles ».

1.2.13 Version 1.6.10 du 04/04/2023

- Modifications dans l'ergonomie de l'interface graphique. Nécessite une re-création des anciens pré-réglages (c.f. § 1.5.1 page 22). Les fichiers de configuration des versions antérieures restent valables.
- Réécriture partielle des modules « Clips Selection » et « System Settings ».

- Intégration du concept de « *ClipSet* » offrant une souplesse accrue dans le choix des clips.
- Ajout de la commande « `PlFrSet` » (numéros 58/357) pour jouer un clip appartenant à un set.
- Mise à jour (partielle) de la documentation.

1.2.14 Version 1.6.9 du 30/03/2023

- Intégration de la nouvelle version des menus.

1.2.15 Version 1.6.8 du 03/03/2023

- Prise en compte du « glisser-déposer » pour les fichiers de configuration. C.f. § 2.1.4 page 47.
- Modifications dans l’ergonomie de l’interface graphique.
- Nouvelles actions GM.
- Corrections de problèmes d’affichage sur écrans « retina ».
- Diverses corrections internes.
- Vérification portage sous Windows.

1.2.16 Version 1.6.6 du 20/02/2023

- Nécessite REAPER 6.75 ou ultérieure.
- Gestion du mode « retina » pour affichage sur Mac OSX.
- Version préliminaire et expérimentale (0.1.2) d’un script eel2, permettant au Game Master d’envoyer des commandes à Reaper. C.f. § 2.19.1 page 79.

1.2.17 Version 1.6.5 du 20/01/2023

- Diverses corrections pour le portage sous Windows. Des tests plus complets restent à effectuer.

1.2.18 Version 1.6.0 du 14/05/2022

- Première version du « play log », qui introduit la notion de « son figé » (un clip, avec l’ensemble des réglages afférents à son jeu), l’enregistrement de ces « sons », et la possibilité de rejouer une partie de la session depuis un moment précis. Version expérimentale servant de « proof of concept ».

1.2.19 Version 1.5.5 du 18/01/2022

- Le Game Master reste entièrement opérationnel, même lorsque Reaper n'est pas en mode lecture, ou que l'interface graphique du Game Master est fermée.
- Tous les fichiers de configuration du Game Master sont désormais écrits dans son langage de script.
- La fonction « `sprintf` » accepte maintenant jusqu'à 15 arguments.

1.2.20 Version 1.5.4b du 28/12/2021

- Ensemble de petites corrections pour rendre homogènes les diverses commandes et raccourcis clavier.
- Développement des modules « Script Manager » et « Debugger ».
- Introduction de nouvelles fonctions dans le langage de script.
- Extensions de la fonction « `table()` ».

1.2.21 Version 1.5.4 du 19/9/2021

- Vitesses et sens de lecture corrigés (dans les playmodes).
- Restructuration de la gestion des commandes système.
- Possibilité de passage d'un Pad 3x3 à 2x2 dans les sensors par « 2 » ou « 3 » tapés au clavier, quand le pointeur est dans le pad des sensors.
- Nouvelles actions système (utilisables depuis les sensors) : 21/320 : lancement d'un clip du groupe courant. 22/321, 23/322 : sélection temporaire d'un mode d'espace ou d'un mode de jeu ; 24/323, 25/324 : sélection « définitive » de mode d'espace et de mode de jeu ; 35/334 : lancement différé d'une suite de commande ; 36/335 : exécution aléatoire d'une suite de commandes.
- Accès aux actions depuis « mSL » grâce à la fonction « `action()` ».
- Modification/création des tables de configuration grâce à la fonction « `table()` » (c.f. § 13 page 199).
- Attribution d'un nom aux actions, qui peut être utilisé dans les commandes.
- Introduction de macro commandes.
- Le nombre de « pads » MIDI dans les capteurs passe de 8 à 20.

1.2.22 Version 1.5.3 du 4/8/2021

- Révision du fonctionnement des players.
- Restructuration de la gestion des actions dans le dispatcher.
- Intégration quasi complète du langage d'extension « *mSL* ».
- Introduction des fonctions « `wait()` », « `yield()` », « `exit()` ».
- Ajout des modules « *Script Manager* » et « *Debugger* ».

- Module «*Links Manager*» pour la gestion des raccourcis clavier, liens MIDI, etc.

1.3 Note aux β ετα-testeurs

Si vous lisez ce texte, c'est que vous êtes devenu, par volonté délibérée, copinage, simple accident, voire à l'insu de votre plein gré, β ετα-testeur. Si vous utilisez cette version, je suis intéressé par toutes vos remarques, suggestions, plaintes, etc. Lisez le manuel, nombre de réponses y existent.

Perspectives d'évolution

J'envisage, dans les mois à venir, de travailler sur les aspects suivants (présentés en ordre accidentel) :

- Redesign des diverses API (Application Programming Interface), création d'une API pour l'historique.
- Création d'une API pour la gestion de l'interface graphique.
- Editeur de séquences de sons.
- Refonte des FileReader, intégration de la lecture des longs fichiers.
- Redesign du play pad et intégration de quelques surfaces MIDI pour la commande du GameMaster.
- Exemples de sessions. Exemples de scripts.

1.4 Installation

1.4.1 REAPER et ses extensions

Il est bien entendu nécessaire de disposer d'une version de REAPER, que l'on travaille sur Mac, ou, hélas, Windows ou Linux ¹.

Si REAPER n'est pas installé sur votre machine, se rendre sur le site de distribution du logiciel («<http://reaper.fm/>»), cliquer sur «**DOWNLOAD REAPER**» en haut à gauche, et choisir la version de REAPER appropriée à votre machine/opérating system. Une fois le fichier d'installation téléchargé (un «**.exe**» sous Windows, un «**.dmg**» sous Mac, etc.), le double-cliquer pour démarrer l'installation. Ceci fait, installer également les extensions au logiciel, à récupérer depuis le site «<https://www.sws-extension.org/>», en choisissant la version cohérente avec le REAPER que vous avez téléchargé. Installer cette extension

1. J'entends par là que, travaillant et développant pour ma part sur Macintosh, je ne vous serai pas d'un grand secours si vous avez des soucis lorsque vous utilisez l'un de ces autres systèmes...

comme vous l’avez fait pour REAPER. Il est possible enfin de charger des « packs de langues », le pack français permettant de « franciser » son installation. Je ne suis pas fanatique de cette opération, car les traductions sont parfois approximatives, certaines sont manquantes, et c’est se couper de toute la littérature (manuels, vidéos) consacrée à REAPER, qui est, elle, en anglais. Notons que la version actuelle du *Game Master* nécessite la version v7.15 de REAPER, ou plus récent.

Vérifier le fonctionnement de votre installation, en lançant REAPER selon la procédure habituelle sur votre machine. Profitez de cette première ouverture du logiciel pour régler les paramètres relatifs à votre carte son. Choisissez le menu « REAPER ▸ Préférences... », puis, dans la fenêtre qui s’ouvre, « Audio ▸ Device ». Sélectionnez la carte son que vous désirez utiliser, puis régler « Request sample rate » sur 48000, et « Request block size » sur 4096. Cliquez « OK » pour valider vos choix.

Vous pouvez éventuellement régler d’autres préférences. Voici celles que je conseillerais :

- Il est pratique de visualiser la piste « *Master* ». Choisir le menu « View ▸ Master Track » pour cocher cette option.
- On n’a pas besoin, pour utiliser le Game Master, de la vue du *Mixer*, ni de celle du *Docker*. Choisir les menus « View ▸ Mixer » et « View ▸ Docker » pour décocher ces options.

1.4.2 Le Game Master

Caveat Le contenu de la distribution évoluant au fil du temps, la description qui suit est probablement, en partie, incorrecte. Se fier de préférence au « README » associé à la version.

En bref L’installation utilise :

- un répertoire, « GameMaster », qui contient les plug-ins « GamePlayer.jsfx » (l’essentiel des algorithmes), « FilePlayer.jsfx » (lecture et rendu des fichiers sons), « SmoothMultiLimiter » (un limiteur multicanaux), quelques plus-ins auxiliaires, un sous-répertoire, « GM-Lib », contenant tous les fichiers d’inclusion nécessaires à leur fonctionnement, et un autre sous-répertoire, « Scripts », où se trouvent les scripts utilisés par un projet spécifique.
- un répertoire, « WAVES », qui contient tous les sons propres à un projet spécifique.

Une « installation » (qu’elle soit destinée à un travail de composition, un projet live, un concert, etc.) consiste en un projet REAPER, dont l’organisation répond à une structure très précise, largement décrite dans ce document. Selon les besoins, les répertoires « GameMaster » et « WAVES » peuvent être placés à l’intérieur de ce projet REAPER, ce qui rend celui-ci autonome, et portable d’une machine à une

autre, soit encore dans le répertoire des ressources de REAPER. Le Game Master est alors accessible à tout projet créé par l'utilisateur, sans qu'il soit nécessaire de copier son code à l'intérieur du projet. Notons que, compte tenu de la gestion des plug-ins de REAPER, ces deux modes de fonctionnement sont incompatibles.

1.4.2.1 Mise en oeuvre [Juin 2023]

Décompresser la version de distribution du « **Game Master** ». Vous obtenez un répertoire du même nom, dans lequel se trouvent (au moins) cinq fichiers ou sous-répertoires :

GameMaster répertoire qui contient le plugin lui-même et quelques fichiers et répertoires auxiliaires. Il s'agit de l'ensemble des codes sources du logiciel, qui sont recompilés par REAPER lors de chaque lancement d'un projet utilisant le « GameMaster ».

GMAuxiliary.eel qui est le script auxiliaire, utilisé par le Game Master pour transmettre des commandes à REAPER.

GameMaster-Doc-1.7.x.pdf qui est la documentation de la version courante.

GM-Octo qui est un projet REAPER, qui vous permettra de tester le « **Game Master** ».

WAVES qui est un répertoire contenant des exemples de clips, utilisés par la session « GM-Octo ».

GM-Octo et WAVES constituent un simple exemple d'utilisation du Game Master, et ne sont pas indispensables au fonctionnement de ce dernier. Ignorez, mais ne supprimez pas les autres fichiers...

Il existe deux modes d'installation possibles pour le GameMaster : le rendre accessible à tous les utilisateurs de la machine, ce qui implique d'avoir accès aux répertoires système de la machine, ou l'utiliser dans un projet unique. Ces deux modes d'installation sont détaillés ci-dessous.

1.4.2.2 Le Game Master pour tous

Une fois le logiciel REAPER installé, lancer son exécution, et chercher (au moyen du menu « Options ▸ Show REAPER ressources path... ») le répertoire des ressources de REAPER. Un répertoire s'ouvre dans le Finder/Explorer, dans lequel se trouvent différents sous-répertoires, dont trois de noms « *Data* », « *Effects* » et « *Scripts* ».

1. (Dé)Placer dans le répertoire « *Effects* » le répertoire « *GameMaster* ».
2. (Dé)Placer dans le répertoire « *Data* » le répertoire de nom « *WAVES* ».
3. (Dé)Placer dans le répertoire « *Scripts* » le fichier « *GMAuxiliary.eel* ».

4. Le répertoire du projet « *GM-Octo* » peut lui-même être placé n'importe où, par exemple sur le bureau ou dans votre répertoire d'accueil.
5. Pour lancer le projet, double-cliquez sur le fichier « *GM-Octo.RPP* »

Cette procédure permet l'installation du Game Master pour tous les utilisateurs de la machine.

1.4.2.3 Le Game Master intégré au projet

Il est possible de placer le Game Master dans un *projet spécifique*. Il ne sera alors accessible que par ce projet, mais cette approche permet de ne pas modifier l'installation de REAPER sur la machine. Elle est pertinente si vous n'avez pas accès aux répertoires du système.

1. Installer le répertoire « *GM-Octo* » dans un endroit accessible, par exemple sur le bureau ou dans votre répertoire d'accueil.
2. Créer dans le répertoire « *GM-Octo* » un nouveau répertoire, de nom « *Effects* », et y placer le répertoire de nom « *GameMaster* ».
3. Créer dans le répertoire « *GM-Octo* » un nouveau répertoire, de nom « *Data* », et y placer le répertoire de nom « *WAVES* ».
4. Créer dans le répertoire « *GM-Octo* » un nouveau répertoire, de nom « *Scripts* », et y placer le fichier « *GMAuxiliary.eel* ».
5. Pour lancer le projet, double-cliquez sur le fichier « *GM-Octo.RPP* ».

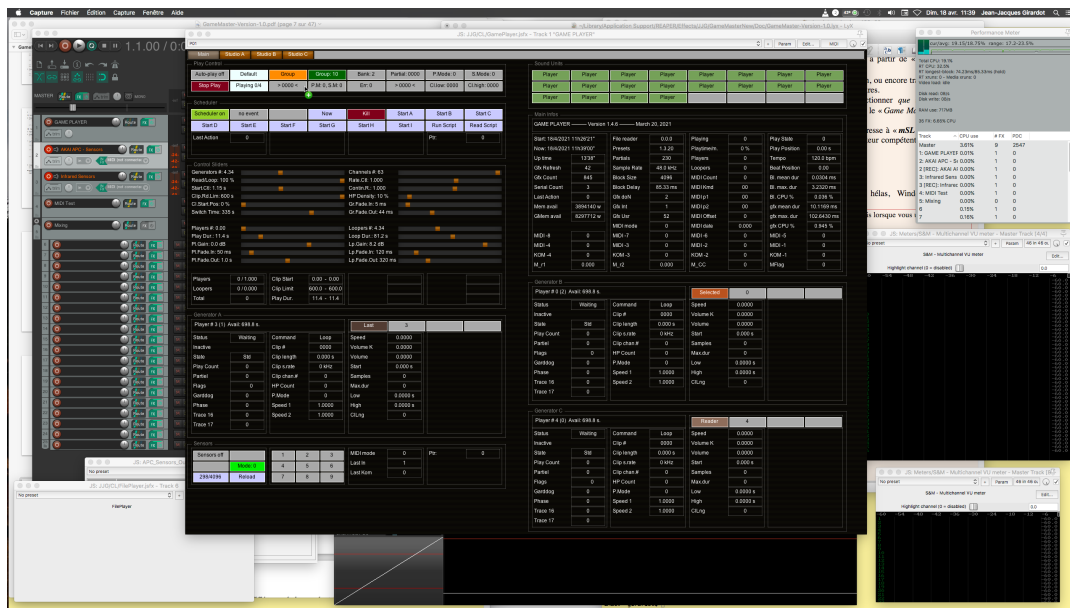
1.5 Premier contact

Double-cliquez sur le fichier `GM-Octo.RPP`. Cette action lance REAPER, qui charge le projet, et affiche quelque chose de similaire à la figure 1.1 page suivante, ou comme 1.2 page 24 si le Game Master n'a pas été configuré...

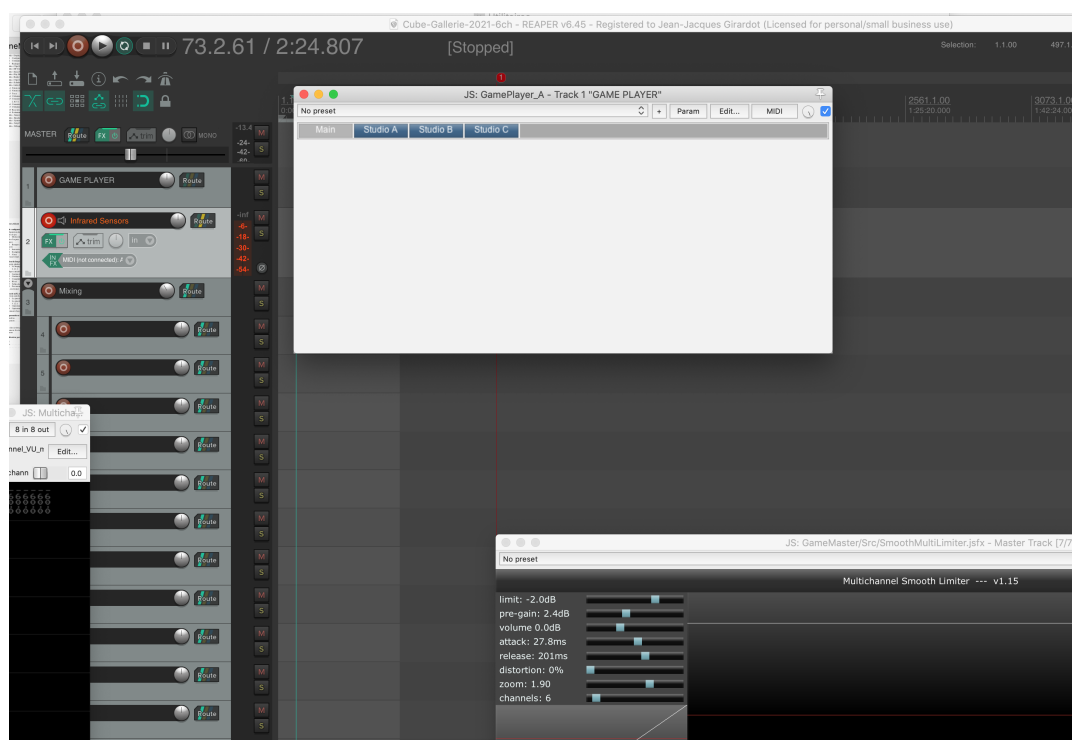
Si vous êtes dans ce dernier cas, il est bon de lire les paragraphes suivants, qui détaillent les différents aspects du logiciel. Sinon, tentez le bouton « Auto-play » du module « Play Control » en haut à gauche de la fenêtre du Game Master...

1.5.1 Mise en œuvre d'une nouvelle version

Le travail décrit ici est à faire une seule fois, lors d'une nouvelle installation du GameMaster sur votre machine. La raison en est qu'il faut créer un fichier initial de préreglages adaptés à votre machine, pour le GameMaster lorsqu'il est intégré à un projet. Voici l'algorithme à utiliser à partir de la session de démonstration, de nom « *GM-Octo* ».

FIGURE 1.1 – Une session *Game Master*

1. Si des versions antérieures du GM avaient été installées, il peut être nécessaire de supprimer les anciens *préréglages*, lorsqu'ils ne sont plus compatibles, voire de supprimer le fichier de sauvegarde des préréglages du GM. Pour ce faire, aller dans le « rrR » (répertoire des ressources de REAPER), descendre dans « presets », et supprimer (ou déplacer, ou renommer) les fichiers de noms « js-GameMaster_GamePlayer_jsfx.ini » ou (si le GM était intégré à un projet) « js-__Project__GamePlayer_jsfx.ini ». Supprimer également, ou mettre de côté, la version antérieure du Game Master (située dans « <rrR>/Effects », probablement sous le nom « Game-Master » ou équivalent).
2. Double cliquer sur le fichier « GM-Octo.RPP ». Ceci lance la session, et affiche la fenêtre du « GameMaster ». Si cette fenêtre ne s'affiche pas immédiatement, cliquer sur la boîte (verte) « FX » de la piste « 1 », de nom « GAME PLAYER ».
3. Si la fenêtre affichée n'est pas entièrement vierge, il se peut qu'elle ait déjà été configurée, et la session sauvegardée, avant la création de la distribution. Elle comporte alors vraisemblablement un cadre, de nom « System Log », qui trace les différentes étapes de l'initialisation du logiciel, jusqu'à affichage du message « ** GAME MASTER READY ** ». Si vous avez déjà configuré votre installation de REAPER il suffit alors de repérer dans le cadre « Play Control » le bouton « Auto-play (off) » et de le cliquer pour lancer le mode

FIGURE 1.2 – Le *Game Master* non configuré

« auto-play », qui doit produire, au bout de quelques instants, une séquence sonore. Si tout vous semble correct, passer à l'étape 9 pour sauvegarder ce preset.

4. Si rien ne marche à cette étape 2, quoi que la fenêtre affiche, choisir dans le menu des pré-réglages (ligne tout en haut de la fenêtre du « GameMaster ») « Reset to factory default ». Le menu affiche alors « No preset » à cet emplacement, le reste du plug-in étant vierge à l'exception des tabulations « Main », « Studio A », « Studio B », etc. Cliquer « Main ».
5. Placer le curseur de la souris dans la partie gauche de la zone vierge du plug-in, et faire un « shift-clic-droit ». Un nouveau menu s'affiche, qui permet de choisir les modules (c.f. 1.5 page 33). Afficher « Modules▷ Play Control » (avec lequel on lancera les players), « Modules▷ Sound Units » (qui va permettre de suivre l'activation des players) et « Modules▷ System Settings » (qui servira à l'initialisation). Choisir également « Panes Arrangement▷ 2 panes »
6. Placer le curseur de la souris dans la partie droite de la fenêtre du plug-in, et afficher « Modules▷ System-Log », grâce auquel on pourra suivre la trace des opérations du plug-in.

7. Appuyer sur la touche « ctrl » (Mac) ou « Windows » (Windows). Le bouton « Refresh » du module « System Settings » passe à l'orange en affichant « Reset ». Le cliquer, relâcher la touche. Cliquer ensuite sur « Refresh ». Ceci lance une réinitialisation des configurations, et diverses traces s'affichent dans le « System Log ». Quand la dernière trace s'est affichée (« ...GM_mSL_ini... » et « ** GAME MASTER READY ** »), on peut essayer de cliquer le bouton « Auto-play (off) » du module « Play Control ». Si tout s'est bien passé, le « GameMaster » doit produire des sons – peut-être au bout d'une ou deux minutes, le temps que l'exploration de tous les clips se termine).
8. Il peut être pertinent, dans la mesure où la session proposée est en octophonie, d'indiquer que l'on va utiliser 8 canaux en sortie. Pour ceci, afficher dans l'un des panneaux le module « Modules▷Parameters A », puis dans ce module, par un « clic-droit » sur le texte « (unset entry) », ajouter le slider « Channels # », qui est par défaut réglé sur « 2 ». Le passer à « 8 » (ou au nombre de HP de votre configuration). Naturellement, le nombre de canaux dans les différentes pistes doit correspondre : pour « MASTER », « Mixing », « Player » (toutes ! – il faut, pour voir ces pistes, cliquer, dans la piste « 4 », « Mixing », sur le petit triangle noir, en haut, à gauche) ou « Phone Mix », sélectionner la boîte [Route], et vérifier que le réglage « Track channels » est bien réglé à 8. On notera cependant que passer le réglage « Channels # » à « 2 » permet d'obtenir une réduction en stéréophonie de la sortie, tous les canaux étant alors repliés sur les deux premiers lors du lancement de nouveaux clips.
9. Sauvegarder le préréglage du « GamePlayer » (touche [+] tout en haut de la fenêtre, à gauche de [Param]), choisir dans le menu « Save Preset... », et introduire le nom de votre choix dans la fenêtre qui s'affiche. Faire « Ok ».
10. De même on peut régler le plugin « *SmoothMultichannelLimiter* » (effet sur la piste « 6 », « Mixing ») sur 8 canaux (slider « channels »). Sauvegarder ensuite le préréglage pour ce plugin ([+], etc...)
11. Sauvegarder la session (« File▷Save Project ») et quitter REAPER.

La session sera automatiquement relancée au prochain lancement de REAPER, avec le plug-in du GameMaster correctement configuré.

1.5.2 Et ensuite ?

Dès lors, si tout marche bien, on peut décider d'installer le « GameMaster » pour tous, afin qu'il soit utilisable depuis tout nouveau projet. Tout d'abord, faire une sauvegarde du projet « GM-Project », par exemple, en en faisant un zip. Ensuite, suivre les directives de la Doc § 1.3.2.1. En pratique, les 5 éléments d'une distribution du « GameMaster », telle qu'expliquée en Doc § 1.3.2, vont s'obtenir et se mettre en place de la manière suivante :

GameMaster changer dans « GM-Octo » le nom « Effects » en « GameMaster », et le déplacer hors du projet « GM-Project » pour l'installer, comme expliqué dans le répertoire des ressources de REAPER.

GMAuxiliary.eel ce fichier est situé à l'intérieur du répertoire « GM-Project/ReaScripts ». Le déplacer selon les directives de Doc § 1.3.2.1. On alors supprimer du projet « GM-Octo » le dossier « ReaScripts » maintenant vide.

GameMaster-Doc-1.6.pdf Elle se trouve au premier niveau de la distribution. Ce document peut être placé n'importe où.

GM-Octo est le répertoire « GM-Octo ».

WAVES ce fichier se trouve dans « GM-Octo/Data ». Le déplacer de même. On peut alors supprimer du projet « GM-Octo » le dossier « Data » maintenant vide.

On a donc installé le « GameMaster pour tous », selon la procédure expliquée en Doc § 1.3.2.2.

Il est dès lors possible de relancer le projet, en double cliquant à nouveau sur « GM-Octo.RPP ». Selon les systèmes, il se peut que la configuration soit encore opérationnelle, ou ne soit plus. Si nous sommes dans ce dernier cas, il faut :

- remplacer dans toutes les pistes (« MASTER », « Mixing », « Player » (toutes !), « Phone Mix », etc.) tous les plug-ins « manquants » (car ils ont été déplacés, et ne sont plus dans le répertoire du projet, que REAPER appelle « <Project> »). Ainsi, « JS : <Project>/GamePlayer.jsfx » doit être remplacé par « GamePlayer.jsfx ». Cliquer pour chaque piste sur la boîte [FX], et, pour chaque plug-in intitulé « JS : <Project>/toto.jsfx », faire « Add » et sélectionner le plug-in correspondant, de nom « JS : toto.jsfx », le déplacer immédiatement sous « JS : <Project>/toto.jsfx », puis faire « Remove » de ce dernier.
- sauvegarder le projet.
- effectuer à nouveau la procédure décrite au §2 ci-dessus. A nouveau, la raison en est qu'il faut créer un fichier initial de préréglages adaptés à votre machine, pour le GameMaster lorsqu'il est installé pour tous les utilisateurs.

Là encore, ce travail ne nécessite d'être fait qu'une fois, et le GameMaster pourra dès lors être utilisé dans tout nouveau projet, comme expliqué dans Doc § 1.7.

1.6 Les mécanismes

Le « *Game Master* » permet donc la génération semi-automatique de séquences électroacoustiques multicanales à partir de « clips ». Les clips sont eux-même des séquences audio courtes (de une seconde à quelques minutes), représentées dans des formats acceptés par le logiciel REAPER (wave, aiff, flac, mp3, etc.), typiquement stéréo, mais aussi multicanales. Le logiciel va jouer ces séquences en leur appliquant

des transformations simples (changements de vitesse, de volume, extraction de sous-séquences et de grains), et envoyer les sons ainsi générés sur certains des canaux audio de l'installation.

Ce travail est pris en charge par un premier plug-in, le « *Game Master* » proprement dit, qui gère l'interface utilisateur et les différents algorithmes, et plusieurs plug-in auxiliaires, les « *Players* », dont le rôle est de « jouer » les clips. Très précisément, chaque player joue, à un moment donné, un seul clip à la fois, et c'est le nombre de players installés dans le projet qui détermine la polyphonie effective de l'ensemble. Selon la puissance de la CPU disponible, il est possible de jouer de 10 à 20 clips simultanément, en sachant que les players inactifs utilisent relativement peu de CPU.

Les clips sont donc des fichiers audio, conservés dans un répertoire unique, et leur nom doit respecter une syntaxe spécifique : préfixe « clip1 », suivi de 4 chiffres correspondant à un numéro de clip (de 0 à 9999), et enfin un suffixe indiquant la représentation du clip (« .wav », « .mp3 », etc.). On obtient ainsi, par exemple, « clip10132.flac » pour désigner le clip 132, qui se trouve être représenté au format « flac ». Le système accepte tous les types de fichiers audio reconnus par REAPER, sans limitation sur la fréquence d'échantillonnage ou le nombre de canaux. On peut légitimement trouver fastidieux de se plier à cette obligation, mais nous verrons qu'un certain nombre d'outils existent, qui permettent d'alléger sensiblement ce travail.

Le fonctionnement du « *Game Master* » est régulé par un grand nombre de paramètres, plusieurs centaines. Il serait fastidieux de régler chacun de ces paramètres pour chacun des clips que l'on désire jouer. On va donc rassembler des ensembles de paramètres qui s'appliquent peu ou prou à un même aspect du logiciel, puis indiquer que tel ou tel clip est géré par tel ou tel ensemble.

Les aspects retenus sont les suivants : le « mode de jeu », qui définit quelles transformations vont s'appliquer aux clips, le « mode d'espace », qui définit de quelle manière les séquences audio vont être restituées sur les canaux audio de l'installation, et enfin le « cadre algorithmique » qui va définir de quelle manière, et à quels moments, s'effectuent les choix de jeux des clips.

1.6.1 Le mode de jeu

Le mode de jeu définit de quelle manière un clip va être joué (ou « interprété ») par un player. Les players proposent deux types de jeu : le jeu séquentiel (« *play* »), et le jeu granulaire (« *loop* »).

Pour le jeu séquentiel, les paramètres suivants entrent en compte :

1. La vitesse de lecture du clip. Le player se comporte à la manière d'un magnétophone : une vitesse plus lente correspond à un décalage des fréquences

vers le grave et une durée de jeu plus longue ; une vitesse plus élevée correspond à un décalage des fréquences vers l'aigu, et une durée de jeu plus courte. On notera que la vitesse de lecture peut être négative, le clip étant ainsi lu à l'envers. Les vitesses de lecture sont typiquement comprises entre 0.1 et 4, 1 correspondant à une lecture normale.

2. La durée de jeu, qui peut être choisie de manière quasi-indépendante de celle du clip. On peut donc ne lire qu'une partie du clip.
3. Le point de départ de lecture du clip, qui n'est pas forcément le début de celui-ci.
4. Le volume du jeu, et les variations éventuelles de celui-ci.
5. L'utilisation (et la nature) de fade-in et fade-out lors de la lecture du clip.

Pour le jeu « granulaire », dans lequel sont lus des segments extraits du clip, interviennent en outre :

1. La durée des segments lus, de quelques millisecondes à quelques secondes.
2. La densité de jeu, c'est-à-dire le nombre de segments joués simultanément.
3. La position dans le clip du début de chaque segment.
4. L'utilisation (et la nature) de fade-in et fade-out pour ces segments.

Un mode de jeu ne va pas nécessairement définir une valeur fixe pour ces paramètres, mais des intervalles de variation, qui peuvent être absolus ou relatifs. Ainsi, une vitesse de lecture peut être décrite comme étant à choisir dans l'intervalle [0.6 1.1].

Le choix de l'ensemble de ces paramètres constitue donc *un* mode de jeu, et il est possible de définir des dizaines ou des centaines de mode de jeu différents. Ces modes de jeu se décrivent dans des *scripts*, fichiers externes au format texte, dans une syntaxe relativement simple, qui est exposée au paragraphe 13.7 page 210.

1.6.2 Le mode d'espace

Le mode d'espace définit vers quels canaux audio de l'installation vont être dirigés les sons produits par un player. Une installation peut elle-même comporter une quantité très variable de haut-parleurs (de deux à une centaine), et il ferait peu de sens de dire, dans l'absolu, que tel son va être dirigé vers le canal 11, par exemple. Nous allons donc définir des « espaces conceptuels », qui seront « plaqués » sur l'installation disponible. Ces espaces conceptuels vont être, par exemple :

- un haut-parleur choisi au hasard ;
- un groupe de haut-parleurs proches les uns des autres ;
- tous les haut-parleurs de l'arrière ;
- un des haut-parleurs du plafond ;
- tous les hauts-parleurs disponibles, etc.

Le mode d'espace va ainsi permettre de choisir un type de groupe, puis d'indiquer quel pourcentage de haut-parleurs va être utilisé, et de quelle manière le *player* va utiliser ces haut-parleurs (simultanément, en séquence, en mono ou multiphonie, etc.). Ces modes d'espace se décrivent dans des scripts, fichiers externes au format texte, dans une syntaxe relativement simple, qui est exposée au paragraphe 13.6 page 208.

Pour chaque installation spécifique, un fichier doit donc être défini, décrivant les associations entre les numéros de canaux et les positions physiques des haut-parleurs.

1.6.3 Les partiels, banques et groupes

Dans une finalité compositionnelle, il semble légitime de regrouper les objets sonores spécifiques choisis par un compositeur pour créer une certaine structure musicale. Nous parlerons de *groupe* pour désigner un tel regroupement.

Une fois les modes de jeu et les modes d'espace définis, il est pratique de regrouper les clips du *groupe* utilisant les mêmes modes de jeu et mode d'espace. Les « *partiels* » vont associer des ensembles de clips à un mode de jeu et un mode d'espace. Les partiels sont eux-même rassemblés dans des *banques*, et cet ensemble de banques constitue le *groupe*. Chaque partiel précise :

- le groupe auquel il appartient ;
- la banque à laquelle il appartient ;
- une séquence de clips (définie par un intervalle, par exemple, tous les clips de 100 à 122, ou encore une structure plus générale, dite « ClipSet ») ;
- un mode de jeu ;
- un mode d'espace ;
- un « poids » associé au partiel, permettant de modifier la probabilité d'utilisation du partiel ;
- une variation de volume, définie par des bornes minimale et maximale ;
- une « classe » associée aux clips du partiel, permettant une sélection de certains types de clips dans l'ensemble des partiels ;
- des indicateurs supplémentaires, autorisant ou excluant certaines transformations appliquées aux clips.

On notera que des partiels différents peuvent faire appels à des clips, des modes de jeu ou des modes d'espace identiques. Toutes ces notions, groupe, banque, mode de jeu, mode d'espace sont représentées par des nombres entiers choisis par l'utilisateur.

Ces partiels se décrivent se décrivent dans des scripts, fichiers externes au format texte, dans une syntaxe relativement simple, qui est exposée au paragraphe 13.8 page 213.

1.6.4 Les « images-de-sons »

Une « image-de-son » est le résultat du jeu d'un clip, déterminé par les paramètres associés au mode de jeu et au mode d'espace, eux-mêmes modulés par un *aléa*, lorsque ces modes de jeu et d'espaces autorisent des variations. Il est possible de conserver l'ensemble des paramètres associés à un son sous une forme relativement compacte, et ainsi de reproduire ultérieurement, à l'identique, un son ou une séquence de sons. C'est le but du *playlog*.

Par défaut, le *Play Log* est activé au début de la session. Ses différents aspects sont accessibles au travers du module « Play Log ». Il comporte un axe temporel, sur lequel sont notés les sons joués avec leur date et leurs caractéristiques. Un lecteur permet d'explorer cet axe, à des vitesses comprises entre -10 et 10 fois la vitesse « normale ». Il est également possible de « noter » à la volée des instants, et d'y revenir ultérieurement.

1.6.5 Les algorithmes

La classification en partiels, banques et groupes, et les caractéristiques auxiliaires de partiels (classe, indicateurs) vont permettre l'écriture d'algorithmes décidant des modalités de jeu des clips.

En plus des caractéristiques liées aux partiels, ces algorithmes vont pouvoir décider des moments où jouer des clips, modifier des paramètres globaux influant le volume général, la densité de jeu, la longueur des jeux individuels, etc.

Pour l'heure, il est possible :

- d'utiliser le *Game Master* en mode « live », en associant des partiels, banques, groupes, ou clips individuels à des « pads », en déclenchant ceux-ci manuellement, soit par la souris, soit en utilisant une surface de contrôle (Korg nanoPAD, Akai APCmini, Arturia BeatStep, etc.)
- de déclencher des clips, ou d'autres types d'actions, aux travers de capteurs spéciaux (par exemple, capteurs infrarouges).
- de laisser l'application choisir les clips à jouer en mode aléatoire (contrôlé par nombre de réglages). On parle alors d'un mode « génératif ».

On peut définir des algorithmes complexes (ou évolués), en utilisant « *mSL* », un « *micro script langage* » écrit pour l'occasion, et intégré au logiciel. L'ensemble de ces algorithmes repose sur plusieurs centaines de fonctions élémentaires incorporées au logiciel, et directement accessibles depuis *mSL*.

1.6.6 Les ClipSets

Un ensemble de clips peut être défini soit par un simple intervalle (tous les clips entre deux numéros, par exemple de « de 120 à 135 »), soit encore par une structure

plus complexe, dite « ClipSet ».

Un ClipSet est un ensemble de clips, définis de manière cumulative, par :

- une liste de numéros de clips
- un intervalle de numéros de clips
- une liste de clips exclus de l'ensemble précédemment décrit
- des conditions spécifiques imposées à ces clips : durée minimale, durée maximale, etc.

La description de la configuration des ClipSets est abordée plus en détail au § 13.5 page 204.

1.7 L'interface utilisateur

L'installation du plug-in sur une piste audio ouvre une interface utilisateur qui présente une apparence que l'on peut qualifier de « sobre » (c.f. fig. 1.3).

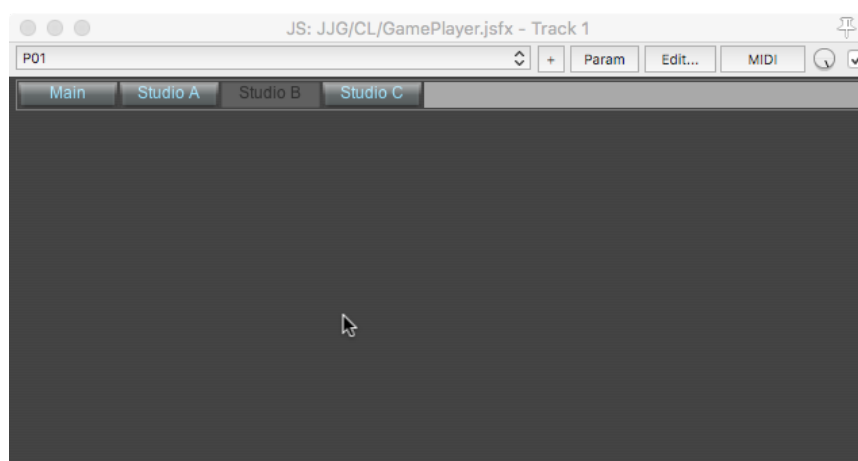


FIGURE 1.3 – L'interface du *Game Master*

Une première « barre » d'information, en haut de la fenêtre du plug-in, est gérée par REAPER, et affiche le nom du préréglage courant (ici « P01 »), ainsi que des flèches de sélection vers le haut et vers le bas, une boîte avec un « + » permettant la sauvegarde de nouveaux préréglages, et les boutons « Param », « Edit... » et « MIDI ». Un potentiomètre circulaire permet de « doser » l'effet (réglage « dry/wet »), et enfin une case, cochée, indique que l'effet est actif. On notera qu'il ne doit y avoir qu'une seule instance du plug-in « Game Master », dans la mesure où tous les plug-ins intervenant utilisent un même espace partagé. On verra cependant plus tard qu'il est possible d'utiliser, dans une même session et au prix de petites modifications, plusieurs instances du plug-in, chacune utilisant ses propres players.

1.7.1 Le plug-in

Le restant de la fenêtre concerne le plug-in proprement dit. Sur cette vue, n'apparaît qu'une barre d'onglets (rectangle horizontal tout en haut de la fenêtre), qui permet, par un clic gauche sur l'un d'eux, de sélectionner l'affichage de la page correspondante. Sur l'image, c'est l'onglet « Studio B » qui est sélectionné, désespérément vide.

La barre des onglets affiche également à sa droite un ruban coloré, qui reflète, en passant du bleu à l'orange, le pourcentage de *players* et *loopers* actifs, ou qui reste gris si aucun son n'est généré.

Un clic droit dans la barre d'onglets fait apparaître un menu (c.f. fig. 1.4) qui

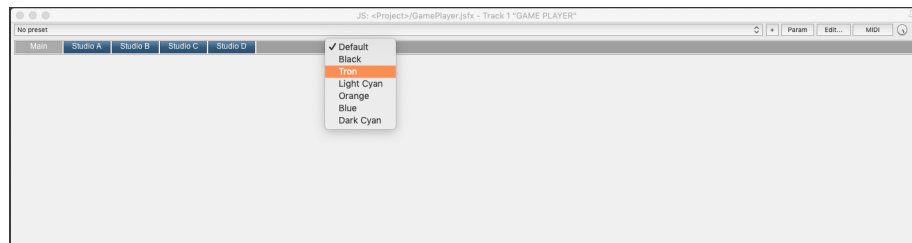


FIGURE 1.4 – L'interface : le menu de configuration de l'apparence

permet de modifier le « look and feel » du plug-in. Notons encore que la fenêtre du plug-in est redimensionnable, permettant de l'adapter aux besoins des utilisateurs et à l'espace d'affichage disponible sur l'écran.

Enfin, un clic droit dans le reste de la fenêtre avec la touche « shift » maintenue fait apparaître le menu de configuration des panneaux (c.f. fig. 1.5 page suivante). Il contient une liste des modules d'interaction que l'on peut afficher dans le panneau courant, ainsi que des commandes permettant de déplacer ou supprimer les modules. Notons que le même module peut être affiché dans différents panneaux. Ces modules sont décrits en détail au chapitre 2 page 43.

En résumé, l'interface utilisateur consiste en divers écrans, dits « *onglets* ». On passe d'un *onglet* à un autre en cliquant sur son nom dans la « barre d'onglets » située tout en haut de l'écran, ou encore par les touches *tab* ou *shift-tab*. Chaque onglet comporte un ou plusieurs *modules*. Chaque module contient lui-même un ou plusieurs éléments graphiques (sliders, afficheurs, pads, etc.), que l'on qualifiera parfois de « *widgets* », permettant les interactions avec l'utilisateur. Les *modules* et les *onglets* font l'objet d'une description spécifique dans les paragraphes suivants

1.7.2 Onglets, Modules, Widgets et interactions

Deux outils interviennent dans les interactions : la souris et le clavier. L'interface tente d'être cohérente, ergonomique et efficiente. Pour ce faire, tous les widgets, et

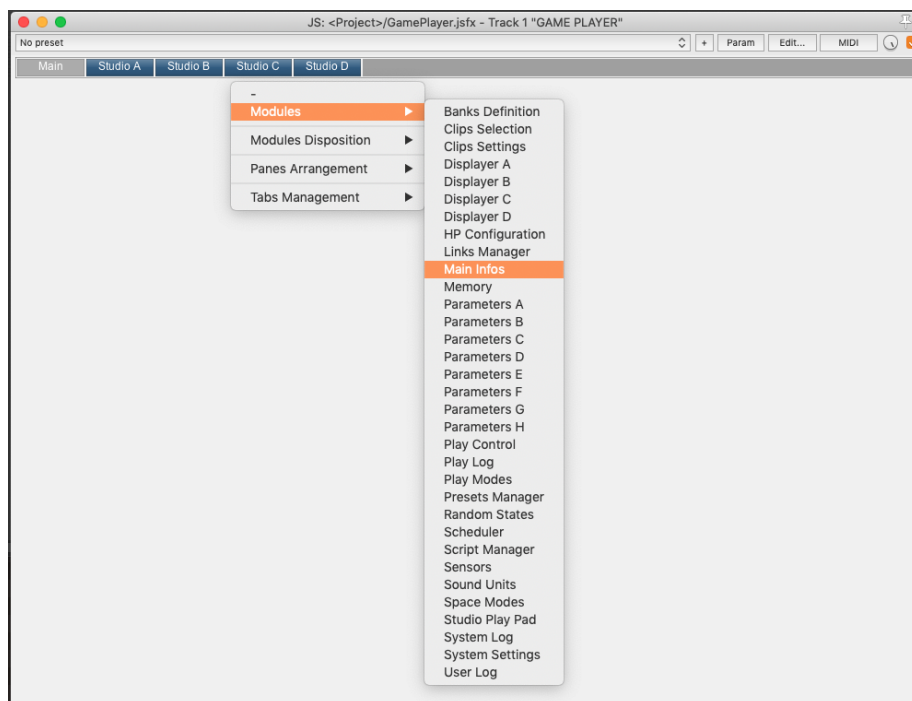


FIGURE 1.5 – L'interface : le menu de configuration des panneaux

tous les items de ces widgets, réagissent de la même manière aux mêmes interactions.

La cohérence des interactions est liée à la position de la souris, qui va être :

- dans un *item* d'un *widget* : une cellule d'un pad, ou, dans le cas d'un slider, le label, le slider lui-même, ou encore le bouton de liaison à un générateur d'aléa.
- dans le widget : pad ou slider.
- dans le module.
- dans l'onglet.

L'autre aspect de la cohérence est liée à la nature de la grandeur représentée par le *widget*, ou l'item spécifique de ce *widget*. Si une *cellule* d'un *pad* représente un numéro de clip modifiable, toutes les cellules ayant cette finalité, dans tous les *pads*, réagissent de la même façon aux mêmes commandes.

1.7.3 Les éléments graphiques

La figure 1.6 page suivante offre un exemple du plug-in configuré en deux panneaux, avec une interface « claire »². On y distingue, de haut en bas, différents modules : à gauche, « Sound Units », « Main Infos », « Studio Play Pad », « Scheduler »,

2. Non pas au sens de « compréhensible », mais au sens de « non foncée ».

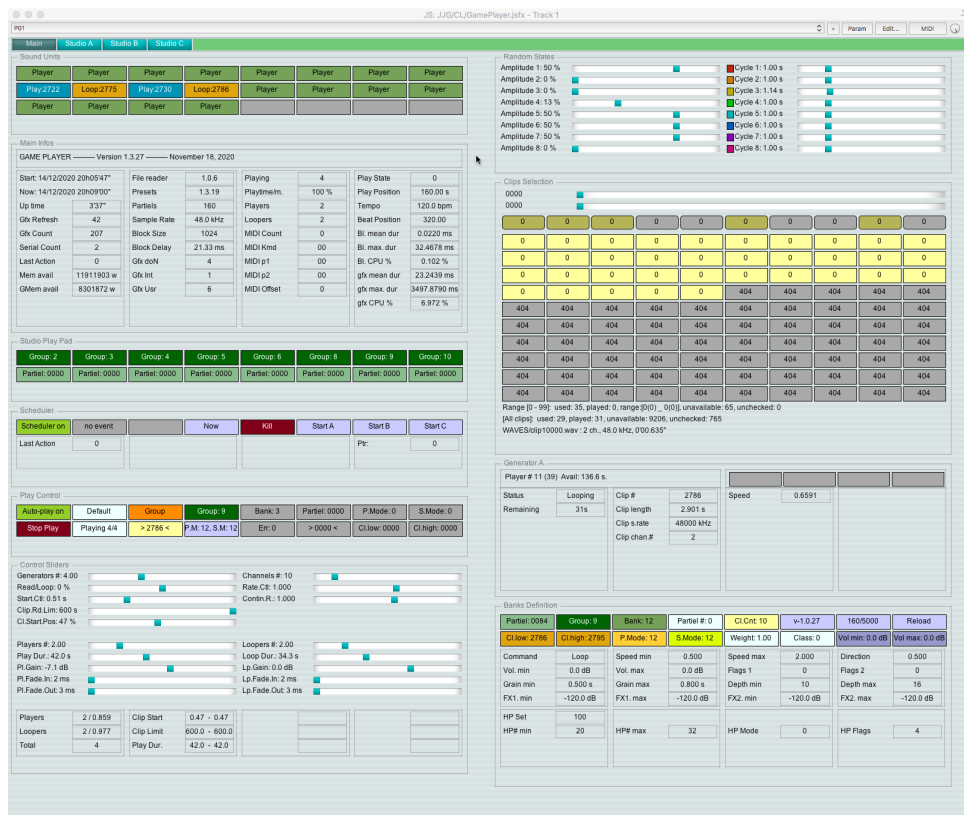


FIGURE 1.6 – Le plug-in configuré

« Play Control », « Control Sliders »; à droite « Random States », « Clips Section », « Generator A », « Banks Definition ».

Ces modules sont constitués de différents éléments graphiques :

Pads qui sont des groupes de *cellules* colorées assemblées en lignes et en colonnes (c.f. « Sound Units » ou « Studio Play Pad »), avec lesquels des interactions sont possibles grâce au clavier ou à la souris.

Sliders qui sont des réglages traditionnels (semblables à des potentiomètres linéaires horizontaux), comme dans les modules « Control Sliders » ou « Random States ». Certains sliders de ce dernier groupe comportent des boîtes juxtaposées, permettant d'activer ou de désactiver la fonction associée.

Afficheurs dont le rôle est d'afficher des valeurs et les étiquettes correspondantes (c.f. « Main Infos »).

1.7.4 Note

L'interface est basée sur le module « `ui-lib.jsfx-inc` » réalisé par Geraint Luff. Qu'il en soit ici remercié.

1.8 Un exemple de mise en œuvre : mode « génératif » ou mode « live »

Nous décrivons ici la mise en œuvre du logiciel en mode « génératif » ou mode « live », lorsqu'il est connecté à des contrôleurs MIDI, susceptibles de lui envoyer des informations de toutes natures. On suppose que les mises en place décrite au paragraphe 1.4 page 19 ont déjà été effectuées.

Il est possible sous REAPER de lier des paramètres de plug-in (dans le cas de JSFX, ils sont au nombre maximum de 64) à des messages MIDI venant de contrôleurs externes, mais nous privilégions ici l'optique d'une mise en œuvre sous la forme d'un plug-in situé dans la chaîne d'entrée audio/MIDI, qui va donc pouvoir recevoir, sans aucun filtrage, l'ensemble des informations MIDI provenant d'un périphérique donné. Cette approche nécessite un certain nombre de réglages, décrits ci-dessous.

1.8.1 Préférences

Après nombre de tests, voici les réglages de REAPER appropriés à cet usage.

Options ▸ Preferences ▸ General ▸ Maximum undo memory use A régler à « 0 ». La raison en est que, pour nombre d'opérations du plug-in, REAPER effectue une sauvegarde (afin de pouvoir faire des « undo »). Or, la taille d'un pré-réglage du Game Master est énorme, et cette sauvegarde (inutile) fait perdre un temps CPU et des E/S disque considérables, au détriment d'opérations plus utiles...

Options ▸ Preferences ▸ Audio ▸ Close audio device when stopped and application is inactive A désactiver.

Options ▸ Preferences ▸ Audio ▸ Buffering ▸ Allow live FX multiprocessing on « x » CPUs Activer. Choisir un nombre raisonnable de CPUs ou conserver le défaut qui est de 4 ou 8 selon les machines.

Options ▸ Preferences ▸ Audio ▸ Playback ▸ Run FX when stopped Activer. Ceci permet au GM de fonctionner, même quand REAPER n'est pas en lecture.

Options > **Preferences** > **Audio** > **MIDI Device** Choisir l'unité (ou les unités) qui va/vont envoyer les commandes. Activer pour chaque unité « Enable input » et « Enable input for control messages ».

1.8.2 Pistes

En partant d'une session REAPER vide, voici comment configurer les différents éléments pour obtenir une session utilisable. Il faut au préalable décider du nombre de canaux que l'on désire utiliser en sortie dans l'installation (jusqu'à 128, mais souvent moins). Ce nombre va conditionner les réglages à effectuer sur les différentes pistes. La figure 1.7 représente une première étape de ce travail.

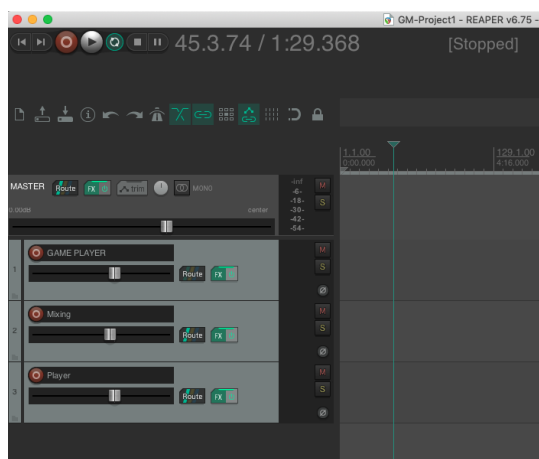


FIGURE 1.7 – Fenêtre REAPER, haut gauche.

1.8.2.1 Master Track

Dans la piste « Master » (si elle n'est pas affichée, choisir le menu « View > Master Track »), cliquer sur la boîte « Route ». Dans la fenêtre qui s'affiche, régler « Track channels » à 64 (ou moins, en fonction de la configuration de HP utilisée), et ajuster les sorties hardware « Audio Hardware Outputs » pour envoyer toutes les pistes vers les sorties voulues de la carte son.

On peut placer aussi, dans les « FX », un vu-mètre multicanal qui permet de tracer les volumes des sorties. Pour ceci, cliquer, dans la piste MASTER, sur la boîte « FX ». Le sélecteur d'effets s'affiche. Cliquer sur « Add ». Une nouvelle fenêtre s'affiche, la liste des effets disponibles. Dans la colonne de gauche, choisir « JS » dans « All Plugins ». Dans la colonne de droite, choisir « JS:Multichannel-VU-meter.jsfx ». Cliquer « Add » en bas à droite. On revient à la fenêtre précédente, où l'on note que le plugin

1.8. UN EXEMPLE DE MISE EN ŒUVRE : MODE « GÉNÉRATIF » OU MODE « LIVE » 37

est maintenant affiché dans la partie droite. Le plug-in a été inséré dans la piste, il est loisible de fermer la fenêtre.

1.8.2.2 Game Player

Créer une première piste indépendante, dite « 1 » (par exemple, par le menu « Track ▸ Insert new track »). Dans « Route », choisir le minimum de canaux (stéréo), désactiver « Master Send ». Placer le plug-in « *GamePlayer.jsfx* » dans les « FX ». On peut nommer cette piste « Game Player » en double-cliquant sur la boîte textuelle, à côté du bouton rouge.

1.8.2.3 File Players

Créer une seconde piste indépendante, dite « 2 », avec le nombre de canaux désirés (64 ou moins), qui va envoyer ses sorties dans la piste master (Dans « Route », « Master Send » coché). Nommons « Mixing » cette piste. Il n'est pas inutile d'y placer, parmi les effets, le compresseur multicanal « *SmoothMultiLimiter.jsfx* », dans lequel on règlera « channels » à la valeur appropriée.

Créer une troisième piste, avec le nombre de canaux désirés et avec envoi des sorties vers la piste parent (Dans « Route », « Parent Send » coché). Sur cette piste, placer dans les effets (« FX ») un plugin « *FilePlayer.jsfx* ». Nommons « Player » cette piste, et transformons la en une sous-piste de la piste « 2 ». Pour ceci, cliquer sur la petite icône grise située en-dessous du numéro de piste « 2 ». La piste « 3 » se retrouve alors légèrement en retrait, tandis qu'un petit triangle noir apparaît au-dessus du numéro de piste « 2 ».

Cliquer à nouveau sur « Route », puis, dans la boîte de dialogue, sur « Add new receive... ». Choisir la piste « Game Player » (c.f. 1.8.2.2), et, dans les réglages qui apparaissent, changer le menu « Post-Fader (Post-Pan) » en « Pre-Fader (Post-Fx) ».

Dupliquer cette piste « 3 » autant de fois que nécessaire, créant autant de *players*, pour obtenir la polyphonie désirée (typiquement, 8 à 20). Toutes ces pistes, étant des sous-pistes de la piste « 2 », vont donc envoyer leurs sorties vers la piste « 2 », intermédiaire, qui elle-même va envoyer vers la piste *master*.

On verra ultérieurement apparaître ces players dans le module « *Sound Units* ».

En cliquant sur le triangle noir de la piste « 2 », on peut réduire au minimum la hauteur occupée par ses sous-pistes.

1.8.2.4 Génération multiphonique, écoute stéréo.

Dans certains cas, l'on ne dispose pas, par exemple au moment d'une composition, du dispositif complet qui va être utilisé pour la restitution de l'oeuvre. Ainsi, l'on peut préparer un travail pour une pièce en 24 canaux, mais se retrouver avec 3

heures disponibles dans un train, son portable, et une paire d'écouteurs. Une manière de contourner le problème est le suivant.

Créer une nouvelle piste (que l'on placera n'importe où, par exemple entre « GAME PLAYER » et « Mixing »), à laquelle on donnera le nom de « Phone Mix ». Dans la fenêtre « Route » de cette piste, on effectuera les modifications suivantes :

1. Régler le nombre de pistes « Track channels » au nombre utilisé pour la session, par exemple 24 dans l'exemple ci-dessus.
2. Décocher l'option « Master send channels from... »
3. Cliquer sur « Add new receive... » et choisir la piste « Mixing ». Régler le nombre de canaux reçus au nombre utilisé pour la session.
4. Cliquer sur « Add new hardware output... » et choisir la sortie « headphone » de l'ordinateur. Lui associer les canaux audio 1 et 2.

Dans la fenêtre « FX » de cette piste, choisir un plug-in qui va permettre de réduire à 2 le nombre de canaux écoutés. Suivant les cas, on peut utiliser « 8xMono to 1xStereo Mixer », « ReaSurround », ou tout autre outil, tel le simplissime « CheapChannelReducer.jsfx » de la distribution.

Il est dès lors possible de travailler sur l'installation, en mode multiphonique, en mutant la piste « Phone Mix », ou encore en mode nomade stéréo, en mutant la piste « Master ».

1.8.2.5 Unités MIDI

Créer éventuellement des pistes destinées à échanger des informations avec des périphériques MIDI. Sur chaque piste, on va placer dans les « IN FX » un plug-in spécialisé destiné à recevoir des informations depuis un périphérique spécifique, et dans les « FX » de la même piste, un autre plug-in spécialisé, qui pour sa part va envoyer des commandes à ce périphérique. Ces plug-ins communiquent par l'intermédiaire de la mémoire partagée `gmem` entre les différents plug-in JSFX.

A l'heure actuelle, il existe des plug-ins destinés à communiquer avec l'*Akai APC Mini*, et avec le *Korg nanoKontrol 2*.

1.8.3 Utilisation du Game Master

Cette configuration est capable de recevoir des commandes MIDI, et d'y répondre.

A titre d'exemple, *Game Master* va jouer les touches du *PlayPad* à la réception des notes C3 à D4# sur le canal MIDI 1, venant de n'importe quel périphérique (le *Virtual MIDI Keyboard* réglé sur « MIDI channel 1 » convient parfaitement).

Notes Un plug-in (dans les « IN FX ») peut recevoir des entrées depuis un seul périphérique MIDI ou depuis tous, mais est assuré de recevoir tout ce qui est envoyé. S'il devient nécessaire d'utiliser plusieurs périphériques MIDI en entrée (optique d'un live), on peut demander à recevoir toutes les entrées MIDI. Il y a alors des réglages à faire pour éviter les interférences entre des commandes identiques de différentes unités, mais ça marche en général. Contre-exemples : l'APC mini d'AKAI, par exemple, est programmé pour envoyer tout sur le canal MIDI 1, et l'on ne peut rien y changer. Le X-touch mini de Behringer envoie sur le canal 11. Si l'on veut utiliser un second contrôleur en même temps, c'est celui-ci qu'il faut régler pour qu'il envoie sur un autre canal MIDI que le 1 :-(.

Notons que REAPER offre aujourd'hui la possibilité de transposer les messages MIDI venant d'un périphérique donné, en changeant le numéro de canal référencé dans les commandes MIDI. Il est donc possible de connecter plusieurs interfaces utilisant les mêmes canaux, à condition de transposer ceux-ci.

On peut aussi juger préférable de créer des « lecteurs MIDI spécialisés » placés chacun en entrée d'une piste dédiée, recevant et traitant les données MIDI d'un périphérique unique, puis envoyant des « événements » au *Game Master*. Ceci permet de reporter toutes les idiosyncrasies d'un périphérique donné au sein d'un seul outil spécialisé.

1.9 Mise en œuvre de contrôleurs MIDI

Il est raisonnablement simple d'utiliser des contrôleurs MIDI matériels pour interagir avec l'application. Ceci implique cependant d'écrire, pour chaque contrôleur, un couple de plug-ins simples, mais spécialisés. L'un de ces plug-ins va servir à la réception des commandes MIDI depuis le contrôleur, l'autre à l'émission, vers ce contrôleur, de commandes MIDI servant de feed-back ou de configurateur.

Soit le contrôleur « STARplus » pour lequel on a développé deux plug-ins, « *STARplus In* » et « *STARplus Out* ». Nous allons les intégrer dans le système de la manière suivante :

- S'assurer que le contrôleur est bien visible par REAPER. Dans « Options » Preferences » MIDI Device », mettre l'entrée du contrôleur en « Enabled+Control », sa sortie en « Enabled ».
- Créer une piste indépendante, avec le minimum de canaux (stéréo). Placer le plug-in « *STARplus In* » dans les « IN FX », et « *STARplus Out* » dans les « FX ». Choisir dans les entrées *Input : MIDI* » STARplus » All channels. Armer la piste en enregistrement. Dans « ROUTE », désactiver « Master Send », ajouter un « MIDI Hardware Output » vers le contrôleur « STARplus ».
- Les deux plug-ins doivent apparaître dans « *Game Master* » Units » (ou non, selon certaines options de configuration).

1.10 Caveat

La documentation est traditionnellement en retard sur l'implantation. Ce qui est décrit dans ce manuel a probablement été opérationnel à une époque. La documentation peut encore être d'actualité, ou être devenue obsolète du fait de modifications dans le logiciel non encore documentées, ou de régression du logiciel suite à son évolution. Ne pas hésiter, si l'une de ces questions se pose, à contacter l'auteur...

Notez cependant que *seule la dernière version en date du logiciel est maintenue*. Si vous avez des problèmes avec une version antérieure, passez à la dernière version, et tentez de reproduire l'erreur.

1.11 Caveat 2 : IASIAS

Ce sigle, « IASIAS », revient à l'occasion comme justificatif de certaines décisions de spécifications ou d'implémentation. Sa signification est la suivante : « I'm A Script In A Sandbox », « Je suis un script dans un bac à sable ». En effet, REAPER se protège, et protège l'utilisateur, des scripts écrits en JSFX en restreignant leurs possibilités d'interférer avec le reste du monde. Ainsi, un script ne peut pas écrire dans un fichier disque, ne peut pas connaître les fichiers disque existant dans son environnement, ne peut pas communiquer avec le système d'exploitation, etc.

Aussi, afin que le « Game Master » connaisse les scripts et fichiers disponibles, il est nécessaire de lui en fournir la liste dans des fichiers texte, dont le nom est convenu à l'avance, et dont chaque ligne contient un nom de fichier, qu'il sera dès lors possible d'accéder. Ces fichiers sont les suivants :

`GM_txt_configs.txt` est le fichier qui va contenir les noms de tous les fichiers de type « texte » qui contiennent des fichiers utiles au « Game Master ».

`GM_mSL_scripts.txt` est le fichier qui va contenir les noms des fichiers « scripts », c'est à dire écrits en « *mSL* » (c.f. chapitre 7 page 149).

Un script, « `mkConfigs.sh` » permet, si l'utilisateur le juge utile, de recréer automatiquement ces fichiers d'information, en explorant le répertoire du « Game Master ». Ce script, qui n'est adapté qu'à *Mac OS*, et peut, peut-être, fonctionner sous *Linux*, se lance dans une fenêtre de terminal, positionné dans le répertoire du plug-in, par la commande :

```
./mkConfigs.sh
```

Ces fichiers sont lus et analysés au lancement du plug-in. En cas de modification ultérieure de leur contenu, il sera nécessaire de quitter et de relancer REAPER, après avoir sauvé l'état du Game Master sous forme d'un préréglage, *puis* la session REAPER.

Par ailleurs, JSFX, en tant que langage, présente beaucoup de qualités, mais aussi de nombreuses restrictions dont certaines sont gênantes dans la pratique : type de données unique, les flottants doubles, fonctions non récursives, et références en avant non possibles, variables non déclarées, avec les erreurs de programmation inévitables que cela implique, absence de construction équivalente au « switch » du langage « C », absence de pointeurs sur des fonctions, choix incongrus pour les variables locales, la représentation des objets, etc. Tous ces inconvénients sont contournés du mieux possible, mais peuvent, à leur tour, contribuer à des difficultés, ou des quasi impossibilités, dans la programmation ou dans les fonctionnalités du logiciel... Que l'utilisateur pardonne au concepteur !

Chapitre 2

Modules de commande standard

L'interface du *Game Master*, on l'a indiqué, est *modulable*, c'est à dire qu'un certain nombre de modules de commandes sont disponibles, et que chaque panneau de chaque onglet est susceptible de recevoir un, ou plusieurs, de ces modules. On choisit ces modules par un « clic-droit » avec la touche « shift » (ou « alt ») enfoncée. Le pop-up menu qui s'affiche alors permet d'afficher (ou de supprimer) un module, et de déplacer verticalement ces modules les uns par rapport aux autres.

Il existe aussi un petit nombre de commandes, toujours accessibles, quels que soient les onglets et les modules affichés. Dans tous les cas, on se souviendra que le *Game Master* ne reçoit les commandes que lorsqu'il est *réceptif*, car on a cliqué dans sa fenêtre, et que le curseur de la souris est positionné à l'intérieur de celle-ci.

2.1 Commandes globales

Elles sont accessibles, soit par utilisation de la souris, soit par des touches clavier. Ainsi, cliquer sur l'un des onglets « Main », « Studio A », etc, permet d'afficher l'onglet correspondant. La touche « Tab » du clavier permet également de passer d'un onglet au suivant.

2.1.1 Souris

Un clic droit dans la barre des onglets (en haut de la fenêtre) permet d'afficher un menu de configuration, qui permet de choisir un « look and feel » pour l'ensemble du plug-in - c.f. fig. 1.4 page 32.

Un shift-clic-droit dans l'un des panneaux permet de visualiser, choisir, ou déplacer les modules affichés dans ce panneau - c.f. fig. 1.5 page 33.

2.1.2 Clavier

Quelques raccourcis claviers sont « globaux », c'est à dire actifs quelle que soit la position de la souris dans la fenêtre :

- « **tab** » Passe à la tabulation suivante
- « **shift-tab** » Passe à la tabulation précédente
- « **<** » Active ou désactive le mode « autoplay » - c.f. section 2.6 page 51.
- « **backspace** » Arrête l'un des players actifs.
- « **&** » Marque l'instant présent dans le playlog (c.f. chapitre 4 page 91).
- « **#** » ou « **suppr** » Arrête l'ensemble des players.
- « **\$** » ou « **_** » Déclenche le jeu d'un clip du groupe courant.
- « **@** » ou « **ù** » ou « **F17** » ou « **⌘** » Verrouille l'ensemble des players actifs - c.f. section 2.8 page 56.

Les autres raccourcis claviers définis dans le plug-in sont propres à un module spécifique, voire à une cellule particulière d'un module.

2.1.3 Interactions génériques typiques

2.1.3.1 Cellules

Lorsqu'une cellule d'un PAD affiche une valeur numérique modifiable, cette valeur peut être modifiée par les interactions suivantes :

souris

- clic-gauche** diminue la valeur d'une unité.
- ctrl-clic-gauche** diminue la valeur d'une dizaine.
- shift-clic-gauche** diminue la valeur d'une centaine.
- ctrl-shift-clic-gauche** diminue la valeur d'un millier.
- clic-droit** augmente la valeur d'une unité.
- ctrl-clic-droit** augmente la valeur d'une dizaine.
- shift-clic-droit** augmente la valeur d'une centaine.
- ctrl-shift-clic-droit** augmente la valeur d'un millier.

clavier

0 à 9 Les chiffres permettent d'introduire une valeur, comprise entre 0 et 9999.

z ou ↖ Remet la valeur courante à 0.

: diminue la valeur courante d'une unité.

= augmente la valeur courante d'une unité.

'left' ou 'right' (flèches de direction gauche et droite du clavier, « ◀ » ou « ▶ ») : passage à la valeur « significative » précédente ou suivante du nombre. Ce passage est en général « intelligent », c'est-à-dire que le passage se fait à une valeur correspondant à un objet effectivement défini dans la configuration. On sautera ainsi du clip « 2179 » au clip « 3001 » si aucun autre clip n'est défini dans cet intervalle.

« b » ou « n » (pour « back » et « next ») : même effet qu'avec les touches « ◀ » ou « ▶ » ci-dessus.

'up' ou 'down' (flèches « ▲ » ou « ▼ » du clavier) : passage rapide (par pas de 10 ou de 100) à la valeur significative suivante ou précédente.

'pgup' ou 'pgdn' (« page up » ou « page down », « ↑ » ou « ↓ ») : passage ultra-rapide (par pas de 100 ou de 1000) à la valeur significative suivante ou précédente.

Notons que lorsque le nombre édité représente un numéro de clip, de groupe, de banque, de mode de jeu ou de mode d'espace, une « correction intelligente » se produit quelques secondes après une interaction : le nombre introduit par l'utilisateur est automatiquement incrémenté ou décrémenté jusqu'à correspondre à un objet (clip, groupe, etc.) existant dans le système.

2.1.3.2 Sliders

FIGURE 2.1 – Exemple de slider

Les sliders permettent une interaction directe de l'utilisateur avec un paramètre de réglage du plug-in. Dans l'exemple de la figure 2.1, on distingue, de droite à gauche, les éléments graphiques suivants :

- le nom de paramètre affiché, ici « Lp.Gain », qui concerne donc le volume sonore des « loopers », ainsi que la valeur courante de ce paramètre, ici « -6.2 dB ».

- un indicateur de forme carré, indiquant par sa présence que le paramètre peut être associé à un générateur d'aléa. En l'occurrence, l'association est active, et le générateur associé est le générateur n°2, représenté par la couleur « orange ».¹
- le slider proprement dit, qui va permettre le réglage du paramètre entre deux valeurs limites, dans ce cas précis entre -120 dB et +12 dB. Le carré bleu est le « curseur » du slider, qui symbolise la valeur du paramètre entre les bornes de variation.

Les interactions possibles avec un slider sont les suivantes :

clic-gauche dans la zone de réglage du slider positionne celui-ci à l'endroit cliqué. On peut déplacer le curseur du slider en bougeant la souris tout en maintenant le bouton gauche enfoncé.

double clic-gauche permet de rétablir la valeur par défaut du paramètre, ici « 0 dB ». On obtient le même résultat avec « ctrl+shift+clic-gauche ».

clic-gauche sur l'indicateur d'association permet d'activer ou de désactiver l'association avec le générateur d'aléa. Activé, l'indicateur affiche la couleur associée au générateur d'aléa choisi ; désactivé, il passe au gris.

clic-droit sur l'indicateur d'association déclenche l'affichage d'un menu, qui permet de choisir un générateur d'aléa parmi huit.

clic (gauche ou droit) sur le nom du paramètre peut provoquer l'affichage d'une entrée indiquant les bornes effectives de la variation du paramètre, en fonction du générateur d'aléa utilisé.

2.1.3.3 Notes

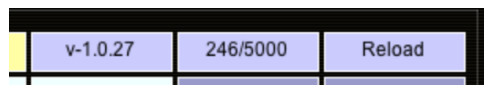


FIGURE 2.2 – Le bouton « Reload »

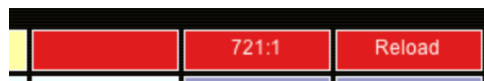


FIGURE 2.3 – Le bouton « Reload »

1. Notons que le générateur d'aléa, malgré sa présence dans ce manuel depuis 2021, n'a jamais été implémenté :-)

Plusieurs modules affichent des informations relatives à des fichiers de configuration, avec en particulier un bouton noté « Reload » (c.f. fig. 2.2). Le fichier de configuration correspondant a été en principe chargé au lancement du plug-in. Lorsque l'analyse du fichier est correcte (ici, le fichier de description des partiels), les cellules correspondantes sont de couleur bleue, et affichent le numéro « d'évolution » du fichier, renseigné dans celui-ci (1.0.27), et le nombre de partiels définis (246), ainsi que le nombre total possible (5000).

Il peut arriver qu'une erreur soit détectée dans le fichier ; la couleur passe alors au rouge, et la cellule précédente affiche un numéro d'erreur, et une « position », qui est le numéro de l'objet que le décodeur du fichier tentait de définir. (c.f. fig. 2.3). Dans cet exemple précis, « 721 » indique un mot-clef inattendu, et « 1 » indique que l'erreur survient tout au début du fichier.

Il est possible alors de corriger le fichier dans une application externe (notepad, textedit, etc.), puis de le recharger en cliquant sur « Reload ». Il est également possible de choisir un autre fichier, par un clic droit sur « Reload ». Un menu apparaît, qui propose divers fichiers pouvant répondre à l'attente de l'utilisateur. Choisir celui qui convient, qui devient dès lors le fichier par défaut associé au préréglage.

2.1.4 Glisser-déposer

Cette opération permet à un logiciel de recevoir un fichier affiché dans une fenêtre du Finder/Explorer. Cliquer sur le fichier de cette fenêtre, et, tout en maintenant appuyé le bouton de la souris, « déplacer » le fichier jusqu'à la fenêtre du Game Master, et enfin relâcher le bouton de la souris.

Cette opération n'est prise en compte, pour l'instant, que pour les fichiers scripts, dont le nom comporte le suffixe « .mSL ». Il est ainsi possible :

- de choisir un fichier de description propre à un aspect du Game Master (par exemple, un fichier décrivant les modes de jeu, les modes d'espace, etc.) et de le glisser dans le module spécifique gérant cet aspect.
- de glisser un fichier script n'importe où dans la fenêtre du Game Master, et de le lâcher en maintenant la touche « alt » enfoncée.

Si, de plus, lors de cette opération la touche « alt » était appuyée au moment où l'on relâche le bouton de la souris, le script est alors immédiatement exécuté.

2.2 Configuration globale

Le Game Master propose donc plusieurs tabulations, qu'il est possible de configurer indépendamment, chaque tabulation pouvant comporter de 1 à 4 colonnes (ou « panneaux »), et chaque colonne un ou plusieurs modules.. Cette configuration s'effectue au travers d'un menu unique, obtenu par un « shift-clic-droit » dans la fenêtre

du plug-in (c.f. fig. 1.5 page 33). Ce menu comporte cinq rubriques, chacune disposant de sous-rubriques :

Modules Donne accès à la liste alphabétique des différents modules pouvant être installés dans la colonne courante, ceux qui s’y trouvent déjà étant repérés par une « tick-mark ». Les nouveaux modules sont insérés en dessous des précédents.

Modules Disposition Permet de déplacer le module courant (dont le nom apparaît en première place dans le menu) d’une position vers le haut ou vers de bas, de le placer tout en haut ou tout en bas de la colonne, de le faire glisser dans la colonne de gauche ou la colonne de droite, ou encore de copier sa référence, pour ensuite en placer une copie dans une autre colonne ou une autre tabulation.

Panes Arrangement Permet d’échanger la colonne courante avec celle qui est située à sa droite ou à sa gauche, de déplacer l’ensemble des colonnes vers la droite ou la gauche, de copier le contenu d’une colonne pour le coller dans une autre colonne. Il est également possible de partager la tabulation en différents panneaux (colonnes), de 1 à 4. « Reset Sizes » permet d’affecter à tous les panneaux la même largeur (au cas où celle-ci avait été antérieurement modifiée), « Clear Pane » de retirer tous les modules du panneau courant, « Clear All Panes » de supprimer tous les modules de tous les panneaux de toutes les tabulations...

Tabs Management Permet de déplacer l’ensemble des panneaux d’une tabulation vers la tabulation précédente ou la suivante, et de choisir un autre nom à la places des étiquettes prédéfinies, « Main », « Studio A », etc.

Thème Permet le choix du thème de présentation de l’interface.

Si l’on a cliqué dans un module pour faire apparaître le menu, le nom de ce module s’affiche en tête de ce menu. Il suffit alors de choisir ce nom pour retirer le module du panneau. Les propriétés d’un module (par exemple, dans le cas d’un module « Parameters X », la liste des paramètres affichés) restent associées à ce module, même si celui-ci n’apparaît nulle part dans l’interface.

Notons encore qu’il est possible, au moyen des touches de clavier « shift-◀ » ou « shift-▶ » d’augmenter la largeur d’une colonne, au détriment de celle de sa voisine, en « poussant » vers la gauche ou vers la droite la limite entre les deux colonnes.

La configuration globale fait partie du « pré-réglage » et est sauvegardée par la procédure habituelle (clic gauche sur la case [+] de la barre REAPER de la fenêtre du plug-in, choix d’une option, etc.).

Main Infos

GAME PLAYER

Version 1.4.9

May 13, 2021

Start: 21/5/2021 19h41'50"

Now: 21/5/2021 22h09'00"

Up time

2h27'12"

Gfx Refresh

40

Gfx Count

19438

Serial Count

18

Last Action

5

Mem avail

1794122 w

File reader

0.0.0

Presets

1.3.21

Partials

240

Sample Rate

48.0 kHz

Block Size

4096

Block Delay

85.33 ms

GMem avail

7786096 w

Gfx doN

9299

Gfx Int

328

Gfx Usr

2555

Playing

7

Playtime/m.

100 %

Players

7

Loopers

0

MIDI Count

0

MIDI Kmd

00

MIDI p1

00

MIDI p2

00

MIDI Offset

0

MIDI date

0.000

Play State

0

Play Position

5.29 s

Sys. Tempo

120.0 bpm

Beat Position

10.58

Bl. mean dur

0.0391 ms

Bl. max. dur

13.3288 ms

Bl. CPU %

0.046 %

gfx mean dur

29.9985 ms

gfx max. dur

187.8390 ms

gfx CPU %

3.045 %

FIGURE 2.4 – Le module « Main Infos »

2.3 Module « Main Infos »

Le module « *Main Infos* » (c.f. fig. 2.4) a pour seule vocation d'afficher les différentes caractéristiques du plug-in : version, heure de début d'exécution, et nombre d'autres informations, souvent dispensables ou incompréhensibles. Sa configuration exacte varie en fonction de l'évolution des versions.

2.4 Module « System Log »

System Log	
<div> <div> <div>GAME PLAYER — Version 1.5.2 — August 4, 2021</div> <div>Start: 08/08/2021 10h58'12"</div> <div>End of initialization</div> <div>Preset Saved -- 08/08/2021 10h58'12"</div> <div>Preset Loaded -- 06/08/2021 22h01'26"</div> <div>Building scripts menu.</div> <div>Building configuration menu.</div> <div>Now: 08/08/2021 10h58'13"</div> <div>HP conf load: HPConfigs.txt</div> <div>Play modes load: PlayModes.txt</div> <div>Space modes load: SM_GranLux_21.txt</div> <div>Banks load: Banks.txt</div> <div>Clips mods. load: Clips.txt</div> <div>Sensors load: Sensors.txt</div> </div> </div>	

FIGURE 2.5 – Le module « System Log »

Le module « System Log » permet l'affichage de messages générés par les différents processus. Sur la figure 2.5, dont la capture a été effectuée immédiatement après le lancement du plug-in, différents messages indiquent successivement la version du

système, l'heure de lancement, puis le fait qu'une sauvegarde de la configuration a été exécutée automatiquement par REAPER (ce qui lui permet la réinitialisation « Reset to factory default »), et enfin qu'un chargement d'un autre préréglage a été exécuté. Suite à ce chargement, le plug-in reconstruit certaines tables, et recharge certains fichiers de configuration.

De manière générale, ce module affiche nombre d'informations d'un grand intérêt, comme « Il est maintenant très exactement 22 heures », ou encore « Une grave erreur système est arrivée, il serait plus prudent d'évacuer la salle ».

Il est possible de modifier certaines options d'affichage, au moyen d'un menu obtenu par un clic droit : passer à un affichage sur 1, 2, 3 ou 4 colonnes, augmenter ou diminuer l'espace entre les lignes, ou encore effacer l'écran.

Enfin, un certain nombre de commandes claviers sont accessibles, lorsque le curseur est dans le module :

- espace, ou « cr » : ajouter une ligne blanche au log.
- « 1 », « 2 », « 3 » ou « 4 » : afficher sur 1, 2, 3 ou 4 colonnes.
- « j » ou « k » : déplacer l'ensemble des lignes vers le haut ou vers le bas.
- « down » ou « up » : déplacer l'ensemble des lignes vers le haut ou vers le bas de dix positions.
- « z » ou « left » : revenir à la dernière ligne affichée.
- « m » et « l » : augmenter ou diminuer l'intervalle entre les lignes.
- « t » : insérer date et heure dans le log.
- « v » : insérer les informations de version et configuration dans le log.
- « p » : si le script auxiliaire est actif, enregistrer le contenu du log dans un fichier texte (nom : « syslog » suivi de la date).

2.5 Modules « Parameters »

Il existe 8 modules « Parameters », notés « A » à « F », dont les fonctionnements sont identiques, chacun pouvant être configuré différemment. La figure 2.6 page suivante montre trois exemples de configurations. Les valeurs des sliders sont contrôlées par la souris (clic gauche). Un clic droit (c.f. fig. 2.7 page 52) permet de remplacer le slider sur lequel on a cliqué par un autre, par un espace, de le supprimer ou de le déplacer vers le haut ou vers le bas. Cliquer sur l'entrée « (unset entry) » permet de rajouter un slider dans le module. Le menu permet aussi de configurer le module pour afficher une ou deux colonnes.

De manière générale, les sliders proposés à l'affichage correspondent aux différents paramètres du logiciel, dont la liste est donnée au chapitre 11 page 183. Le même paramètre peut être référencé dans différents modules, voire plusieurs fois dans le même module, si utile. Toutes les occurrences d'un même paramètre sont synchronisées dans tous les affichages.

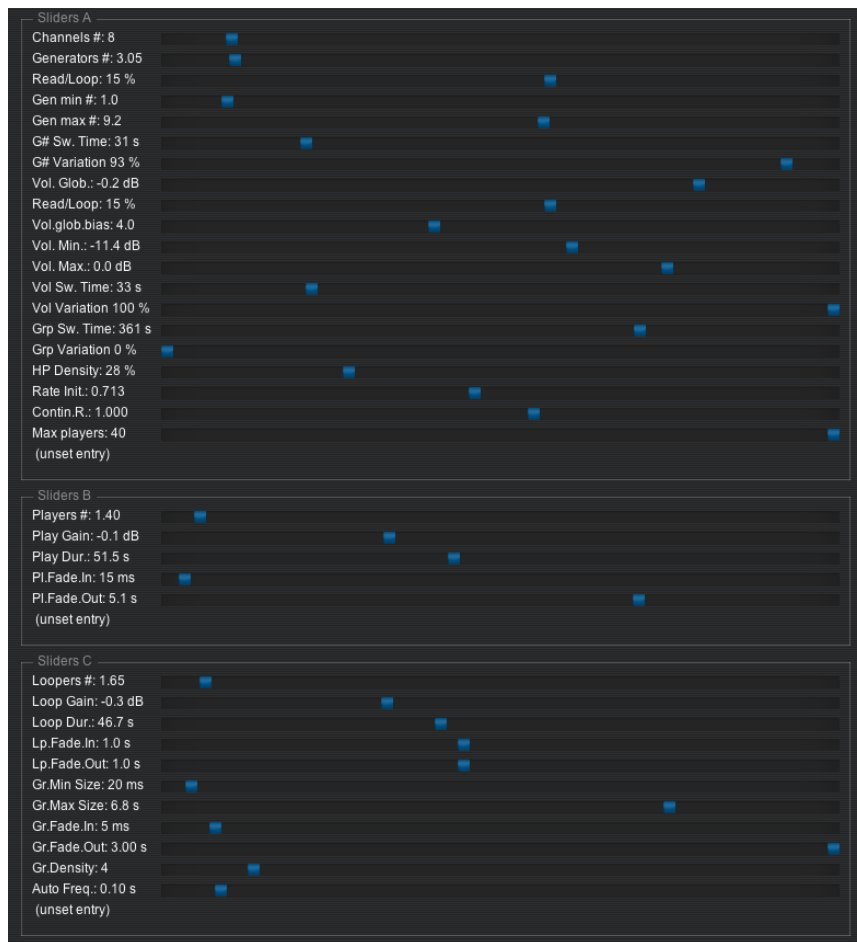


FIGURE 2.6 – Trois modules « Parameters »

2.6 Module « Play Control »

Le module « *Play Control* » (c.f. fig. 2.8 page suivante) est constitué d'un PAD de 16 cellules. Son utilisation est liée au mode « génération automatique ». Les cellules sont cliquables, et certains raccourcis clavier leur sont associés. Les cellules grisées sont celles dont le contenu n'est pas significatif pour le mode de jeu choisi.

Voici une description des différentes cellules (numérotées de gauche à droite et de haut en bas), et des interactions possibles avec elles.

- 1 Auto-play** Mode « Auto-play », pouvant être *off* (grisé) ou *on* (vert clair). Lorsque l'*auto-play* est actif, il y a génération (aléatoire et réglable par différents paramètres) de lecture de clips. Le mode est activé ou désactivé par un clic gauche sur la cellule correspondante, ou par la touche « < ». Un clic droit verrouille tous les lecteurs et interrompt la génération automatique, de même

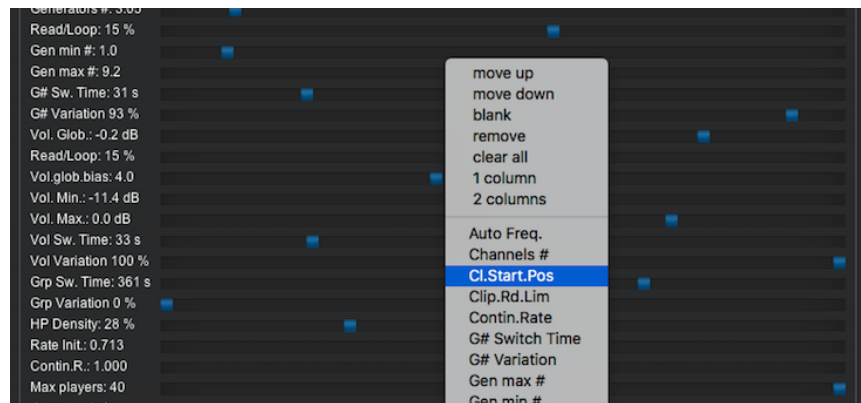


FIGURE 2.7 – Configuration du module « Parameters »

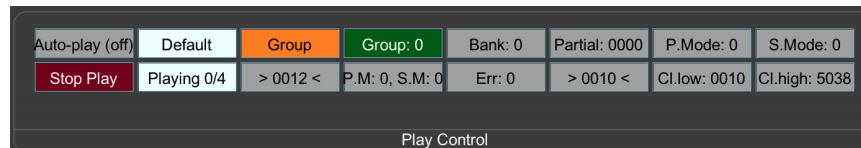


FIGURE 2.8 – Le module « Play Control »

que la touche « @ ».

2 Mode global Actuellement, un seul mode global de fonctionnement existe, le mode « Default ». « Default » indique que le studio est en mode de génération algorithmique par défaut. Dans ce mode, il est uniquement fait appel au générateur par défaut, qui, en gros, tire au hasard les clips à jouer. Son fonctionnement est décrit plus en détail ci-après.

3 Sélection Cette cellule indique quelle sélection est appliquée sur les clips joués en mode *auto-play*. La sélection s'opère au travers d'un menu, ouvert par un clic droit sur la cellule. Les options de ce choix sont les suivantes (sachant que « choisi » signifie ici « tiré au hasard ») :

Studio Pad le prochain clip joué sera choisi dans les clips associés au « *Studio Play Pad* » (c.f. section 2.7 page 54)

Group le prochain clip joué sera choisi dans les clips associés au groupe dont le numéro est affiché dans la cellule 4.

Bank le prochain clip joué sera choisi dans les clips associés au groupe et à la banque du groupe, dont les numéros respectifs sont affichés dans les cellules 4 et 5.

Partiel le prochain clip joué sera choisi dans les clips associés au partiel dont le numéro est affiché dans la cellule 6.

Clip le prochain clip joué sera le clip dont le numéro a été sélectionné dans la cellule 14, en lui appliquant les caractéristiques du partiel dont le numéro est indiqué dans la cellule 6.

Range le prochain clip joué sera choisi dans la sélection définie par les cellules 15 et 16, en leur appliquant les caractéristiques du partiel dont le numéro est indiqué dans la cellule 6.

Any Clip joue n'importe quel clip du répertoire, avec les paramètres de jeu définis par le partiel choisi dans la cellule 6.

Any Bank Choisit au hasard une banque, ou plus précisément, un partiel, et joue un clip de ce partiel. Dans ce cas particulier, le choix dans l'ensemble des partiels est équiprobable.

Un clic gauche sur cette cellule joue un clip, choisi par le mode courant.

4 Groupe Cette cellule affiche le paramètre associé au mode de sélection indiqué dans la cellule 3. Ce paramètre est le numéro de groupe.

5 Bank Cette cellule affiche le paramètre associé au mode de sélection indiqué dans la cellule 3. Ce paramètre est le numéro de banque.

6 Partiel Cette cellule affiche le paramètre associé au mode de sélection indiqué dans la cellule 3. Ce paramètre est le numéro de partiel.

7 Mode de Jeu (« P.Mode » pour « Play Mode ») Cette cellule permet de choisir un mode de jeu qui sera affecté au partiel 0. Les deux cellules 7 et 8 sont activées lorsque le partiel 0 est sélectionné (cellule 6), en grisé sinon.

8 Mode d'espace (« S.Mode » pour « Space Mode ») Cette cellule permet de choisir un mode d'espace qui sera affecté au partiel 0.

9 Stop Play Un clic gauche sur cette cellule permet d'interrompre tous les players actifs, un clic droit les interrompt également, mais de plus désactive le mode Autoplay.

10 Players actifs Cette cellule affiche à tout moment le nombre de players actifs.

11 Dernier Clip Cette cellule affiche à tout moment le numéro du dernier clip joué. Un clic gauche sur cette cellule rejoue à l'identique le clip.

12 Modes Cette cellule affiche le mode de jeu et le mode d'espace du dernier clip joué.

13 Erreur Cette cellule affiche le numéro de la dernière erreur survenue.

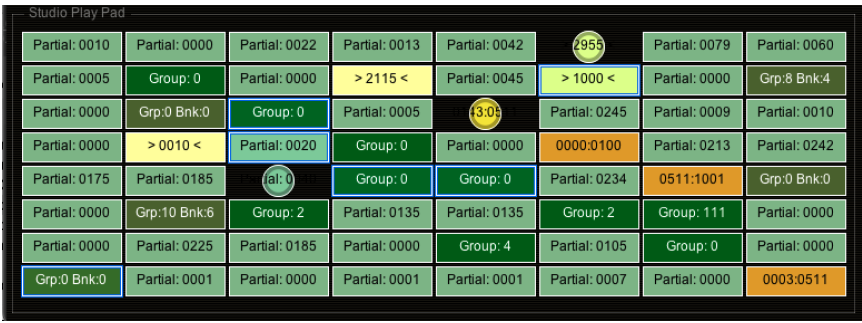
14 Clip Cette cellule permet le choix d'un numéro de clip, qui est joué lorsque l'option « Clip » de la cellule « 3 Sélection » est activée.

15 et 16 Intervalle Ces deux cellules définissent un intervalle de numéros de clips, qui sera affecté au partiel 0.

Partiel zéro C'est le seul partiel que l'on peut modifier interactivement, avec la possibilité de choisir le mode de jeu (cellule 7), le mode d'espace (cellule 8), et l'intervalle des clips (cellules 15 et 16). Les valeurs de « *Grp* » et « *Bnk* » sont également utilisées pour définir le partiel « 0 ». Ce partiel peut être référencé, comme tout autre partiel, dans le « *Studio Play Pad* ».

Générateur algorithmique par défaut « *Default* » est la seule heuristique de jeu disponible à l'heure actuelle. Elle consiste à déclencher des sons tant que le nombre de clips joués simultanément est inférieur à la polyphonie choisie (réglée par le curseur « *Generators* », ajustable de 0 jusqu'au nombre de générateurs disponibles dans la session). Dans ce mode, le son choisi est déterminé par la valeur de la troisième cellule du pad.

2.7 Module « Studio Play Pad »



Partial: 0010	Partial: 0000	Partial: 0022	Partial: 0013	Partial: 0042	2955	Partial: 0079	Partial: 0060
Partial: 0005	Group: 0	Partial: 0000	> 2115 <	Partial: 0045	> 1000 <	Partial: 0000	Grp:8 Bnk:4
Partial: 0000	Grp:0 Bnk:0	Group: 0	Partial: 0005	(3:0)	Partial: 0245	Partial: 0009	Partial: 0010
Partial: 0000	> 0010 <	Partial: 0020	Group: 0	Partial: 0000	0000:0100	Partial: 0213	Partial: 0242
Partial: 0175	Partial: 0185	al: 0	Group: 0	Group: 0	Partial: 0234	0511:1001	Grp:0 Bnk:0
Partial: 0000	Grp:10 Bnk:6	Group: 2	Partial: 0135	Partial: 0135	Group: 2	Group: 111	Partial: 0000
Partial: 0000	Partial: 0225	Partial: 0185	Partial: 0000	Group: 4	Partial: 0105	Group: 0	Partial: 0000
Grp:0 Bnk:0	Partial: 0001	Partial: 0000	Partial: 0001	Partial: 0001	Partial: 0007	Partial: 0000	0003:0511

FIGURE 2.9 – Le module « Studio Play Pad »

Ce module est spécifiquement conçu pour les expérimentations en studio, ou encore pour des performances en mode « live ». Il affiche 64 cellules « cliquables » (clic gauche), arrangées sous la forme de 8 lignes par 8 colonnes, permettant, dans la mesure de la disponibilité des lecteurs, le lancement immédiat de la lecture d'un clip choisi dans une famille particulière, configurable individuellement pour chaque cellule. Chaque cellule (c.f. fig. 2.9) peut ainsi jouer un clip spécifique, ou un clip extrait d'un groupe, d'une banque, d'un partiel, d'un intervalle, etc. Cette configuration s'effectue au moyen d'un menu, ouvert par un clic droit sur la cellule, puis en choisissant (c.f. § 2.1.3 page 44) une valeur appropriée pour un groupe, une banque, un clip, etc. Précisément, il est possible de définir les « familles » suivantes de clips :

- un partiel (c.f. § 1.6.3 page 29), défini par son numéro. Il comporte des indications sur les modes de jeu et les modes d'espace, ainsi qu'un intervalle de clips. Un clip de cet intervalle est joué avec les paramètres définis dans

le partiel. Dans l'exemple de la fig. 2.9 page précédente, la première ligne comporte, de gauche à droite, cinq cellules, d'une couleur « vert clair » (précisément, « Dark Sea Green ») réglées sur les partiels 10, 0, 22, 13 et 42 respectivement.

- un groupe, défini par un numéro de partiel comportant ce groupe. La cellule comporte une coloration plus sombre (« Dark Green ») que celle d'un partiel et affiche le numéro du groupe. Cliquer sur la cellule va provoquer la lecture d'un clip du groupe, tiré au hasard (ex : ligne 2, cellule 2).
- un groupe et une banque, également définis par le numéro de partiel - cellule de couleur vert olive (« Dark Olive Green »). La cellule affiche les numéros du groupe et de la banque. Cliquer sur la cellule va provoquer la lecture d'un clip de la banque du groupe indiquée, tiré au hasard (ligne 3, cellule 2) ; rappelons qu'une banque peut être composée de plusieurs partiels.
- un clip spécifique. La cellule est jaune clair (« Clear_Yellow ») et comporte le numéro du clip (ligne 4, cellule 2). Ce clip va être joué avec les paramètres du partiel, même s'il ne fait pas partie des clips associés au partiel.
- un intervalle de numéros de clips. La cellule est jaune orangé (« Goldenrod ») et comporte les bornes de l'intervalle (ligne 4, cellule 6). Le clip, tiré au hasard dans l'intervalle, sera joué avec les paramètres du partiel, même s'il ne fait pas partie des clips associés au partiel.

Chaque touche peut être configurée en mode « jeu unique » ou « jeu répétitif ». On passe d'un mode à l'autre par *cmd-clic-gauche* sur la cellule. Celle-ci est marquée d'un liseret colorée lorsqu'elle se trouve en mode « jeu répétitif » (ex : ligne 2, cellule 6, ou ligne 3, cellule 3).

En mode « jeu unique », un clic déclenche une lecture unique d'un clip, extrait d'une famille spécifique. Plusieurs clics successifs vont lancer autant de lectures indépendantes de clips extraits de cette même famille.

En mode « jeu répétitif », le clip dont la lecture a été déclenchée est répété jusqu'à ce qu'un nouveau clic sur la même cellule désactive la répétition. Tant que le clip joue, la cellule prend la forme d'un cercle pour indiquer le mode « répétition » (ex : ligne 1, cellule 6).

Il est possible d'associer ce module à un contrôleur MIDI externe, afin de pouvoir déclencher chaque son depuis ce contrôleur.

Signalons enfin que le module, tout en n'affichant que 64 cellules, peut en comporter plus. Les raccourcis claviers « (» et «) » permettent de déplacer de deux lignes la partie visible des cellules, et « > » se repositionne tout en haut.

2.7.1 Commandes globales

Les touches du clavier « a » à « i » permettent de jouer les clips de la première ligne du pad, les touches « q » à « k » ceux de la seconde (association réalisée sur un

clavier Mac français :-).

- w** passe temporairement les cellules du PAD en mode « partiel »
- v** passe temporairement les cellules du PAD en mode « groupe »
- b** passe temporairement les cellules du PAD en mode « banque » (groupe et banque dans le groupe)
- c** passe temporairement les cellules du PAD en mode « clip »
- l** passe temporairement les cellules du PAD en mode « intervalle »
- x** passe temporairement les cellules du PAD en mode « affichage des erreurs »

2.7.2 Commandes locales

Un clic gauche sur une cellule provoque le lancement du clip correspondant, ou (mode « jeu répétitif ») l'arrêt de celui-ci. *cmd-clic-gauche* sur la cellule change le mode de jeu de la cellule (unique ou répétitif).

Un clic droit déclenche un menu, permettant de choisir le mode (clip unique, groupe, groupe et banque, etc.) dans lequel opère la cellule.

Un clic central passe au mode « suivant » de la cellule.

2.8 Module « Sound Units »

Sound Units							
Player 1	Player 2	Player 3	Player 4	Play:2516	Player 6	Player 7	Player 8
Player 9	Player 10	Player 11	Player 12	Player 13	Player 14	Player 15	Player 16
Player 17	Loop:2511	Player 19	Loop:2502	Player 21	Player 22	Play:3211	Play:3253
Loop:3219	Loop:3229	Loop:3214	Play:2127	Player 29	Player 30	Player 31	Player 32
Player 33	Player 34	Player 35	Player 36	Player 37	Player 38		

FIGURE 2.10 – Le module « Sound Units »

Le module « Sound Units » (c.f. fig. 2.10) est constitué d'un PAD affichant l'état des différents players. Ce nombre est variable, et à tout instant il est possible d'ajouter ou de supprimer des pistes comportant des players dans leurs effets. Dans la figure représentée, il y a 38 players actifs, numérotés de 1 à 38, représentés par les cellules d'un PAD de taille 5 par 8, les 2 dernières cellules, grisées, étant inactives.

Chaque cellule représente l'un des players et affiche son état : « Player » (fond vert olive, première entrée par exemple) pour un player inactif, ou encore « Loop » (fond rouge pâle - ligne 4 entrée 1, clip 3219) ou « Play » (fond bleu, ligne 3, entrée 7,

clip 3211) pour un player jouant un clip en mode loop ou en mode play, en précisant le numéro du clip joué.

Lorsqu'un player est choisi pour jouer un clip, il devient actif, charge le clip en mémoire, le joue, puis redevient inactif. Il est possible de *verrouiller* un player. La case correspondante est alors entourée d'un petit liseret, et sa teinte est changée pour indiquer l'état du player (c'est le cas de l'entrée 5, ligne 1, clip 2516, ou de l'entrée 4, ligne 3, clip 2502). Dans ce cas, le clip est joué en permanence, jusqu'à ce que le lecteur soit déverrouillé. Enfin, un clip peut être verrouillé, et en mode silencieux : le player est actif, mais son volume de sortie est réglé à 0 (cas du player 18 ; ligne 3 entrée 2, clip 2511, ou du player 28, entrée 4, ligne 4, clip 2127).

Lorsqu'un player a terminé son travail, il redevient inactif. La couleur affichée passe alors progressivement du bleu au vert (ex : Players 17, 15, 13) ou du rouge au vert (Players 22, 12, 4, 29). Un player peut être indisponible (on a supprimé sa piste, ou celle-ci a été mutée). Il s'affiche alors avec une couleur violette. Ainsi, pour cette capture d'écran, les players 11 et 16 ont été mutés.

Le module propose un certain nombre de commandes globales (s'appliquant à tous les players, quelle que soit la position de la souris à l'intérieur du module), ou des commandes locales, s'appliquant à la cellule - et donc au player correspondant - sur laquelle est positionné le pointeur de la souris.

2.8.1 Commandes globales

@ Verrouille tous les players, coupe l'autoplay.

Déverrouille tous les players.

del Interrompt tous les players.

bs Interrompt l'un des players.

ctrl-shift-left-clic Verrouille tous les players, coupe l'autoplay.

shift-left-clic Déverrouille tous les players.

f « full volume » : rétablit le volume standard de jeu des clips actifs

g coupe ou lance l'autoplay.

h « half volume » : passe les clips actifs en « demi » volume.

i passe les changements de volumes en mode « immédiat »

l Verrouille tous les players, coupe l'autoplay.

n commute l'affichage des players par numéro de player, ou par numéro de piste.

o passe les changements de volumes en mode « semi-rapide »

p passe les changements de volumes en mode « ultra lent »

- q** commute entre affichage des numéros de players ou de piste, et l’affichage des numéros des derniers clips joués.
- r** Déverrouille, démute tous les players et relance l’autoplay.
- u** Verrouille et démute tous les players.
- w** Déverrouille tous les players.
- z** Verrouille et mute tous les players, coupe l’autoplay.

2.8.2 Commandes locales

Elles s’appliquent au player sur lequel est positionné la souris

- left-clic** sélectionne le player pour édition éventuelle dans le module « Dynamic Play Parameters ». Il affiche alors le numéro du dernier clip joué. Si le player est actif, il est en outre verrouillé
- right-clic** Si le player est inactif, le dernier clip joué par ce player est relancé. Si le player est actif, cette action le verrouille, et le mute ou le démute.
- ctrl-right-clic** tue le player courant, qui redevient inactif.
- a** affiche les informations détaillées sur le player concerné dans le module « Générateur A ».
- b** affiche les informations détaillées sur le player concerné dans le module « Générateur B ».
- c** affiche les informations détaillées sur le player concerné dans le module « Générateur C ».
- d** affiche les informations détaillées sur le player concerné dans le module « Générateur D ».
- e** rejoue le clip précédemment joué par ce lecteur.
- E** rejoue et verrouille le clip précédemment joué par ce lecteur.
- k** « kill » : interrompt le player courant, qui redevient inactif.
- m** verrouille et mute ou démute le player.
- s** passe en mode « solo » le player correspondant, et le sélectionne pour édition éventuelle dans le module « Dynamic Play Parameters ».
- t** verrouille/déverrouille le player

2.9 Module « Displayer »

Il existe en fait 4 modules identiques, les « *Displayers* » A, B, C, D. Chacun de ces modules (c.f. fig. 2.11 page suivante) peut afficher l’état détaillé de l’un des



FIGURE 2.11 – Le module « Displayer A »

générateurs sonores (les seuls players pour l'instant), dont les états sont résumés dans le module « *Sound Units* » - c.f. § 2.8 page 56.

Chaque displayer comporte un PAD, dont la première cellule affiche « Reader », « Last », « Edit » ou « Selected ». On passe d'un mode à un autre par un clic (gauche ou droit) sur la cellule. En mode « Reader », il y a affichage permanent du lecteur indiqué. En mode « Last », il y a affichage du dernier lecteur actif. En mode « Selected », il y a affichage du dernier lecteur sélectionné dans le module « *Sound Units* ». Enfin, en mode « Edit », il y a affichage permanent du lecteur indiqué, complété d'un certain nombre de sliders, permettant d'influer en temps réel sur le fonctionnement du lecteur.

La seconde cellule indique le numéro du player affiché. On passe au lecteur suivant par un clic droit, au lecteur précédent par un clic gauche.

La troisième cellule affiche le numéro du clip joué par le player, la cellule 4 indique que le numéro du clip affiché est celui du clip actuellement joué par le player, la cellule 5 indique le numéro de configuration des canaux (et donc des haut-parleurs) utilisés, et la cellule 6 va proposer certains réglages qui seront décrits ultérieurement.

Le reste du module est consacré à l'affichage des caractéristique du clip joué, et

des réglages correspondants.

Les informations affichées, et les contrôles disponibles sont modulables. Un clic droit sur l'un des affichages, ou sur l'un des sliders, ou sur l'entrée spéciale « (unset entry) » ouvre un menu qui permet d'ajuster les informations affichées à ses besoins du moment.

Note Ce module est obsolète pour l'instant, n'ayant pas encore été adapté à la nouvelle version des players.

2.9.1 Configuration des afficheurs

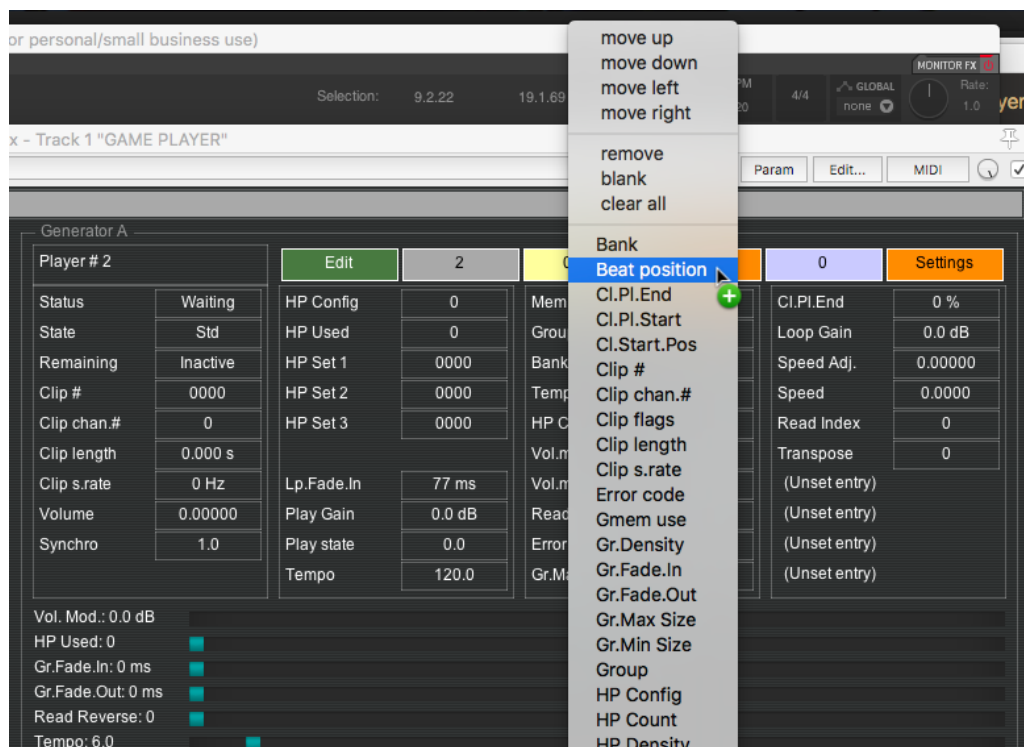


FIGURE 2.12 – Menu de configuration des afficheurs

Les « afficheurs » sont répartis en quatre colonnes, et vont présenter des informations relatives au player ou au clip en cours de jeu. Leur description complète est explicitée dans la sections consacrée aux paramètres du « Game Master ». La figure 2.12 montre le menu de configuration des afficheurs. Il est déclenché par un clic droit sur l'un des afficheurs, ou sur une entrée notée « (unset entry) ». Le menu lui-même va permettre d'effectuer l'une des actions suivantes :

- déplacer un afficheur, vers le haut, le bas, la gauche ou la droite ;

- supprimer l’afficheur sur lequel on a cliqué (« remove »), ou encore tous les afficheurs (« clear all »);
- remplacer l’afficheur sur lequel on a cliqué par un autre sélectionné dans la liste, ou, si l’on a cliqué sur une entrée « (Unset) », d’ajouter un nouvel afficheur en fin de liste, ou encore (« blank »), d’insérer une ligne blanche pour augmenter la lisibilité.
- présenter les afficheurs sur une ou deux colonnes.

On notera qu’il est possible, si cela est jugé utile, d’afficher plusieurs fois la même information à différentes positions.

2.9.2 Configuration des contrôleurs

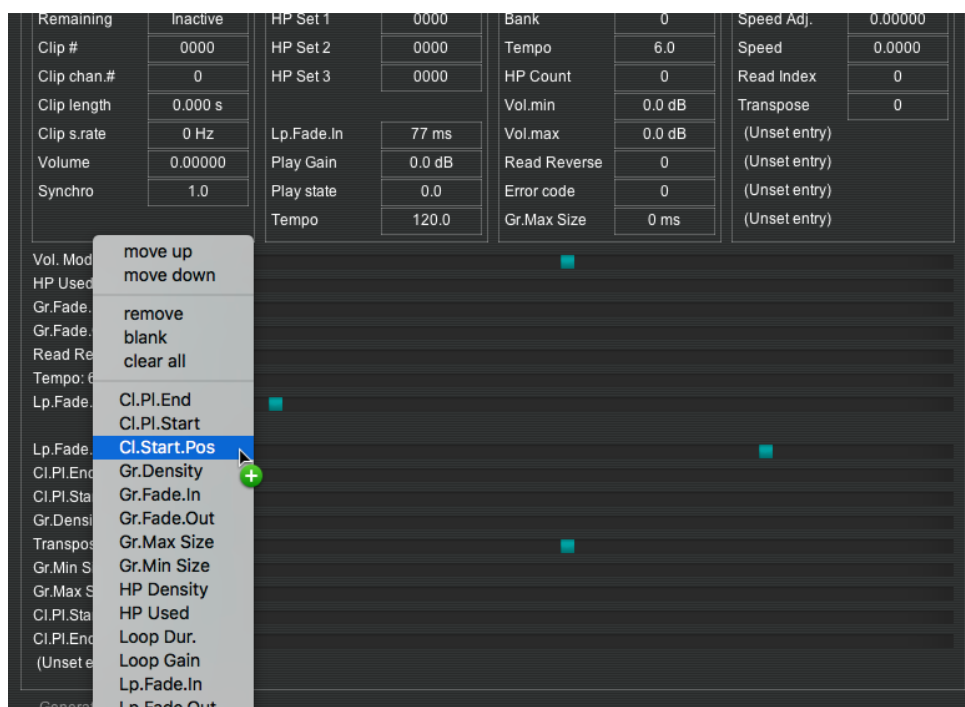


FIGURE 2.13 – Menu de configuration des contrôleurs

Tout comme les afficheurs, la présentation des contrôleurs (réglages de type *slider* permettant de modifier interactivement le comportement des players) est modulable au moyen d’un menu spécialisé (c.f. figure 2.13), qui va permettre d’effectuer l’une des actions suivantes :

- déplacer un contrôleur, vers le haut ou vers le bas ;
- supprimer un contrôleur sur lequel on a cliqué (« remove »), ou encore tous les contrôleurs (« clear all »);

- remplacer le contrôleur sur lequel on a cliqué par un autre sélectionné dans la liste, ou, si l'on a cliqué sur une entrée « (Unset) », d'ajouter un nouveau contrôleur en fin de liste, ou encore (« blank »), d'insérer une ligne blanche pour augmenter la lisibilité.
- présenter les contrôleurs sur une ou deux colonnes.

2.9.3 Réglages globaux

Le menu « Settings », obtenu par un clic droit dans la cellule de ce nom, va permettre différents types d'interactions avec les players.

Rappelons ici quelques notions :

- les « Players » génèrent un signal sonore à partir de différents réglages, dont le numéro du clip à jouer. Ces réglages leur sont transmis par le « Game Master » en fonction des algorithmes utilisés.
- chaque « Displayer » peut afficher l'état d'un player. L'affichage des réglages d'un player est déterminé par le « layout » du displayer, c'est à dire la mise en page, configurée par l'utilisateur, des contrôleurs et des afficheurs.
- les « Réglages » déterminent eux la manière dont un clip va être joué par un player.

Il existe simultanément 8 layouts, nommés de « A » à « H ». Les layouts « A » à « D » sont utilisés pour la présentation des « Displayers » « A » à « D ». Les layouts « E » et « F » sont conservés dans un préréglage global du « Game Master », les layouts « G » et « H » sont propres à une session, et peuvent être utilisés comme données temporaires pour échanger des layouts entre deux préréglages. Il est possible (au moyen du menu « Settings »), de copier un layout depuis un autre.

Chaque « Réglage » est propre à un player, et peut être modifié interactivement au moyen du displayer. Le menu « Settings » va permettre :

- de relancer l'exécution d'un player, avec les derniers paramètres qui lui ont été transmis, en mode « Play » ou en mode « Loop ».
- de changer le numéro du clip utilisé par le player, ou le numéro de la configuration de HP utilisée.
- de dupliquer un réglage, en le transmettant à l'un des players inactifs.

2.10 Module « Clips Settings »

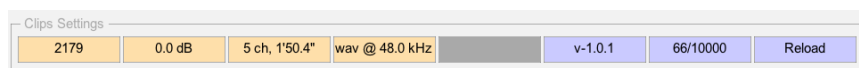


FIGURE 2.14 – Modules : Clips Settings

Ce module (c.f. figure 2.14 page ci-contre) propose l’affichage de certaines propriétés des clips, et permet de recharger le fichier de configuration des *volumes* des clips, de nom « `config-Clips.mSL` ». Il contient un *pad* unique, d’une seule ligne, dont les cellules, de gauche à droite, ont les fonctions suivantes :

- 1 – affichage du numéro de clip courant. Ce numéro n’est lié à aucune action spécifique du reste de l’application. On peut le modifier au moyen des interactions usuelles (c.f. § 2.1.3 page 44).
- 2 – affichage de la valeur de correction de volume appliquée au clip correspondant. Cette cellule n’est pas modifiable.
- 3 – affichage du nombre de canaux du clip et de sa durée. Cette cellule n’est pas modifiable.
- 4 – affichage du type du clip et de sa fréquence d’échantillonnage. Cette cellule n’est pas modifiable.
- 5 – affichage du nom du répertoire des clips.
- 6 – numéro de version du fichier « `config-Clips.mSL` » si celle-ci est précisée dans l’en-tête du clip.
- 7 – nombre de clips concernés par les modifications décrites dans le fichier « `config-Clips.mSL` ».
- 8 – bouton, permettant (clic gauche) de recharger le fichier « `config-Clips.mSL` », après une modification éventuelle de celui-ci au moyen d’un éditeur extérieur.
 - Ce bouton devient « Remove » si l’on appuie sur « ctrl ». Cliquer a pour effet d’effacer le nom du fichier de configuration. En cas de réinitialisation, aucun fichier ne sera rechargé.
 - Le bouton devient « Reset » si l’on appuie sur « alt ». Cliquer a pour effet de réinitialiser le nom du fichier de configuration à « `config-Clips.mSL` ».
 - Glisser-déposer un fichier « *mSL* » à l’intérieur du cadre du module remplace le nom du fichier de configuration par celui du fichier qui vient d’être déposé. Si l’on appuie sur « alt » durant cette opération, ce nouveau fichier de configuration est également exécuté, changeant immédiatement les caractéristiques des clips. Notons que l’opération entraîne également un nouveau scan de l’ensemble des clips (celui-ci s’exécutant en parallèle avec les autres opérations du Game Master).

Si une erreur est détectée lors de la lecture du fichier « `config-Clips.mSL` », la couleur des deux dernières cellules passe au rouge foncé, et l’avant-dernière cellule affiche le numéro de l’erreur détectée, ainsi que le dernier numéro « valide » de clip rencontré dans le fichier, indications permettant de corriger celui-ci.

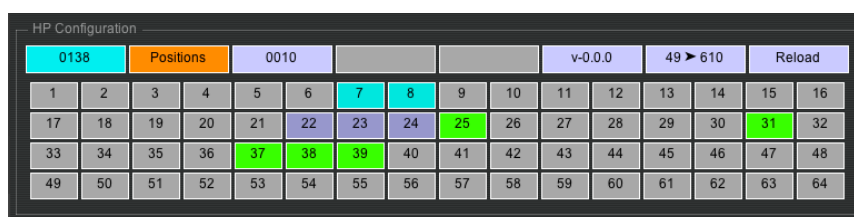


FIGURE 2.15 – Le module « HP Configuration »

2.11 Module « HP Configuration »

Le module « HP Configuration » (c.f. fig. 2.15) permet l’affichage des différentes configurations de haut-parleurs, ou plus précisément, de canaux de sorties, qui, en général, alimentent directement les haut-parleurs. Les configurations sont décrites par un fichier texte externe (c.f. section 13.3 page 200). Le nom par défaut de ce fichier externe est « `config-HP.mSL` », mais d’autres fichiers de configuration peuvent être choisis. Il suffit d’exécuter un clic-droit sur le bouton « Reload » pour choisir un autre fichier texte, qui sera utilisé comme configuration des HP.

- La première cellule affiche le numéro d’une configuration de haut-parleur. On passe aux configurations précédentes ou suivantes par des *clic-gauche* et *clic-droit*, et l’on peut aussi taper directement, lorsque le pointeur de la souris est dans cette première cellule, le numéro de la configuration que l’on veut visualiser.
- La seconde cellule permet de choisir le mode de visualisation d’une configuration, sous la forme de la liste des HP, ou sous la forme de la position de ces HP. La troisième cellule indique le nombre de HP de la configuration.
- La sixième cellule indique le numéro de version du fichier de configuration, la septième précise sous la forme « $N \triangleright P$ » le nombre total de HP, N , et le nombre de configurations définies, P .
- La dernière cellule permet, par un *clic-gauche*, de recharger le fichier de configuration (après modification éventuelle de ce dernier), et par un *clic-droit* de choisir un autre fichier, dont le nom sera dès lors attaché au préérage.
 - Ce bouton devient « Remove » si l’on appuie sur « ctrl ». Cliquer a pour effet d’effacer le nom du fichier de configuration. En cas de réinitialisation, aucun fichier ne sera rechargé, mais une configuration par défaut sera créée, appropriée à une utilisation stéréo.
 - Le bouton devient « Reset » si l’on appuie sur « alt ». Cliquer a pour effet de réinitialiser le nom du fichier de configuration à « `config-HP.mSL` ».
 - Glisser-déposer un fichier « *mSL* » à l’intérieur du cadre du module remplace le nom du fichier de configuration par celui du fichier qui vient

d'être déposé. Si l'on appuie sur « alt » durant cette opération, ce nouveau fichier de configuration est également exécuté, changeant immédiatement les caractéristiques de la configuration des HP.

Si le fichier de configuration comporte une erreur de syntaxe, ces dernières cellules sont affichées en rouge, et indiquent le numéro de la dernière configuration valide rencontrée, et un numéro d'erreur. En général, ces deux indications suffisent pour corriger le fichier dans un éditeur externe, et le recharger immédiatement.

2.12 Module « Space Modes »

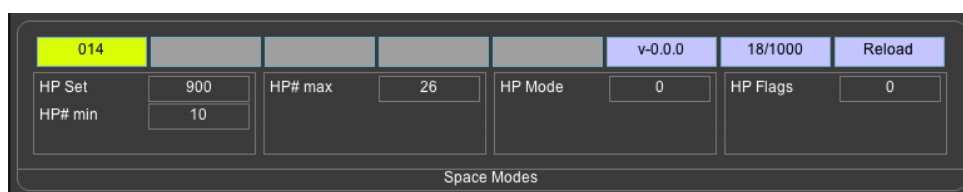


FIGURE 2.16 – Le module « Space Modes »

Ce module (c.f. fig. 2.16) permet d'afficher les différents modes d'espace disponibles dans la configuration. Les modes d'espace sont décrits par un fichier texte externe, dont le nom par défaut est « `config-SModes.mSL` », mais d'autres fichiers de configuration peuvent être choisis. Il suffit d'exécuter un clic-droit sur le bouton « Reload » pour choisir un autre fichier texte, qui sera utilisé comme configuration des modes d'espace.

La partie haute du module comporte un *pad* d'une ligne. La première cellule indique le numéro de la configuration affichée. On passe d'un mode au suivant ou au précédent par un clic (droit ou gauche) dans la cellule. Le numéro de la configuration peut aussi être modifié grâce au clavier, lorsque le pointeur de la souris est à l'intérieur du module :

- flèches gauche et droite, pour passer au numéro de configuration précédent ou suivant, flèches haute et basses pour se déplacer de 10 en 10.
- chiffres, pour taper un numéro de configuration à afficher.

Les cellules 2, 3, 4, 5 sont inutilisées, la cellule 6 indique la version du fichier de définition, la cellule 7 nombre de modes définis dans le fichier, et le nombre total d'entrées disponibles, et la cellule 8 comporte le bouton « Reload ». Un clic gauche recharge le fichier de définition (utile si on vient de le modifier avec un éditeur externe), un clic droit permet de choisir, avec un menu, un nouveau fichier de définition.

Sous le *pad*, sont affichés divers indicateurs décrivant les caractéristiques du mode d'espace : numéro de la configuration de haut-parleurs à utiliser (c.f. § 2.11

page 64), nombre de HP minimal et maximal à utiliser, modes de ces HP (en fait, indication inutilisée), indicateurs associés à ce mode.

2.13 Module « Play Modes »

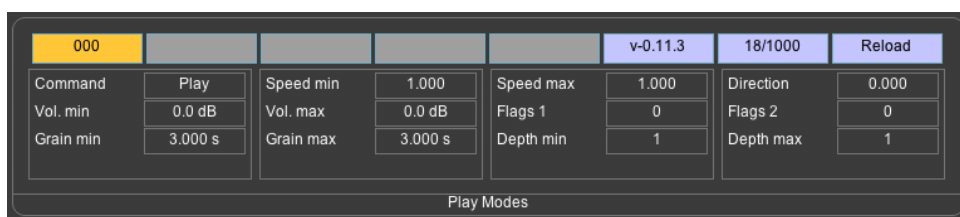


FIGURE 2.17 – Le module « Play Modes »

Ce module (c.f. fig. 2.17) permet les différents modes de jeu dans la configuration. Les modes de jeu sont décrits par un fichier texte externe, dont le nom par défaut est « `config-PModes.mSL` », mais d'autres fichiers de configuration peuvent être choisis. Il suffit d'exécuter un clic-droit sur le bouton « Reload » pour choisir un autre fichier texte, qui sera utilisé comme configuration des modes de jeu.

La partie haute du module comporte un *pad* d'une ligne. La première cellule indique le numéro de la configuration affichée. On passe d'un mode au suivant ou au précédent par un clic (droit ou gauche) dans la cellule. Le numéro de la configuration peut aussi être modifié grâce au clavier, lorsque le pointeur de la souris est à l'intérieur du module :

- flèches gauche et droite, pour passer au numéro de configuration précédent ou suivant, flèches haute et basse pour se déplacer de 10 en 10.
- chiffres, pour taper un numéro de configuration à afficher.

Les cellules 2, 3, 4, 5 sont inutilisées, la cellule 6 indique la version du fichier de définition, la cellule 7 nombre de modes définis dans le fichier, et le nombre total d'entrées disponibles, et la cellule 8 comporte le bouton « Reload ». Un clic gauche recharge le fichier de définition (utile si on vient de le modifier avec un éditeur externe), un clic droit permet de choisir, avec un menu, un nouveau fichier de définition.

Sous le *pad*, sont affichés divers indicateurs décrivant les caractéristiques du mode de jeu : commande « Play » ou « Loop », vitesses minimales et maximales de jeu, direction de la lecture (normale, 0, ou inverse, 1), volume minimal et maximal, exprimés en dB. Des valeurs supplémentaires sont liées au mode « Loop » : tailles maximale et minimale des grains, nombre maximal et minimal de grains à jouer simultanément.

2.14 Module « Banks Definition »

Partial: 0117	Group: 10	Bank: 1	Partial #: 4	Cl.Cnt: 1	v-1.0.27	258/5000	Reload
Cl.Low: 2816	Cl.High: 2816	P.Mode: 1	S.Mode: 1	Weight: 0.50	Flags: 0	Vol min: 0.0 dB	Vol max: 0.0 dB
Command	Play	Speed min	1.000	Speed max	1.000	Direction	0.000
Vol. min	-6.0 dB	Vol. max	0.0 dB	Flags 1	0	Flags 2	0
Grain min	3.000 s	Grain max	3.000 s	Depth min	1	Depth max	1
FX1. min	-120.0 dB	FX1. max	-120.0 dB	FX2. min	-120.0 dB	FX2. max	-120.0 dB
HP Set	1100	HP# max	4	HP Mode	0	HP Flags	1
HP# min	2						

Banks Definition

FIGURE 2.18 – Le module « Banks Definition »

Ce module (c.f. fig. 2.18) permet d'afficher les différentes banques présentes dans la configuration. Les configurations sont décrites par un fichier texte externe, dont le nom par défaut est « config-Banks.mSL », mais d'autres fichiers de configuration peuvent être choisis. Il suffit d'exécuter un clic-droit sur le bouton « Reload » pour choisir un autre fichier texte, qui sera utilisé comme configuration des banques.

La partie haute du module comporte un *pad* de deux lignes. La première cellule indique le numéro du partiel affiché, le partiel étant l'objet de base de la définition des banques. On passe d'un partiel au suivant ou au précédent par un clic (droit ou gauche) dans la cellule. Le numéro de partiel peut aussi être modifié grâce au clavier, lorsque le pointeur de la souris est à l'intérieur du module :

- flèches gauche et droite, pour passer au partiel précédent ou suivant, flèches haute et basse pour se déplacer de 10 en 10.
- chiffres, pour taper un numéro de partiel à afficher.


La modification d'un numéro de partiel met à jour les diverses entrées :

1. Le numéro de partiel lui-même. Les partiels sont numérotés automatiquement lors de leur définition. Le partiel « 0 » a un statut particulier, et peut être modifié dynamiquement.
2. Le numéro du groupe auquel est rattaché le partiel.
3. Le numéro de la banque dans le groupe
4. Le numéro du partiel dans la banque.
5. Le nombre de clips référencés dans le partiel
6. La version du fichier de définition
7. Le nombre de partiel définis dans le fichier, et le nombre total d'entrées disponibles

8. Le bouton « Reload ». Un clic gauche recharge le fichier de définition (utile si on vient de le modifier avec un éditeur externe), un clic droit permet de choisir, avec un menu, un nouveau fichier de définition.
9. Le numéro du premier clip référencé dans le partiel
10. Le numéro du dernier clip référencé dans le partiel
11. Le mode de jeu spécifié dans le partiel.
12. Le mode d'espace spécifié dans le partiel
13. Le « poids » associé à ce partiel
14. Divers indicateurs associé à ce partiel
15. Le volume « minimal » de jeu des clips du partiel
16. Le volume « maximal » de jeu des clips du partiel. Les deux valeurs sont exprimées en dB et peuvent être identiques.

Sous le pad de contrôle, sont situées deux zones, affichant des indications complémentaires relatives au mode de jeu et au mode d'espace référencés par le partiel. On notera par exemple que le mode de jeu comporte des indications sur le volume auquel doit être joué le clip auquel ce mode de jeu est appliqué. Cette variation potentielle de volume est combinée avec celle qui est indiquée dans le partiel pour donner le volume définitif de jeu du clip.

2.15 Module « Scheduler »



Scheduler							
Scheduler on	Next: 7	11.1	Now	Kill	Task A	Task B	Task C
Task D	Task E	Task F	Task G	Task H	Task I	Task J	Task K

FIGURE 2.19 – Le module « Scheduler »

Le module « Scheduler » permet de contrôler les « processus légers » implémentés dans le plug-in. Ceci inclut les tâches écrites en *mSL* (c.f. chapitre 7 page 149), les actions liées au jeu automatique, les opérations impliquant un délai, etc. Autant dire que lorsque le scheduler est arrêté, la plus grande partie des opérations du logiciel sont inopérantes. Le rôle et les fonctions du Scheduler sont abordés très en détail au chapitre 9 page 169. On se contente ici de décrire les interactions avec ce mécanisme proposées par ce module.

Un pad à 16 cellules permet les interactions avec le Scheduler. Ses cellules ont les fonctions suivantes :

- 1. Bouton « on/off ». Passe du vert au gris et réciproquement, indiquant si le scheduler est en fonctionnement ou non. Un clic (droit ou gauche) relance le scheduler, un ctrl-clic l'interrompt.
- 2. Indicateur de la première tâche en attente (n° 7 sur la figure).
- 3. Période d'attente avant le lancement de cette tâche, en secondes.
- 4. « Now » : cliquer sur ce bouton lance la tâche immédiatement, sans attendre la durée affichée en (3). Un « ctrl-clic » tue la tâche.
- 5. « Kill » supprime la tâche en attente.
- 6 à 16 : lancement par un clic gauche de la tâche correspondante (de « A » à « K »). Des processus peuvent être ainsi déclenchés. Un clic droit retire la tâche correspondante de la file d'attente. Des interfaces internes permettent d'associer un code (écrit en *JSFX* ou en *mSL*) à l'une des tâches.

Les tâches actives sont repérées par une couleur différentes (ici, tâches B et C). Dans l'état actuel des choses, trois tâches sont « prédéfinies » dans le GM, correspondant aux tâches « A », « B » et « C ».

- A : provoque, à intervalles irréguliers, l'incrémementation du numéro courant de Groupe. Lorsque le GM est en mode « *Auto-play* », le numéro de groupe est utilisé pour choisir les clips qui vont être joués. Le délai moyen entre deux changements de groupes est réglé par le paramètre « Grp Sw. Time ».
- B : changement du volume moyen utilisé par les générateur. Le délai moyen entre deux changements de volume est réglé par le paramètre « Vol Sw. Time ». Les paramètres « Vol min » et « Vol max » déterminent les bornes de variation du volume, exprimé en dB, « Vol Variation » permet de choisir, à son minimum, de petites variations autour de la moyenne de « Vol min » et « Vol max », et à son maximum, des nombres qui sont plus proches des limites. Enfin, « Vol.glob.bias » détermine la durée de la transition entre le volume précédent et le nouveau volume. Cette transition est très rapide pour de petites valeurs du paramètre, très lente pour la valeur maximale.
- C : changement du nombre de générateurs (players ou loopers). Le délai moyen est réglé par le paramètre « G# Sw. Time ». Les paramètres « Gen min # » et « Gen max # » déterminent les bornes de variation du nombre de générateur, et le paramètre « G# variation » permet de choisir, à son minimum, de petites variations autour de la moyenne de « Gen min # » et « Gen max # », et à son maximum, des nombres qui sont plus proches des limites.

2.16 Module « Links Manager »

Le module « Links Manager » va permettre de gérer, au sein du Game Master, la création de liens entre une « source », événement spécifique (message MIDI, uti-

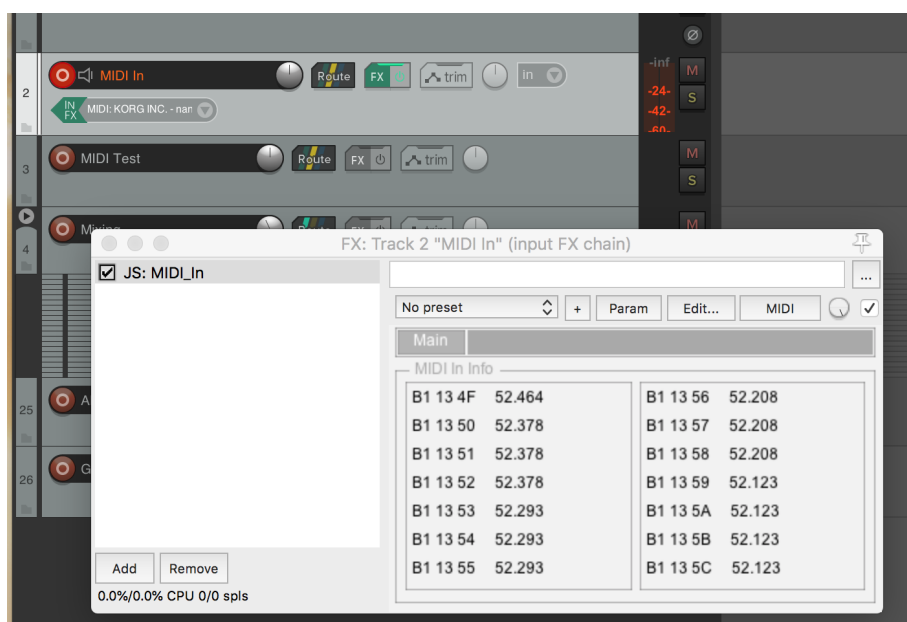


FIGURE 2.20 – Le plug-in d'interface MIDI

lisation d'un Slider, événement interne, touche du clavier de l'ordinateur) et une « cible », opération particulière du logiciel (réglage d'un paramètre, exécution d'une commande interne).

2.16.1 Nature des sources

Le module accepte à l'heure actuelle quatre types de sources :

- Les messages MIDI, venus d'un périphérique externe ou d'un autre logiciel.
- Les sliders natifs des plug-ins JSFX, au nombre de 64, qui peuvent être automatisés.
- Les raccourcis clavier (touche clavier, avec modificateurs éventuels)
- Les événements internes, créés par le logiciel lui-même.

Le choix des sources s'effectue par des clics gauches et droits (avec/sans « ctrl ») dans la cellule 1.

2.16.2 Natures des cibles

Le module définit à l'heure actuelle deux types de cibles :

- Les paramètres (c.f. § 2.5 page 50) qui influent sur les différents aspects de l'exécution du Game Master ; On choisit cette cible en manipulant l'un des paramètres affichés dans un module « Parameters », puis éventuellement, en

modifiant son numéro par des clics gauches et droits (avec/sans « ctrl ») dans la cellule 5.

- Les actions internes, qui correspondent à nombre d'opérations du Game Master. On choisit cette cible par un clic droit dans la cellule 11, qui affiche un menu permettant le choix d'une action spécifique. *Notons que les liens vers les actions ne sont pas encore opérationnels.*

2.16.3 Trace

Il est possible de tracer les événements entrants, ainsi que la gestion des liens, en activant la trace des actions (cellule 9).

2.16.4 Utilisation de périphériques MIDI

On s'intéresse dans un premier temps aux diverses commandes qu'un périphérique MIDI est susceptible d'envoyer à l'application. Pour ce faire, on passe par un plug-in spécifique, qui peut être écrit spécifiquement pour le périphérique connecté, ou par un plug-in « généraliste », qui se contentera de transmettre les informations reçues. C'est ce dernier, « MIDI_In », qui est illustré à la figure 2.20 page précédente. Pour l'utiliser, il convient :

- de créer une piste spécifique, ici la n°2, nommée « MIDI In », que l'on commute en enregistrement (clic gauche sur le bouton rouge sombre, qui devient actif, rouge clair)
- de placer le plug-in « MIDI_In » dans les **plug-in d'entrée de la piste** (ici, la « flèche » verte, à gauche, de nom « IN FX », et non les plug-in de sortie, de nom « FX » tout court).
- de choisir le périphérique MIDI connecté en entrée, ici un *Korg nanoCONTROL2*, au moyen du menu associé au sélecteur placé immédiatement à droite de la flèche « IN FX », clic-droit, « *Input : MIDI* ▸ *Korg Inc. - nanoCONTROL2* ▸ *All channels* » (Il conviendra préalablement de signaler, dans les préférences de REAPER, que l'on va utiliser ce périphérique).

Les commandes actionnées sur le périphérique MIDI sont dès lors tracées dans la fenêtre du plug-in. Ici, par exemple, « B1 13 4F » (les valeurs sont en hexadécimal) a été envoyé par action sur le contrôleur (le « B » initial) du canal MIDI « 1 », de numéro 19 (hexadécimal « 13 »), qui a envoyé la valeur « 79 » (hexadécimal « 4F »). Le quatrième nombre indique une date en secondes depuis l'ouverture de la session. Les traces antérieures (de haut en bas, et de gauche à droite) montrent que l'on a fait varier la position de ce contrôleur (l'un des potentiomètres) de la position « 5C » à « 4F ».

Il est possible de vérifier que les commandes MIDI reçues par le plug-in « MIDI_In » sont bien transmises au Game Master en cliquant sur le bouton « MIDI Log » du



FIGURE 2.21 – Le module « Links Manager », mode trace

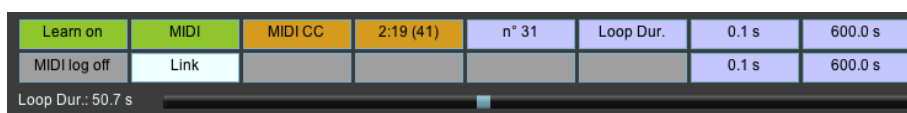


FIGURE 2.22 – Le module « Links Manager », choix des éléments du lien

module « Links Manager » (c.f. fig. 2.21). La trace des commandes MIDI est alors affichée dans le journal de bord (c.f. § 2.4 page 49).

2.16.5 Connexion d'un contrôleur MIDI

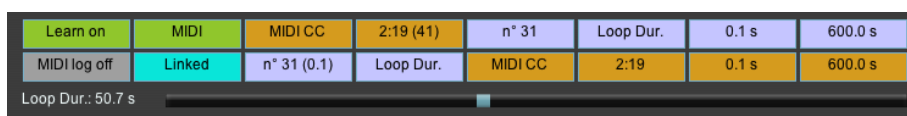


FIGURE 2.23 – Le module « Links Manager », lien établi

Vérifier que la cellule 2 (type de la source) affiche bien « MIDI » (au besoin, cliquer sur la cellule pour changer de mode). On peut dès lors associer un contrôleur du périphérique à un paramètre du logiciel :

- cliquer le bouton « Learn » (cellule 1), qui passe au vert en devenant actif. Le module affiche également le slider de réglage du paramètre « courant »
- choisir un paramètre que l'on désire contrôler. Dans la figure 2.22, l'on a cliqué, quelque part dans un des modules « Parameters », sur le contrôleur « Loop Dur(ation) », dont une copie s'affiche dès lors dans le module. Les caractéristiques du paramètre, numéro interne (31), nom, valeur minimale et valeur maximale, s'affichent également dans les cellules 5 à 8 du Pad. Des clics gauches et droits (avec/sans « ctrl ») dans la cellule 5 permettent de passer aux paramètres précédents ou suivant
- choisir l'un des contrôleurs du périphérique MIDI, en le manipulant (déplacement du slider, rotation du potentiomètre, appui sur une touche). Le numéro de ce contrôleur, ainsi que sa valeur, s'affichent dans les cellules 3 et 4 du Pad. On notera que la cellule 10 affiche « Link » sur un fond blanc, montrant que la création d'une liaison entre le paramètre et le contrôle est possible.

- cliquer (clic gauche) sur ce bouton « Link » (cellule 10), qui passe au cyan et affiche « Linked », montrant que la liaison est établie (fig. 2.23 page ci-contre). Les cellules 11 à 16 affichent dès lors les caractéristiques de la liaison. On peut vérifier que toute la course du contrôleur MIDI permet d'obtenir toutes les valeurs permises du paramètre.
- Pour supprimer la liaison, cliquer (clic gauche) dans la cellule 11 ou la cellule 13.

Note Si un contrôleur MIDI est déjà utilisé dans un lien, l'action associée est affichée dans les cellules 11 et 12. Il est possible de supprimer cette association par un clic gauche dans la cellule 11. Si la cible du lien (le paramètre à commander) est déjà utilisée dans un lien, la commande associée est affichée dans les cellules 13 à 16. On peut supprimer cette association par un clic gauche dans la cellule 11. Notons qu'une commande particulière (un contrôleur MIDI spécifique) ne peut être associée qu'à un seul paramètre, mais que plusieurs commandes différentes peuvent être associées au même paramètre. Dans ce dernier cas, les cellules 13 à 16 affichent successivement (avec un délai de l'ordre de 5 secondes) les différentes commandes associées à ce paramètre.

2.16.5.1 Sélectivité

Learn on	MIDI	MIDI CC	2:7 (0)	n° 36	Lp.Fade.In	180 ms	780 ms
MIDI log off	Linked	n° 36 (180)	Lp.Fade.In	MIDI CC	2:7	180 ms	780 ms
Lp.Fade.In: 180 ms							

FIGURE 2.24 – Le module « Links Manager », bornes de variations modifiées

On peut également ne choisir qu'un sous-ensemble des valeurs permises au paramètre. C'est ce que montre la figure 2.24. Pour ce faire,

- choisir la valeur du réglage du paramètre, ici « Lp. Fade In », à laquelle on veut voir correspondre le « 0 » du contrôleur, par exemple, 180 ms. Cliquer sur la cellule 7, qui va maintenant afficher cette valeur, au lieu du « 2 ms » initial.
- choisir la valeur du réglage du paramètre à laquelle on veut voir correspondre la valeur maximale du contrôleur (127, en termes de MIDI), par exemple 780 ms. Cliquer sur la cellule 8, qui affiche alors 780 ms au lieu de 30000.
- cliquer enfin sur « Link » pour rétablir la liaison. On peut vérifier que les positions extrêmes du contrôleur MIDI correspondent maintenant à ces deux valeurs.

Quand les liaisons souhaitées ont été réalisées, il convient de repasser « Learn » et « MIDI log » en mode « off ».

Certains périphériques MIDI utilisent des boutons (on/off) comme contrôleurs. Ainsi, sur le Korg nanoKONTROL2, tous les boutons sont associés à des contrôleurs, chacun envoyant alternativement ce contrôleur avec la valeur « 00 » ou « 7F » (127). Ces boutons peuvent, de la même façon, être associés à un paramètre, et vont donc pouvoir envoyer alternativement deux valeurs différentes.

Note Il est également possible d'établir un lien qui, quelle que soit la valeur du contrôleur, envoie toujours la même valeur. Pour ceci, après le choix du contrôleur MIDI et du paramètre à contrôler, cliquer sur la cellule 3, qui affiche dès lors « MIDI CC/SV » (pour « single value »). Choisir, avec le slider de réglage du paramètre la valeur désirée, puis cliquer la cellule 10, « Link ». Le contrôleur enverra alors toujours la même valeur au paramètre.

2.16.6 Connexion d'une note MIDI

Learn on	MIDI	MIDI Note	11:40 (0)	n° 100	Gr.Density	10	10
MIDI log off	Linked	n° 100 (10)	Gr.Density	MIDI note	11:40	10	10

Gr.Density: 10

FIGURE 2.25 – Le module « Links Manager », Lien à une note MIDI

De la même manière, il est possible d'associer, à une note MIDI, une valeur spécifique d'un contrôleur. Dans l'exemple de la figure Le module « Links Manager », Lien à une note MIDI, le contrôleur a envoyé la note MIDI 40. La cellule 2 affiche alors « MIDI Note ». L'utilisateur a choisi le paramètre « Gr. Density » (numéro interne 100), qu'il a réglé à 10, puis cliqué sur la cellule « Link » (10), qui affiche maintenant « Linked ». Tout appui sur la touche correspondant à la note MIDI 40 va dès lors régler le paramètre « Gr. Density » à la valeur 10.

2.16.7 Utilisation d'un slider JSFX

Learn on	Slider	Slider 50	(0)	n° 95	Gr.Max Size	2 ms	30000 ms
MIDI log off	Linked	n° 95 (2)	Gr.Max Size	Slider 50	(0)	2 ms	30000 ms

Gr.Max Size: 2 ms

FIGURE 2.26 – Le module « Links Manager », Lien à un slider natif

Les plug-ins JSFX peuvent déclarer jusqu'à 64 sliders. C'est le cas du Game Master, mais ces sliders sont masqués, et ne sont donc pas accessibles par l'interface utilisateur. La raison en est que l'approche graphique utilisée par le Game Master offre plus de souplesse dans les caractéristiques des réglages, et permet d'en utiliser un plus grand nombre. Cependant, les sliders natifs JSFX peuvent être automatisés. Le module « Links Manager » va permettre d'associer n'importe quel slider natif à n'importe quel paramètre, qui pourra dès lors bénéficier de l'utilisation des courbes d'automation offertes par REAPER.

Pour ceci, on va régler la source des liens sur « Slider » (clics gauche/droit dans la cellule 2). Le slider natif courant (initialement, le numéro 1) s'affiche alors dans l'interface, tout en haut de la fenêtre, juste au-dessus de la barre des onglets.

- Choisir le numéro de slider désiré (de 1 à 64), par des clics gauches ou droits (avec/sans « ctrl ») dans la cellule 3.
- Choisir le paramètre à commander, soit en cliquant sur son nom dans l'un des modules « Parameters », soit par des clics gauches ou droits (avec/sans « ctrl ») dans la cellule 5.
- Manipuler le slider natif, d'un bout à l'autre de sa course, pour permettre au Game Master de repérer les bornes extrêmes de son réglage.
- cliquer enfin sur « Link » pour rétablir la liaison. On peut vérifier que la manipulation du slider natif permet dès lors de modifier le paramètre choisi.

Tout comme avec un contrôleur MIDI, il est possible de restreindre l'intervalle des valeurs commandées par le slider : manipuler le réglage du paramètre pour obtenir une première valeur, cliquer la cellule 7, choisir une seconde valeur, cliquer la cellule 8, et enfin cliquer sur la cellule « Link » (10) pour rétablir le lien.

2.16.8 Raccourcis clavier

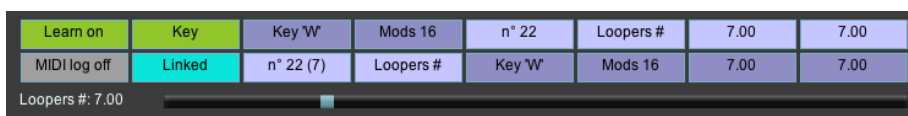


FIGURE 2.27 – Le module « Links Manager », Lien à une touche clavier

Il est également possible de créer des raccourcis clavier, en associant une touche du clavier (et d'éventuels modificateurs, shift, command, alt ou control) à une action spécifique, ou à une valeur d'un paramètre.

- Choisir (cellule 2) le mode « Key »
- taper une touche du clavier, avec des modificateurs. La touche s'affiche (cellules 3 et 4), ici « alt-W »

- Choisir le paramètre à commander, soit en cliquant sur son nom dans l'un des modules « Parameters », soit par des clics gauches ou droits (avec/sans « ctrl ») dans la cellule 5.
- Choisir la valeur désirée pour le paramètre, ici « 7 » pour le paramètre « Loopers # ».
- cliquer enfin sur « Link » pour rétablir la liaison (c.f. fig. 2.26).

On notera que les raccourcis claviers sont désactivés en mode « Learn on », et qu'il faut repasser en « Learn off » pour tester le raccourcis qui vient d'être créé.

2.16.9 Événement interne.

L'interface est définie, mais les événements internes ne sont pas encore implémentés en tant que déclencheurs d'actions.

2.17 Module « Sensors »



FIGURE 2.28 – Le module « Sensors »

Le module « Sensors » permet de travailler avec une configuration spécifique de capteurs de mouvement (détecteurs infra-rouge) communiquant en MIDI sur USB avec l'ordinateur. Dans cette configuration, neuf capteurs sont utilisés, qui envoient les notes midi de « 0 » à « 8 ». Le module a donc été conçu spécifiquement pour cette configuration, mais les concepts généraux développés permettraient de l'adapter aisément à tout autre configuration.

Il repose sur l'utilisation d'un fichier de commandes, le fichier « `config-Sensors.mSL` », qui contient un ensemble de commandes décrivant, sous forme de séquences de nombres et de mots-clefs, le comportement attendu du Game Master en réponse aux informations venant des capteurs. La syntaxe de ce fichier est décrite au § 13.9 page 216.

Le module propose, à gauche, un premier pad de six cellules, servant à l'interface avec la partie du logiciel gérant les capteurs. Les cellules ont la fonction suivante :

1. Sensors on/off. Permet de diriger les entrées MIDI vers le module, ou de les ignorer.

2. Play Ctl. on/off. Gère la multiphonie des sons déclanchés par interaction avec les capteurs. Lorsque ce contrôle est activé, le paramètre « » permet de définir le nombre maximal de players actifs. Que ce nombre est atteint, si un nouveau son est lancé, le son le plus anciennement déclanché est arrêté.
3. Unlock/Stop. Un clic gauche déverrouille tous les capteurs inhibés. Un clic droit interrompt tous les sons.
4. Mode : permet de passer (clic gauche/clic droit) au mode précédent ou suivant (de 0 à 7). A chaque mode est associé un comportement spécifique de l'ensemble des capteurs, et une couleur spécifique.
5. indique le nombre de commandes définies dans le fichier de configuration, ou affiche un numéro d'erreur et sa position.
6. permet (clic gauche) de recharger, après modification externe, le fichier de configuration, ou (clic droit) d'en choisir un nouveau dans une liste proposée.

A la droite du pad de commande, un second pad à 9 cellules, une par capteur, indique l'activation des capteurs. Chaque cellule prend initialement la couleur du mode courant, couleur qui passe progressivement au gris en une vingtaine de secondes. Il est possible de passer d'une configuration 3x3 à 2x2 en tapant « 2 » ou « 3 ».

Quelques indicateurs supplémentaires permettent d'afficher la dernière commande MIDI reçue, et la dernière fonction exécutée par le module.

2.18 Module « Random States »

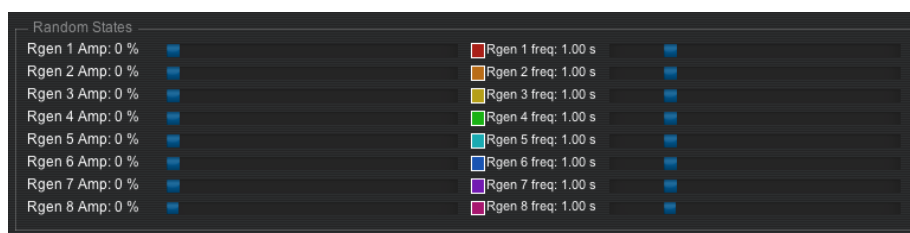


FIGURE 2.29 – Le module « Random States »

Le module « Random States » est mentionné seulement en passant, dans la mesure où ses fonctionnalités ne sont pas encore connectées au restant de l'application. Il devrait, à terme, faire office de modulateur pour les différents paramètres, en proposant diverses formes et sources de modulation.

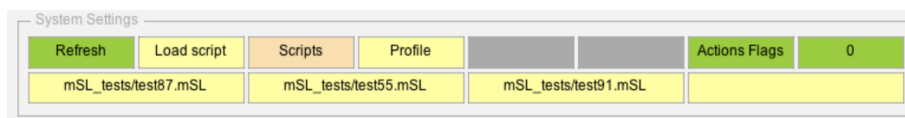


FIGURE 2.30 – Le module « System Settings »

2.19 Module « System Settings »

Le module « System Settings » permet de modifier, au moyen de scripts, nombre de paramètres associés au Game Master. Les cellules ont les fonctions suivantes :

1. Le bouton « Refresh » permet de recharger l'ensemble des fichiers de configuration du logiciel, et de relancer l'exploration des clips, étape nécessaire si de nouveaux clips sont ajoutés alors que le Game Master est actif.
2. Le bouton « Load script » permet (clic droit) d'afficher le menu de la liste des scripts connus du plug-in. Un script sélectionné est immédiatement chargé et exécuté. Son nom apparaît en outre dans la liste des derniers scripts chargés, liste occupant les lignes suivantes du pad.
3. Le bouton « Script » (inactif) peut être commuté en « Kill Scripts » (par appui sur la touche « `ctrl` »), qui permet d'interrompre l'exécution de tous les scripts « *mSL* » actifs.
4. Inutilisé
5. Imprime dans le log la liste des *fichiers* utilisés pour configurer le Game Master.
6. Le bouton « Aux. Script » envoie au script auxiliaire (écrit en « eel2 »), s'il est présent, la commande définie par (« Aux command », « Aux par 1 », « Aux par2 », etc.). Ce bouton est coloré en gris lorsque le script n'a pas été lancé. Il est vert lorsque le script est actif, et prêt à répondre à une commande, orangé-brun lorsqu'il est occupé à traiter une commande longue. Par appui sur la touche « `ctrl` », le bouton passe au rouge, et permet d'interrompre le script par l'envoi d'une commande d'arrêt. Si celle-ci ne fonctionne pas, utiliser le menu REAPER « Action » (c.f. § 2.19.1 page suivante).
7. « Action Flag » permet de sélectionner par un menu (clic droit) l'un des flags du système. La valeur correspondante est alors affichée, en hexadécimal ou en décimal (taper 'X' pour alterner), dans la cellule suivante. 'up' et 'down' permettent de se déplacer dans la liste des flags.
8. Taper un nombre (en hexadécimal ou décimal) permet de modifier le flag dont le nom apparaît dans la cellule précédente. Les valeurs introduites sont immédiatement affectées à l'indicateur choisi par la cellule 7. Selon le mode, différents caractères sont reconnus :

- (a) Mode hexadécimal : les caractères '0' à '9', 'A' à 'F' et 'a' à 'f' permettent de composer le nombre. Les touches 'left' et 'right' génèrent des décalages à gauche (insertion de 0) ou à droite (suppression du caractère de droite). '+' et '-' incrémentent ou décrémentent le nombre. 'Z' le remet à zéro.
- (b) Mode décimal : les caractères '0' à '9'. Les touches 'left' et 'right' permettent de décaler le point décimal. '+' et '-' incrémentent ou décrémentent le nombre. 'backspace' supprime la partie décimale du nombre. 'Z' le remet à zéro.

Les autres cellules affichent les noms des derniers scripts « *mSL* » exécutés. Cliquer sur l'une d'elles relance le script correspondant.

2.19.1 Script auxiliaire

Ce module permet des interactions avec un script auxiliaire, écrit en « eel2 », qui peut être lancé depuis REAPER. L'implémentation décrite ici est expérimentale, et sera très probablement modifiée dans les versions ultérieures du Game Master. Voici comment installer et charger ce script :

1. Lancer REAPER. Choisir le menu « Options ▸ Show REAPER Resource path in Explorer/Finder ». REAPER affiche alors son répertoire des ressources.
2. Installer le fichier « *GMAuxiliary.eel* » dans le répertoire « Scripts » du répertoire des ressources de REAPER.
3. Choisir dans le menu « Actions » l'option « Show actions list ». Ceci ouvre la fenêtre des actions.
4. Dans cette fenêtre, cliquer sur le bouton « New action... ». Un menu s'affiche, choisir « Load Reascript ». Le sélecteur de fichiers s'affiche, montrant le répertoire des ressources de REAPER. Naviguer, choisir le fichier que vous venez d'installer, « *GMAuxiliary.eel* ».
5. Le nom « Script : *GMAuxiliary.eel* » s'affiche en « Description » dans la liste des actions de REAPER. Ce travail (rubriques 1 à 5) se fait une seule fois, le script apparaissant dès lors dans la liste des actions de REAPER à chaque nouveau lancement de celui-ci.
6. Le script est maintenant connu de REAPER, mais est inactif. Pour l'exécuter, sélectionner son nom dans la fenêtre des actions, et cliquer le bouton « Run » en bas à droite. On peut alors fermer la fenêtre des actions (bouton « Close »). Le script affiche une fenêtre (« ReaScript console output »), avec un premier message, « GMA started » suivi de la date s'il est arrivé à établir une connexion avec le Game Master.

7. Le script est maintenant susceptible de répondre aux commandes venues du Game Master. Dans le module « System Settings », le bouton « Aux. Script » doit s'afficher en vert.
8. Notons que dans le menu « Actions » s'affiche une nouvelle rubrique : « Running script : GMAuxiliary.eel ». Si l'on choisit cette rubrique, le dialogue « ReaScript task control » s'affiche, qui permet d'arrêter le script en cliquant sur « Terminate instances ».
9. Aux lancements suivants de REAPER, le script apparaît dans la fenêtre des actions. Il suffit de le choisir et de cliquer « Run » pour qu'il devienne actif.

Les différentes actions proposées par le script sont explicitées au chapitre 12 page 189. Précisons ici que le fonctionnement du Script est *asynchrone* par rapport au Game Master : celui positionne certaines valeurs dans la mémoire partagée, puis doit attendre l'intervention du Script. Celui-ci consulte la mémoire partagée à chaque frame graphique (soit entre 20 et 30 fois par seconde), et exécute l'action demandée, positionne diverses valeurs dans la mémoire partagée, puis se remet en attente. On ne peut donc envoyer qu'une commande à la fois au script, et l'exécution de celle-ci est asynchrone.

2.19.2 Commande du script auxiliaire depuis le module « System Settings »

Lorsque la souris est sur l'un des boutons 7 ou 8, il est possible d'envoyer des commandes au script auxiliaire. Attention : ces commandes sont décrites au chapitre 12 page 189, mais seules les commandes effectivement réalisées par le script, c'est-à-dire à partir du numéro 111 et jusqu'à 140 seront reconnues, les autres étant ignorées.

Supposons que l'on désire exécuter la commande « quitter REAPER » depuis le module System Settings. Cette commande, nous dit la liste des Actions de REAPER, a pour numéro 40004. On effectuera la suite de manoeuvres suivantes :

1. Choisir dans la cellule 7 « Aux command » (soit par le menu : clic droit, soit en se déplaçant au moyen des touches « up » et « down »).
2. Passer éventuellement l'affichage en mode décimal : touche « X ».
3. Taper le numéro de la commande du script qui permet d'exécuter des actions REAPER, 125 dans la liste du chapitre 12 page 189.
4. Choisir maintenant « Aux par 1 » qui permet d'introduire le premier (et ici unique) paramètre de cette commande 125.
5. Taper la valeur 40004, qui est le numéro d'action associée à la commande « Quit » dans les Actions de REAPER.

6. Envoyer la commande au script, soit en tapant « \$ », soit en cliquant la cellule 6, « Aux Script ».

2.19.3 Commande du script auxiliaire depuis un script « *mSL* » ou depuis un pad...

L'envoi de commandes au script auxiliaire s'effectue, depuis un programme *mSL*, par la fonction « `action` ». Ainsi, la commande décrite ci-dessus pour arrêter REAPER peut s'écrire tout simplement :

```
action(125, 40004);
```

Pour enchaîner plusieurs actions du script auxiliaire, il faut les lancer une par une, avec un temps d'attente de l'ordre d'un dixième de secondes. Ex : se positionner au marqueur de numéro 3, puis lancer la lecture, s'écrira :

```
action(113, 3); wait(0.1); action(114);
```

Dans une action liée à un pad de senseurs, par exemple : « le capteur 2 sélectionne le marqueur de numéro 3, puis lance la lecture de REAPER », on utilisera de même un « Wait » explicite :

```
... DefSeq 2 DoKmd 113 3 Wait 0.1 114 ...
```

Notons que ces valeurs 113 et 114 peuvent être représentées par leurs codes, en *mSL* :

```
action('GoMark, 3); wait(0.1); action('StrtRead);
```

ou encore (notez la subtile différence, les actions sont réunies dans un bloc « data » unique)

```
action('(GoMark 3 Wait 0.1 StrtRead));
```

Et pour les pads de senseurs :

```
... DefSeq 2 DoKmd GoMark 3 Wait 0.1 StrtRead ...
```

2.20 Module « Script Manager »

Le module « Script Manager » offre un début de gestion des scripts écrits en « *mSL* ». Son utilisation est essentiellement réservée, pour l'instant, au test de tels scripts. L'utilisateur curieux peut (menu accessible par un clic droit sur le bouton « Load ») lancer le chargement et l'exécution du script choisi. La trace de ces opérations s'affiche dans le « log » du système (c.f. § 2.4 page 49). Plusieurs scripts peuvent

Script Manager							
Load	Show Mem.	Run GC	Unlocked		Xop: 0	Fr.Ent 32	No err.
Mem size	28311391	Used mem	0	GC count	0	Pr.Active	0
Free mem	0	Used % m.	0.00 %	mSL Lock	0	Pr.Wait	0
Free % m.	0.00 %	Used blocks	0	wrng fr.	0	Pr.Lock	0
Free blocks	0	Free [Big]	0	Free [Mini]	0	Free [Med.]	0
Free [Huge]	1	Tmp. Inst/bl	0	Free [Large]	0	Max.gc.dur	0.00 ms.
Instr./second	0	Tmp. Dur/bl	0.00 ms.	Max. Inst/bl	0	Max xop.d	0.00 ms.
Max. Inst/s.	0	malloc #	0	Max. Dur/bl	0.00 ms.	Max GLp	2.16 ms.
		mfree #	0	Active	0	Max stack	0
		low []	376612	Thread. @	0		
		high []	28868096	Th. mark.	0		

FIGURE 2.31 – Le module « Script Manager »

être simultanément actifs. On notera que l'exécution de ces scripts, qui consomment pour certains (et « c'est étudié pour ») énormément de CPU, ne perturbe pas le fonctionnement normal du Game Master.

Après exécution de quelques scripts, il est possible de lancer le « *Garbage Collector* » par le bouton « *Run GC* », clic gauche (mode silencieux) ou clic droit (mode bavard), et vérifier que les valeurs affichées dans « Mem size » et « Free mem » sont identiques.

Les valeurs concernant la gestion dynamique de la mémoire ne sont pas affichées en temps réel. Pour en obtenir une vision instantannée, il faut cliquer sur la cellule « Show Mem. » ce qui lance les calculs (rapides) correspondants.

2.21 Module « Memory »

Le module « Memory » est utilisé pour la mise au point du système et de scripts écrits en « mSL ». Son utilisation apporte une aide pertinente lors de l'écriture de scripts, de test des sensors, etc.

L'ensemble de ses fonctions sera décrit ultérieurement.

codegen blocks	@442152	<-->	-70	442226	-1	-1	0
442152 :	'memr'	28311548	'size'	28311391			
442156 :	'frhd'	10	'frd'	150			
442160 :	'fmwd'	151	'fmwd'	28753701			
442164 :	0	-10	-10	8			
442168 :	-13	-13	12	-16			
442172 :	-16	16	-19	-19			
442176 :	24	-22	-22	32			
442180 :	-25	-25	48	-28			
442184 :	-28	64	-31	-31			
442188 :	96	-34	-34	128			
442192 :	-37	-37	192	-40			
442196 :	-40	256	-43	-43			
442200 :	384	-46	-46	512			
442204 :	-49	-49	768	-52			
442208 :	-52	1024	-55	-55			
442212 :	1536	-58	-58	2048			
442216 :	-61	-61	3072	-64			
442220 :	-64	4096	-67	-67			
442224 :	6144	-70	-70	8192			
442228 :	-73	-73	12288	-76			

Debugger

FIGURE 2.32 – Le module « Memory »

Chapitre 3

Le module « Clips Selection »

3.1 Introduction

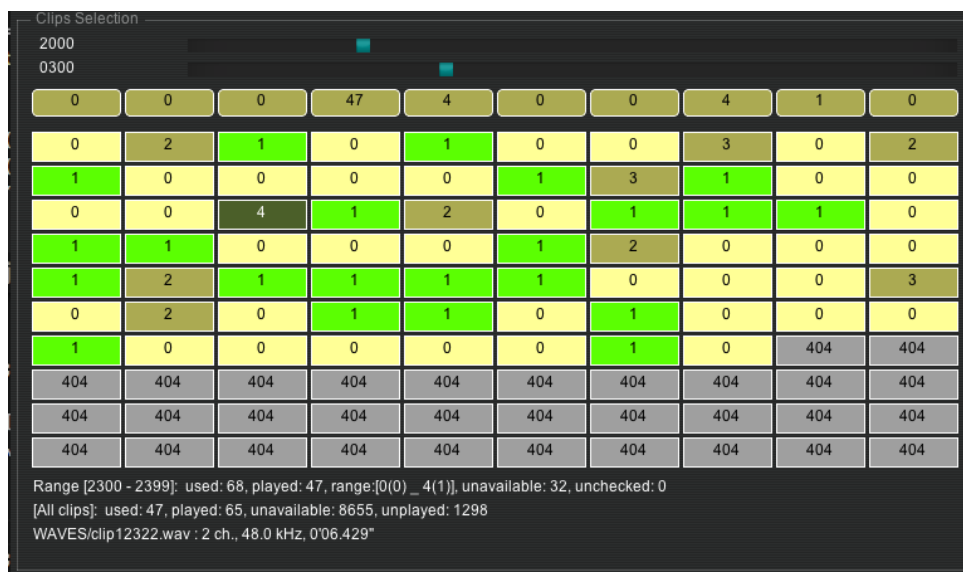


FIGURE 3.1 – Le module « Clips Selection »

Le module « Clips Selection » permet de configurer et d’explorer l’ensemble des clips disponibles dans un répertoire de fichiers approprié à un projet spécifique du Game Master.

Rappelons les concepts :

- chaque projet REAPER utilisant le Game Master peut disposer d’un répertoire spécifique, dans lequel peuvent être situés les clips pertinents à ce projet. Ce répertoire a pour nom « WAVES », et est situé dans le dossier « data »,

lui-même placé dans le répertoire du projet. Cette configuration permet de disposer d'un projet entièrement autonome, qui peut être installé n'importe où sans problème.

- ce répertoire contient des fichiers dont le nom obéit à la syntaxe suivante : « clip1nnnn.xxx », dans laquelle « nnnn » est un numéro à 4 chiffres, allant de « 0000 » à « 9999 », et « xxx » est l'extension associée au fichier son, qui peut être « wav », « aif » ou « aiff », « flac », « mp3 », etc., selon le format de ce dernier. Il peut donc y avoir jusqu'à 10000 fichiers dans un tel répertoire, chacun étant accessible aux différents lecteurs.
- si le projet ne fait pas appel à un répertoire spécifique, il peut utiliser un répertoire global (avec la même hiérarchie : « data/WAVES » placé dans le dossier des ressources de REAPER (répertoire accessible par le menu « Options▷ Show REAPER resource path in explorer/finder... »

3.2 Description

Le module comporte, en haut, deux sliders, permettant la sélection d'un groupe de 100 fichiers (le premier slider varie de 0000 à 9000 par pas de 1000, le second de 0000 à 0900 par pas de 100).

Le mini-pad d'une ligne, placé sous les sliders, indique, pour les dix groupes de 100 fichiers situés dans la même zone de 1000 déterminée par le premier slider, le nombre de fichiers joués dans chaque groupe. Ainsi, on peut voir qu'aucun fichier n'a été joué dans les trois premiers groupes (2000 à 2099, 2100 à 2199, 2200 à 2299), que 47 fichiers ont été joués dans le groupe 2300 à 2399, celui qui est affiché, que 4 l'ont été dans le groupe suivant, etc.

Ce groupe courant de 100 fichiers est affiché dans un pad de taille 10 par 10, les fichiers y étant implicitement numérotés de 00 à 99. Dans la capture d'écran de la figure 3.1 page précédente, les sliders définissent le groupe des fichiers de numéros compris entre 2300 et 2399, et l'affichage du groupe résume l'état des différents fichiers : le numéro « 404 » sur fond gris caractérise un fichier inexistant, un numéro sur fond rouge signale des erreurs ayant pu se produire lors de la lecture du fichier correspondant, et les numéros sur fond jaune à vert indiquent le nombre de fois où le fichier a été joué. Ainsi, le fichier 2301 a été joué 2 fois, le fichier 2307 trois fois, le fichier 2322, 4 fois, etc.

Si la touche « cmd » est enfoncée, ce sont les numéros de clips qui sont affichés dans le pad. On peut changer l'affichage par défaut par « shift-▲ » ou « shift-▼ » : numéros des clips, volumes par défaut, durées.

Positionner le pointeur de la souris sur une case affiche les caractéristiques du clip correspondant dans la dernière ligne du texte situé sous le pad. Un clic gauche va jouer ce clip, un clic droit va le sélectionner pour une opération ultérieure.

Notons que l'exploration de l'ensemble des clips est coûteuse en temps. Elle n'est pas exécutée immédiatement au lancement du Game Master, mais s'exécute, en parallèle du fonctionnement de celui-ci, au cours des premières minutes de l'exécution.

3.3 Interactions

- Les sliders permettent de choisir le groupe de 100 clips qui va être affiché.
- Dans la barre de sélection (pad de 10 cellules, situé sous les sliders) :
 - Clic-gauche dans l'un des pads affiche le groupe de 100 clips correspondant.
 - Clic-gauche+Ctrl : relance une exploration des clips ayant rencontré une erreur lors de l'exploration initiale.
 - Clic-gauche+Ctrl+Alt : relance une exploration systématique de tous les clips.
- Clavier :
 - Les flèches (« up », « down », « left », « right ») permettent de passer au groupe ou à la banque précédent(e) ou suivant(e).
- Dans le pad d'affichage du groupe :
 - Appuyer sur « cmd » affiche les numéros de clips.
 - Clic-gauche : joue le clip choisi avec les caractéristiques (play mode et space mode) du partiel 0.
 - Clic-gauche+Ctrl : joue et verrouille le clip choisi avec les caractéristiques (play mode et space mode) du partiel 0.
 - Clic-gauche+Alt : interrompt les clips en cours de jeu, et joue le clip choisi avec les caractéristiques (play mode et space mode) du partiel 0.
 - Clic-gauche+Alt+Shift : interrompt les clips en cours de jeu, et joue et verrouille le clip choisi avec les caractéristiques (play mode et space mode) du partiel 0.
 - Clic-gauche+Shift : remet à 0 les caractéristiques d'un clip. S'il doit être joué, il sera vérifié à nouveau.
 - Clic-gauche+Alt+Shift : remet à 0 toutes les caractéristiques des clips situés entre cette case et la position antérieurement sélectionnée par un « cd ».
 - Clic-gauche+Ctrl+Shift : active ou désactive le clip (un clip désactivé n'est pas joué, et affiche en rouge sombre « 999 »).
 - Clic-droit : sélectionne le clip correspondant.
 - Clic-droit+Cmd : force une revérification du clip.
 - Clic-droit+Ctrl+Shift : supprime tous les clips importés par glisser-déposer situés entre cette case et la position antérieurement sélectionnée par un « cd ». Les fichiers audio originaux ne sont naturellement pas modifiés.

- Clic-droit+Ctrl+Shift+Alt : remet à 0 les caractéristiques de l'ensemble des clips du système, et relance un check de ceux-ci. Ceci permet de changer le répertoire « WAVES », puis de vérifier l'ensemble des clips, sans avoir à relancer la session.

Clavier : il est possible, en n'utilisant que les touches du clavier, de se déplacer dans l'ensemble des clips et de faire appel à toutes les fonctions du module.

- Les flèches (« up », « down », « left », « right ») permettent de se « déplacer » dans la banque en sélectionnant le clip voisin (en haut, bas, etc.) du clip courant.
- Les flèches (« up », « down »), avec « shift », commutent les différents modes d'affichage.
- Espace ou « Return » : joue le clip choisi avec les caractéristiques (play mode et space mode) du partiel 0.
- Touches « 0 » à « 9 », « a » à « z », puis « Shift+A » à « Shift+Z » : jouent le clip choisi avec les caractéristiques (play mode et space mode) du partiel « N », « N » étant déduit de la touche choisie (0 à 9 pour les chiffres, « 10 » à « 35 » pour les minuscules, et « 36 » à « 61 » pour les majuscules), et donc compris entre « 0 » et « 61 ».
- Touche « : » : remet à 0 les caractéristiques du clip. S'il doit être joué, il sera vérifié à nouveau.
- Touche « ! » : supprime l'alias importé à cet emplacement.
- Touche « / » : remet à 0 les caractéristiques de chaque clip du bloc. Si un clip est joué, il sera vérifié à nouveau.
- Touche « = » : active ou désactive le clip (un clip désactivé n'est pas joué, et affiche en rouge sombre « 999 »).
- Touche « + » : active ou désactive chaque clip du bloc.

On notera qu'il est particulièrement pratique de définir explicitement des partiels (de numéro 0 à 63, c.f. § 13.7 page 210) avec des modes d'espace et des modes de jeu spécifiques. On peut alors sélectionner un clip (par « clic-droit »), puis par l'une des touches chiffre ou lettre, écouter de quelle manière il « sonne » avec les caractéristiques choisies pour ce partiel (et interrompre son jeu avec « suppr » ou « del »).

3.4 Glisser-déposer

La version 1.7.1 introduit le « glisser-déposer » de fichiers audio. Cette opération consiste à sélectionner, hors de REAPER, un ou plusieurs fichiers audio, dans le Finder du Mac ou dans l'Explorer de Windows, à déplacer le pointeur de la souris jusqu'à l'une des cent cases du groupe affiché dans le module. Relâcher le bouton de la souris : les fichiers choisis sont associés à ce numéro de clip et aux numéros de

clips suivants. Il est possible « d'importer » ainsi environ une centaine de clips par opération. Notons les aspects suivants :

- Les fichiers « clip1xxx.ttt » antérieurement associés aux numéros de clips choisis ne sont ni effacés ni modifiés. Ils sont simplement « masqués » par cette opération d'importation.
- Les fichiers « importés » ne sont ni modifiés, ni effacés, ni déplacés. L'opération consiste simplement à conserver un chemin d'accès à ces fichiers.
- Cette association temporaire disparaît à la fin de la session, sauf si celle-ci est sauvegardée par la commande « save » de REAPER. Il est également possible de sauvegarder un preset dans le GM, qui permettra, en le rechargeant, de retrouver ces importations.
- L'importation d'un nouveau fichier à l'emplacement d'un clip remplace toute importation antérieure.
- La suppression d'un alias fait réapparaître un clip « classique » éventuellement caché.

Chapitre 4

Le playlog

Comme signalé antérieurement, il est possible de conserver l'ensemble des paramètres associés à un son sous une forme relativement compacte, et ainsi de reproduire ultérieurement, à l'identique, un son ou une séquence de sons. C'est le but du *playlog*. Par défaut, le Play Log est activé au début de la session. Il se comporte comme un axe temporel, sur lequel sont notés les sons joués avec leur date et leurs caractéristiques. Un lecteur permet d'explorer cet axe, à des vitesses comprises entre -10 et 10 fois la vitesse « normale ». Il est également possible de « noter » à la volée des instants, et d'y revenir ultérieurement.

Ses différents aspects sont accessibles au travers du module « Play Log » (fig. 4.1 page suivante).

4.1 Concepts généraux

L'*enregistreur*, lorsqu'il est actif, conserve en mémoire partagée tous les paramètres essentiels concernant un son (environ 80 valeurs). Il leur associe une date d'enregistrement. Le temps de l'enregistrement progresse à la vitesse relative « 1 » en mode enregistrement, mais ne varie pas lorsque l'enregistreur est en pause ou à l'arrêt. L'on pourrait assimiler cet axe des temps à une « piste » d'un enregistreur MIDI, qui conserverait beaucoup plus d'informations que de simples données MIDI. Notons que les sons déclenchés en mode « stop » ou « pause » ne sont pas enregistrés. Le GameMaster peut conserver jusqu'à 80000 sons ainsi représentés, ce qui correspond à une session de plusieurs dizaines d'heures. Il n'est pas possible actuellement de sauvegarder ces sons sous cette forme, mais il est loisible de préparer une séquence, et de l'enregistrer sous forme audio au moyen de la commande « Save live output to disk... » de REAPER.

Le *lecteur*, lorsqu'il est actif, va suivre cette « piste », et déclencher le jeu des sons au fur et à mesure qu'il les rencontre. Le lecteur peut fonctionner à vitesse normale,

accélérée ou ralentie, voire en sens inverse. Cette « vitesse » ne change ni la hauteur, ni la durée des sons joués ; il faut la considérer comme un « tempo », qui n'agit que sur l'intervalle temporel entre les sons. Le lecteur est indépendant de l'enregistreur, et les deux peuvent être actifs simultanément.

Enfin, à tout moment, soit pendant l'enregistrement, soit pendant la lecture, il est possible de « marquer » un point précis de la piste. Ces « *marqueurs* » jouent un double rôle, car ils conservent également la liste des sons joués à cet instant, qu'il sera possible de rejouer ultérieurement. On peut sauter, en lecture, vers n'importe quel marqueur, ou encore utiliser une paire de marqueurs pour délimiter une boucle de lecture.

4.2 Interface utilisateur

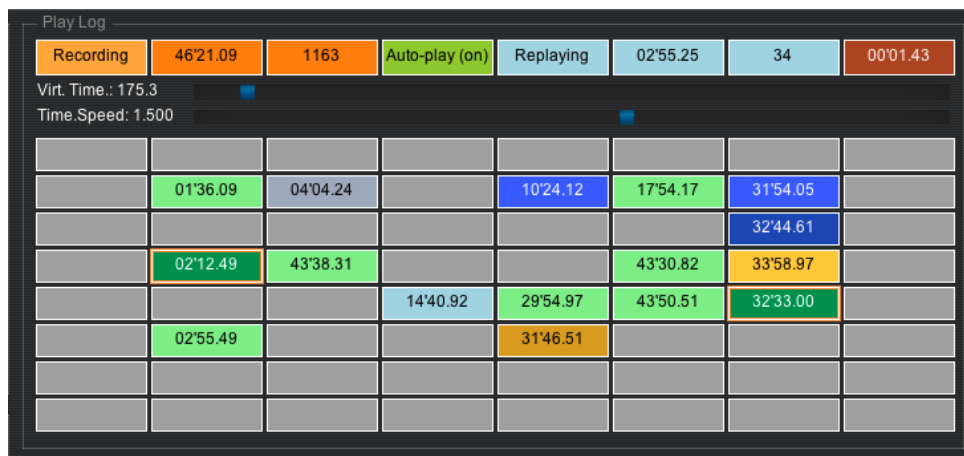


FIGURE 4.1 – Le module « Play Log »

La figure 4.1 décrit, de haut en bas, les trois éléments qui composent l'interface avec le play log :

- un premier pad, dit « *pad principal* », qui comporte les diverses commandes (boutons et afficheurs)
- deux sliders, qui permettent de régler la position et la vitesse du « replay »
- un second pad, dans lequel on va pouvoir noter jusqu'à 64 marqueurs.

4.3 Commandes d'enregistrement et de rejouage

Ces fonctions sont assurées par le premier pad de huit cellules, dont les rôles sont les suivants :

1. Mode d'enregistrement. Cette cellule affiche trois états : « Stopped », « Paused » et « Recording ». Un clic-gauche assure le passage du mode « Recording » à « Paused », et d'un autre mode à « Recording ». Un « cmd-cg » (clic-gauche avec la touche « cmd ») a la même action, mais applique en plus cette action au mode rejouage (c.f. cellule 5), ce qui permet de synchroniser les deux fonctions. Un « cd » (clic-droit) affiche un menu, grâce auquel on peut choisir l'un des trois modes, et deux options :

Lock sounds on pause cette option a pour effet de verrouiller les sons qui jouent lorsque l'on passe en mode pause. Les sons sont déverrouillés lorsque l'on repasse en mode « Recording ».

Record replayed sounds indique si l'on doit, ou non, lorsque l'on est en mode « Recording », enregistrer également les sons générés par le mode « Replay ».

2. Record time. La valeur affichée correspond à la durée d'enregistrement. Elle ne progresse que lorsque la cellule (1) est en mode « Recording ». Un « cg » passe de l'affichage en heures-minutes-secondes à secondes-milisecondes et réciproquement. Cette commutation s'applique à toutes les cellules affichant une durée. Un « cd » commute l'option « Locks sounds on pause ».
3. Item count. Affichage du nombre effectif de sons enregistrés depuis le début de la session. Un « cd » commute le mode « Record replayed sounds ».
4. Auto-Play. Cette cellule reproduit la cellule correspondante du module « play control ». Un clic gauche permet de passer l'auto-play de « on » à « off » et réciproquement. Un « cmd-cg » coupe tous les sons. Un « cd » coupe tous les sons et arrête l'auto-play.
5. Mode de rejouage. Cette cellule affiche trois états : « Stopped », « Paused » et « Replaying ». Un clic-gauche assure le passage du mode « Replaying » à « Paused », et d'un autre mode à « Replaying ». Un clic droit affiche un menu, qui permet de choisir l'un des trois modes, « Rewind » qui remet à 0 la date de lecture, et diverses options :

Auto stop cette option a pour effet de passer le lecteur en « Stop » lorsque le dernier son de la piste a été joué. Dans le cas contraire, la lecture (silencieuse, donc) se poursuit jusqu'au bout de la piste enregistrée.

Pause after each sound lorsque cette option est activée, le lecteur passe en pause après chaque nouveau son joué.

Lock replayed sounds (K) verrouille chaque son rejoué.

Mute replayed sounds (M) cette option inhibe le rejouage des sons

Markers include replayed sounds (G) lorsque cette option est sélectionnée, les sons rejoués par le rejoueur sont aussi enregistrés par l'enregistreur. A

noter que cette option utilise peu d'espace, car elle génère simplement plusieurs références au même son enregistré.

Step mode (S) si cette option est activée, le lecteur saute immédiatement au son suivant, le rejoue, puis s'arrête.

Immediate jump into loop (J) si cette option est activée, le lecteur saute immédiatement en début de boucle dès que celle-ci est activée, au lieu d'attendre que la lecture atteigne la boucle.

Skip long silences (E) Lorsque cette option est activée, les silences de plus de 30 secondes environ sont sautés lors de la lecture.

Repeat mode (L) arrivé en fin de piste, le lecteur, au lieu de s'interrompre, repart au début.

6. Play time. La valeur affichée correspond à la position du lecteur dans la piste. Un « cg » passe de l'affichage en heures-minutes-secondes à secondes-milisecondes et réciproquement. Cette commutation s'applique à toutes les cellules affichant une durée. Un « cd » commute le mode « Repeat ». Dans ce mode, le rejouage repart du début lors de l'arrivée en fin de piste.
7. Item number. La valeur affichée est le numéro d'enregistrement du son qui va être joué en mode « replay », ou « 0 » si aucun son n'est à jouer. Un « cd » commute le mode « Marqueurs include replayed sounds ».
8. Next play. Cette cellule indique dans combien de temps le prochain son va être rejoué en mode « replay ». Cette valeur est une durée réelle, et tient compte de la vitesse de rejouage. Un « cd » commute le mode « Play Control ». Dans ce mode, le réglage « Max Players » permet de limiter le nombre maximal de players actifs à un moment donné. La cellule est de couleur bleu en mode « replay », vert en mode « Play control », brun en mode « Replay mute ».

4.4 Réglages de position et de vitesse

Sous ce premier pad, deux potentiomètres permettent de régler dans la piste enregistrée la « position » de la « tête de lecture » (« Virtual Time »), ainsi que la vitesse de lecture (« Time Speed »).

4.4.1 Le temps virtuel

Il s'agit ici de la date associée au lecteur. La valeur affichée est en secondes (mais aussi inscrite dans la cellule 6 du premier pad). Elle est réglable de 0 secondes jusqu'à la taille de la piste enregistrée. Après toute modification, la lecture se poursuit à la nouvelle valeur sélectionnée, celle-ci évoluant au fur et à mesure de la lecture.

Lorsque la souris est positionnée sur le « label » du réglage, à gauche du slider proprement dit, les commandes suivantes peuvent être utilisées :

- cd ou cg Le clic droit (resp. clic gauche) avance (resp. retarde) la position temporelle de 10 secondes.
- cmd-cd ou cmd-cg Associé à la touche « cmd », le clic droit (resp. clic gauche) avance (resp. retarde) la position temporelle de 60 secondes.
- alt-cd ou alt-cg Associé à la touche « alt », le clic droit (resp. clic gauche) avance (resp. retarde) la position temporelle d'une seconde.
- cmd-alt-cd ou cmd-alt-cg Associé aux touches « cmd » et « alt », le clic droit (resp. clic gauche) avance (resp. retarde) la position temporelle d'un dixième de seconde.

4.4.2 La vitesse temporelle

Ce second réglage affiche le rapport entre le temps de la lecture et le temps réel. Une valeur de « 1 » indique que la lecture s'exécute à la même vitesse que l'enregistrement, « 2 » signifie qu'elle se produit deux fois plus vite, « 0.5 » deux fois plus lentement, et « -1 » en vitesse inverse (la « partition » est en quelque sorte lue à l'envers, mais les sons ne sont pas transformés). Ce rapport de temps de lecture est réglable entre -10 et +10.

Lorsque la souris est positionnée sur le « label » du réglage, les commandes suivantes peuvent être utilisées :

- cd ou cg Le clic droit (resp. clic gauche) ajoute (resp. retranche) 0.1 au facteur « Time speed ».
- cmd-cd ou cmd-cg Associé à la touche « cmd », le clic droit (resp. clic gauche) ajoute (resp. retranche) 1 au facteur « Time speed ».
- alt-cd ou alt-cg Associé à la touche « alt », le clic droit (resp. clic gauche) ajoute (resp. retranche) 0.01 au facteur « Time speed ».
- cmd-alt-cd ou cmd-alt-cg Associé aux touches « cmd » et « alt », le clic droit (resp. clic gauche) positionne à « 1 » (resp. « -1 ») le facteur « Time speed ».

4.5 Les marqueurs

Le second pad du module « Play Log » est de taille 8 par 8 ; chaque cellule peut contenir un « marqueur », c'est à dire la désignation temporelle d'un instant de l'enregistrement. Durant celui-ci, il suffit de cliquer sur une case libre pour créer un marqueur, qui repère l'instant précis où l'on a cliqué. Dans la figure 4.1 page 92, le

pad a été peuplé de quelques makers. Chacun d'entre eux indique la date de création (en heure, minutes, secondes et centisecondes).

Pour créer un marqueur, il suffit de cliquer dans une case inutilisée du pad. La date de création est alors inscrite dans la cellule, et le marqueur est coloré en vert clair. La « date » utilisée est celle de l'enregistreur (case 3) si celui-ci est en « Recording » ou en « Pause », la date du lecteur sinon. Une paire de marqueurs peut être utilisée pour représenter les limites d'une boucle de lecture, qui sera répétée jusqu'à suppression de la boucle.

Lors de la création du marqueur sont également enregistrées les références aux sons en train d'être joués. Il est possible de modifier la finalité du marqueur, pour le transformer en un bouton déclenchant certains des sons ainsi enregistrés. Très précisément, le marqueur peut être utilisé pour jouer un son spécifique, un son au hasard, un son en « round robin », la séquence, ou un « ensemble », c'est-à-dire tous les sons simultanément.

4.5.1 Opérations

Voici une liste des opérations disponibles (sauf précisé explicitement, ces opérations s'appliquent à des marqueurs, et sont ineffectives pour des cases vides).

- | | |
|------------|--|
| cg | Dans une case vierge, crée un marqueur, et lui associe la date de l'enregistreur en mode « Recording » ou « Pause », la date du lecteur sinon. Dans une case utilisée, va positionner le lecteur à la date correspondante, ou va déclencher le jeu de sons en fonction du type du marqueur. |
| cmd-cg | Dans une case vierge, crée un marqueur, et lui associe la date de lecture. Dans une case utilisée, positionne le lecteur à la date correspondante, quel que soit le type du marqueur cliqué. |
| ctl-cg | crée un marqueur dans cette position, en remplaçant éventuellement le marqueur préexistant. |
| ctl-alt-cg | supprime le marqueur courant. |
| cmd-alt-cg | change temporairement le marqueur courant en limite de boucle. « cmd-alt-cg » restitue la fonction antérieure. Il n'y a au maximum que deux marqueurs de boucle, en marquer un troisième désactive le plus ancien. Le mode bouclage est activé lorsque deux marqueurs de boucle sont actifs. |
| cd | Dans une case vierge : passe de « Replay » en « Pause » et réciproquement. Dans une case marquée, ouvre un menu, avec diverses rubriques :

Type ▸ Inactive conserve le marqueur, mais le rend inactif. |

- Type** ▷ **Mark Point** le marqueur a pour fonction d'indiquer une position dans la piste des sons enregistrés, et de positionner la tête de lecture à cet emplacement lorsqu'il est cliqué (option par défaut).
- Type** ▷ **Single Sound** cliquer le marqueur déclenche un son unique de la liste des sons associés au marqueur, par défaut le premier.
- Type** ▷ **Random Sound** cliquer le marqueur déclenche un son au hasard de la liste des sons associés au marqueur.
- Type** ▷ **Round Robin** cliquer le marqueur déclenche le son « suivant » de la liste des sons associés au marqueur.
- Type** ▷ **Ensemble** cliquer le marqueur déclenche simultanément l'ensemble des sons de la liste.
- Type** ▷ **Sequence** cliquer le marqueur lance la séquence chronologique des sons de la liste.
- Type** ▷ **Change type (Y)** change le type du marqueur en passant cycliquement de « inactif » à « sequence ».
- Date** ▷ **Mark date** choisit (option par défaut) la date de création du marqueur comme date associée à celui-ci.
- Date** ▷ **First item** choisit la date de jeu du premier item (par ordre chronologique) comme date associée au marqueur.
- Date** ▷ **Last item** choisit la date de jeu du dernier item (par ordre chronologique) comme date associée au marqueur.
- Date** ▷ **Sequence end** choisit la date de la fin de la sequence (incluant le silence suivant le dernier item) comme date associée au marqueur.
- Lay out** ▷ **Move left** déplace le marqueur courant (par échange éventuel) vers la gauche.
- Lay out** ▷ **Move right** déplace le marqueur courant (par échange éventuel) vers la droite.
- Lay out** ▷ **Move up** déplace le marqueur courant (par échange éventuel) vers le haut.
- Lay out** ▷ **Move down** déplace le marqueur courant (par échange éventuel) vers le bas.
- Lay out** ▷ **Push left** déplace le marqueur courant vers la gauche (en repoussant éventuellement les marqueurs contigus).
- Lay out** ▷ **Push right** déplace le marqueur courant vers la droite (en repoussant éventuellement les marqueurs contigus).
- Lay out** ▷ **Duplicate (D)** copie le marqueur dans la première cellule libre à sa droite.

Lay out ▷ **Cut (X)** supprime le marqueur courant (qui peut être ultérieurement collé dans une autre cellule).

Lay out ▷ **Copy (C)** copie le marqueur courant.

Lay out ▷ **Paste (V)** colle le marqueur courant dans la cellule.

Option ▷ **Toggle inactive (N)** rend inactif le marqueur sans perdre aucune des informations qui lui sont associées.

Option ▷ **Toggle loop point (F)** change le marqueur en *limite de boucle* sans perdre aucune des informations qui lui sont associées. Il ne peut y avoir que deux limites.

Edit ▷ **xxx** propose 24 sous-menus pour éditer la date associée au marqueur, permettant d'ajouter ou de soustraire de 1ms à 30' à cette date. Ces fonctions sont décrites au § 4.5.2.

cmd-cd déclenche tous les marqueurs correspondant à cette case et aux cases situées en dessous, dans la même colonne, jusqu'à la première case vide ou jusqu'au bas de la colonne. La première case rencontrée de type « Mark Point » lance la lecture à la position correspondante, les autres « Mark Point » sont ignorés.

ctrl-cd déclenche tous les marqueurs correspondant à cette case et aux cases situées à droite de celle-ci, dans la même ligne, jusqu'à la première case vide ou la fin de la ligne. La première case rencontrée de type « Mark Point » lance la lecture à la position correspondante, les autres « Mark Point » sont ignorés.

4.5.2 Edition des marqueurs

Lorsque la souris est dans une case correspondant à un marqueur, il est possible d'éditer la valeur de celui-ci (c'est-à-dire la date désignée par le marqueur) avec les caractères suivants introduits au clavier :

'▶' incrémente la valeur du marqueur d'une seconde.

'◀' décrémente la valeur du marqueur d'une seconde.

cmd-'▶' incrémente la valeur du marqueur d'un dixième de seconde.

cmd-'◀' décrémente la valeur du marqueur d'un dixième de seconde.

alt-'▶' incrémente la valeur du marqueur d'un centième de seconde.

alt-'◀' décrémente la valeur du marqueur d'un centième de seconde.

cmd-alt-'▶' incrémente la valeur du marqueur d'un millième de seconde.

cmd-alt-'◀' décrémente la valeur du marqueur d'un millième de seconde.

'▲'	incrémente la valeur du marqueur de dix secondes.
'▼'	décrémente la valeur du marqueur de dix secondes.
cmd-'▲'	incrémente la valeur du marqueur d'une minute.
cmd-'▼'	décrémente la valeur du marqueur d'une minute.
alt-'▲'	incrémente la valeur du marqueur de cinq minutes.
alt-'▼'	décrémente la valeur du marqueur de cinq minutes.
cmd-alt-'▲'	incrémente la valeur du marqueur de trente minutes.
cmd-alt-'▼'	décrémente la valeur du marqueur de trente minutes.
'='	arrondit la valeur du marqueur à un nombre entier de secondes.
cmd-'='	arrondit la valeur du marqueur à un nombre entier de dixièmes de secondes.
alt-'='	arrondit la valeur du marqueur à un nombre entier de centièmes de secondes.
cmd-alt-'='	arrondit la valeur du marqueur à un nombre entier de millièmes de secondes.
':'	arrondit la valeur du marqueur à un multiple de 10 secondes.
cmd-':'	arrondit la valeur du marqueur à un multiple d'une minute.
alt-':'	arrondit la valeur du marqueur à un multiple de cinq minutes.
cmd-alt-':'	arrondit la valeur du marqueur à un multiple de trente minutes.
'c'	copie le marqueur courant.
'd'	duplique le marqueur courant dans la première case disponible à sa suite.
'f'	change le marqueur courant en limite de boucle.
'n'	rend « neutre » le marqueur courant. Un nouveau caractère « n » lui rend sa signification antérieure. La cellule ainsi marquée prend une couleur gris bleuté.
'b'	rend à un marqueur « neutre » sa fonctionnalité antérieure.
'x'	coupe le marqueur courant.
'v'	colle (dans une case vierge) le dernier marqueur copié ou coupé.
'y'	change le type du marqueur en passant cycliquement de « inactif » à « sequence ».

4.6 Autres commandes

Notons que, lorsque la souris est dans ce module certaines touches du clavier sont interprétées comme des commandes :

'a'	inverse l'option « Auto play ». Modifie la couleur de la cellule (4) du <i>pad principal</i> .
'h'	interrompt l'enregistrement et la lecture. Les couleurs des cellules (1), (4) et (5) sont modifiées.
'i'	inverse le sens de la lecture du replay.
'j'	inverse l'option « Immediate jump into loop ». Modifie la couleur de la cellule (5) en mode loop.
'k'	inverse l'option « Lock replayed sounds ».
'l'	inverse l'option « Repeat mode ». Modifie la couleur de la case (6).
'm'	inverse l'option « Mute replay », qui permet de muter uniquement les sons du mode replay. Modifie la couleur de la case (8).
'o'	inverse l'option « Locks sounds on pause ». Modifie la couleur de la case (1).
'p'	bascule de replay à pause et réciproquement. Permet de passer au son suivant en mode « Step ». En mode « Step », la cellule (5) affiche une couleur particulière.
'q'	permet de répéter le son précédent en mode « Step ».
'r'	bascule du mode enregistrement au mode pause et réciproquement.
's'	inverse l'option « Step mode ».
't'	inverse l'option « Record replayed sounds ». Modifie la couleur de la case (3).
'u'	inverse l'option « Unlock replayed sounds ».
'w'	alterne différentes vitesse de lecture entre -2.5 et 2.5.
'z'	interrompt enregistrement, lecture, et production de son.

Chapitre 5

Les Paramètres Dynamiques de Jeu

La version 1.7.0, au travers d'une réécriture complète des « players », introduit une formalisation de certains paramètres de jeu, qui effectuent une interaction directe avec les sons en cours d'écoute. L'idée est de permettre au compositeur de disposer d'un ensemble de paramètres dont il peut juger immédiatement de l'influence sur le son en cours d'audition. Ces paramètres peuvent naturellement être utilisés au cours d'une performance « live ».

Distinguons trois moments dans la spécification des sons :

1. Lors de la création des fichiers de description des configurations de HP, modes d'espaces, modes de jeu, et définition des banques, il est possible de prévoir tout ou partie des paramètres de jeu d'un clip. Certains peuvent rester indéfinis, d'autres s'expriment sous forme d'un intervalle de valeurs, dans lequel une valeur unique sera choisie au moment du jeu.
2. Lors du lancement d'un clip, les paramètres non définis sont calculés à partir des réglages des paramètres courants, ou sont choisis dans les intervalles de valeurs selon certains critères.
3. Au cours du jeu d'un clip, de nouveaux paramètres entrent en action : les *modificateurs*, ou *modulateurs de jeu*, qui sont interactifs et agissent immédiatement sur le rendu sonore. C'était le cas du paramètre « Dynamic Rate » qui intervient pour modifier la vitesse de lecture des clips : une valeur inférieure à 1 diminue cette vitesse, une valeur supérieure à 1 l'augmente.

C'est dans cette dernière catégorie que se rangent un certain nombre de nouveaux paramètres, dont la description constitue le corps de ce chapitre.

5.1 Note sur les « grains »

Le mode « loop » qui génère des boucles à partir d'un fichier son correspond dans les faits, pour certains réglages, à une synthèse granulaire. Il est possible en effet

de faire varier la longueur de ces boucles de moins d'une milliseconde à plusieurs minutes (la boucle pouvant alors être plus longue que le fichier source, repartant si besoin au début de ce dernier). Une « courte » boucle (moins d'une seconde) sera dite « grain » pour reprendre ce terme propre à la synthèse granulaire.

Un grain est donc un fragment de fichier source, modulé par une enveloppe de type « ASR », pour « Attack, Sustain, Release ». Un volume relatif est associé au grain, qui passe de 0 à 1 durant la phase « Attack », demeure à 1 lors de la phase « Sustain », et repasse de 1 à 0 durant la phase « Release ». Chacune de ces trois phases a une durée réglable au moyen de deux paramètres, qui sont le « plateau » et « l'inclinaison ». Le plateau définit la durée de la phase « Sustain », sous la forme d'un pourcentage de la durée total du grain. « L'inclinaison » est également un pourcentage, qui définit le rapport entre la durée de la phase « Attack » et celle de la phase « Release ». Si nous imaginons un grain d'une durée de 1 seconde, la valeur du plateau à 50% correspond donc à un « Sustain » de 0.5 secondes, ce qui laisse également 0.5 secondes pour l'attaque et la release. Une valeur de « l'inclinaison » de 50% correspondra à une durée égale à 0.25 secondes pour les phases d'attaque et de release. Une valeur de 30% pour l'inclinaison correspondra à une attaque de 0.15 secondes et une release de 0.35 secondes.

Les formes des courbes correspondant aux phases d'attaque et de release d'un grain sont elles-mêmes variables, définies par les paramètres « shape In » et « shape Out », qui font varier ces formes de concaves à convexes.

Enfin, le point de début d'un grain (d'une boucle) est fixé par le paramètre « Position Dynamique », dont la valeur (un pourcentage) permet de fixer un point de référence dans le clip, une autre valeur « Variation de position » permettant de fixer l'intervalle, autour de ce point de référence, dans lequel va être choisi le point de départ du clip. Pour une valeur de 0% de cette variation, le point de départ de la boucle est figé au point de référence, pour une valeur de 100% de cette variation, l'intervalle dans lequel est choisi le point de départ recouvre la totalité du clip. Pour toute valeur intermédiaire, par exemple 20%, l'intervalle dans lequel est choisi le point de départ du clip recouvre une partie de celui-ci, autour de la position dynamique.

5.2 Note sur les modes de jeu

Dans les fichiers de description des modes de jeu il est possible de laisser des paramètres « indéfinis ». Dans les versions antérieures du Game Master, ces paramètres recevaient une valeur par défaut. A partir de la version 1.7, ces paramètres restent indéfinis jusqu'au moment du jeu du clip utilisant ces modes de jeu ou d'espace. Ils prennent alors une valeur par défaut, qui dépend des réglages courants et de certains paramètres dynamiques. Pour les modes de jeux, une « facette » spécifique décrit de quelle manière ces choix sont effectués. Cette facette est introduite par le mot-clef

« FLM » (pour « Flags Modifiers »), et est définie par deux valeurs numériques entières, « FLM1 » et « FLM2 ». Ces valeurs sont des combinaisons logiques de flags, et se prêtent bien à une notation en hexadécimal. Si un indicateur est « positionné » (sa valeur n'est pas nulle), ceci signifie que la valeur correspondante va être mise à jour dynamiquement. Dans le cas contraire, une valeur par défaut est utilisée. Une facette « FLM » est donc une combinaison d'indicateurs, qui ont la signification suivante :

- 0x1** le volume du jeu du clip va être mis à jour dynamiquement
- 0x2** inutilisé
- 0x4** la vitesse de jeu du clip, ou des grains du clip, va être mise à jour dynamiquement
- 0x8** inutilisé
- 0x10** La taille des grains va être mise à jour dynamiquement
- 0x20** « L'inclinaison » du grain va être mise à jour dynamiquement
- 0x40** inutilisé
- 0x80** La valeur du « plateau » du grain va être mise à jour dynamiquement
- 0x100** Le point de départ de la lecture du grain est dynamique
- 0x200** La variation autour du point de lecture du grain est dynamique
- 0x400** La « masse » du nuage de grains généré est dynamique
- 0x800** L'asynchronisme de la génération des grains est dynamique
- 0x1000** Le sens de lecture de chaque grain est dynamiquement défini
- 0x2000** L'inharmonicité (vitesse de lecture) des grains individuels est dynamiquement variable
- 0x4000** L'algorithme régissant cette inharmonicité est dynamiquement redéfinissable
- 0x8000** La forme du Fade In des grains est dynamiquement redéfinissable
- 0x10000** La forme du Fade Out des grains est dynamiquement redéfinissable

Les deux valeurs « FLM1 » et « FLM2 » obéissent à la même définition. « FLM1 » est utilisée pour définir les paramètres dynamiques à la création du son, « FLM2 » est utilisée en cours de jeu, permettant toutes les transformations dynamiques désirées. Ainsi, un même drapeau peut être positionné à « 1 » dans « FLM1 », indiquant que la facette correspondante va prendre la valeur du paramètre correspondant au moment du lancement du clip, tout en étant positionné à « 0 » dans « FLM2 », indiquant que la facette ne sera pas modifiée durant le jeu du clip.

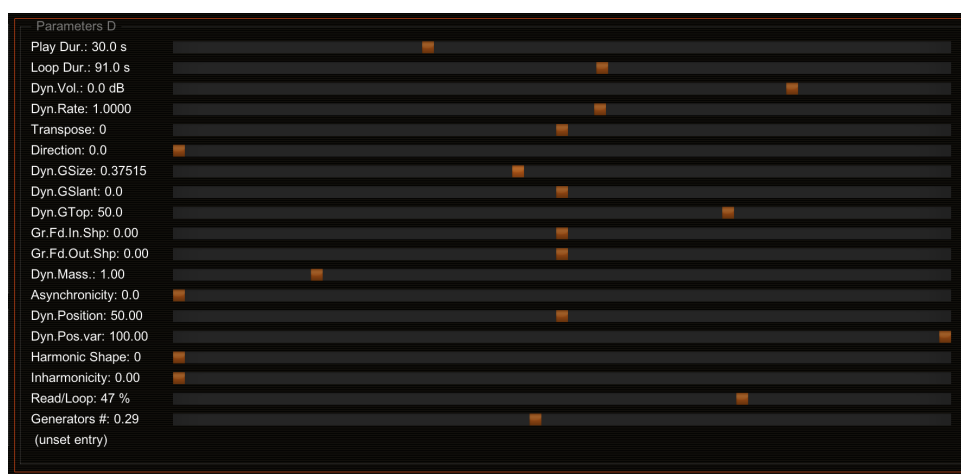


FIGURE 5.1 – Affichage des paramètres dynamiques de jeu

5.3 Les paramètres et leurs actions

Une dizaine de nouveaux paramètres ont été définis, et quelques anciens paramètres ont vu leurs spécifications éclaircies. Ils permettent des ajustements ou des variations en temps réel de valeurs définies par les modes de jeu et les modes d'espace. Les paramètres dynamiques ont tous une valeur par défaut, qui ne modifie pas les choix effectués lors de la définition du fichier des modes de jeu. La figure 5.1 montre l'affichage de ces paramètres.

Dans la description ci-dessous, le terme « usage » suivi d'un ou plusieurs numéros indique à quel moment (1, 2 et 3 ci-dessus) ces paramètres sont pris en compte. La présence de la mention « flm » indique que l'utilisation du paramètre est conditionnée par les valeurs des facettes « FLM1 » et « FLM2 ».

Play Dur. (« Play duration »). Exprimé en secondes. Durée du jeu d'un clip en mode « Play », exprimée en secondes. Cette valeur est utilisée si la durée de jeu d'un clip n'est pas fixée dans le mode de jeu. Cette valeur est strictement respectée par les players. Usage : 2.

Loop Dur. (« Loop duration »). Exprimé en secondes. Durée du jeu d'un clip en mode « Loop », exprimée en secondes. Cette valeur est utilisée si la durée de jeu d'un clip n'est pas fixée dans le mode de jeu. Cette valeur est strictement respectée par les players. Usage : 2.

Vol. Glob. (« Volume Global »). Exprimé en dB. Modifie le volume de l'ensemble des sons générés. Usage : 2, 3.

Dyn. Vol. (« Volume Dynamique »). Exprimé en dB. Modifie le volume des sons générés par les players. Usage : flm, 2, 3. Defaut : 0dB.

Dyn. Rate (« Vitesse de lecture »). Facteur sans dimension [0.01 ... 10]. Modifie la vitesse de lecture du clip ou des grains du clip. Usage : flm, 2, 3. Default : 1.

Transpose (« Transposition »). Facteur entier sans dimension [-24 ... 24]. Modifie par demi-tons la vitesse de lecture du clip ou des grains du clip. Usage : flm, 2, 3. Default : 0.

Direction (« Sens de lecture »). Probabilité (de 0 à 1) d'inversion du sens de lecture de chaque grain individuel. Usage : flm, 2, 3. Default : 0.

Dyn. Gsize (« Taille du grain »). Facteur sans dimension [0.001 ... 100]. Modifie la taille des grains générés. Usage : flm, 2, 3. Default : 1.

Dyn. Top (« Plateau »). Pourcentage. Modifie le plateau des grains générés. Usage : flm, 2, 3. Default : 50%.

Dyn. Slant (« Inclinaison »). Pourcentage. Modifie l'inclinaison des grains générés. Usage : flm, 2, 3. Default : 50%.

Gr.Fd.In.Shp (« Forme de fade-in des grains »). Réglable de -1 (forme très concave) à 1 (forme très convexe) en passant par 0 (montée linéaire). Usage : flm, 2, 3. Default : 0.

Gr.Fd.Out.Shp (« Forme de fade-out des grains »). Réglable de -1 (forme très concave) à 1 (forme très convexe) en passant par 0 (descente linéaire). Usage : flm, 2, 3. Default : 0.

Dyn. Position (« Point de référence »). Pourcentage. Usage : flm, 2, 3. Default : 50%.

Dyn.Pos.var (« Variation autour du point de référence »). Pourcentage. Usage : flm, 2, 3. Default : 100%.

Dyn. Mass (« Densité du nuage de grains »). Facteur sans dimension [0.01 ... 100], multipliant la valeur initiale définie par la facette « GDepth » du mode de jeu. Usage : flm, 2, 3. Default : 1.

Asynchronicity Pourcentage. Ne modifie pas la densité, mais agit sur le synchronisme de la génération des grains. Usage : flm, 2, 3. Default : 0.

Harmonic Shape Numéro (0 à 10) d'un algorithme de transformation des grains. Expérimental. Usage : flm, 2, 3. Default : 0.

Inharmonicity Paramètre (Pourcentage) utilisé par certains algorithmes évoqués ci-dessus. Usage : flm, 2, 3. Default : 0.

Tous ces aspects peuvent être définis de manière indépendante au moyen de sliders (c.f. fig. 5.1 page ci-contre), mais aussi rassemblés et manipulés en une structure unique, les « paramètres dynamiques de jeu ». Ces « préréglages » peuvent être créés dynamiquement, conservés, édités et modifiés, chargés pour modifier instantanément

les jeu des clips courants, mais aussi associés spécifiquement à un clip particulier et conservés dans l'historique (c.f. la description du module « Playlog », 4 page 91).

Note Tous ces paramètres ont une influence sur le timbre des sons obtenus, même si dans certains cas cette influence est à peine perceptible. L'idéal est de travailler sur un seul son à la fois, jusqu'à obtenir le timbre recherché, puis d'enregistrer les paramètres au moyen du module « *Dynamic Play Parameters* ». La commande « s » (solo) du module « *Sound Units* » permet précisément de placer tous les players en attente, en écoutant le seul son sélectionné.

5.4 Le module « Dynamic Play Parameters »

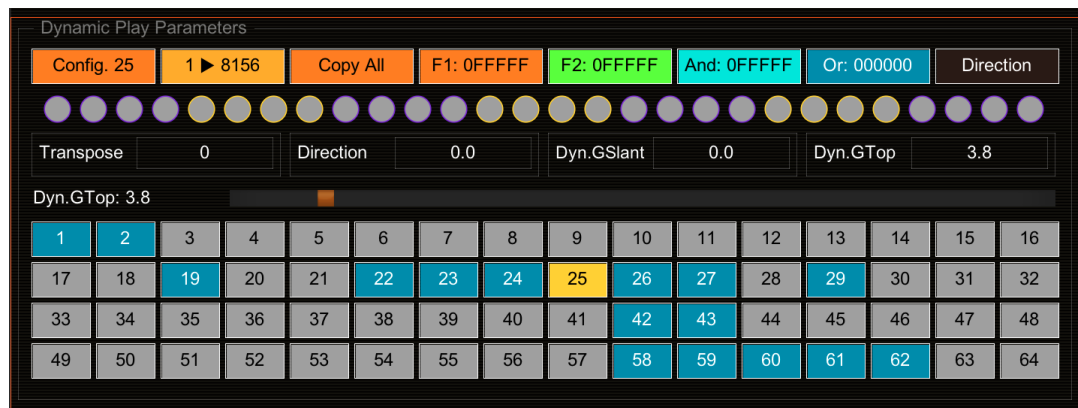


FIGURE 5.2 – Le module « Dynamic Play Parameters »

Ce module, dont la figure 5.2 donne l'aspect, va nous proposer nombre d'outils pour la gestion des paramètres dynamiques.

Il se compose de cinq zones, disposées du haut vers le bas :

1. Le *pad principal* contient 8 cellules, qui vont permettre de réaliser différentes interactions avec les paramètres, les préréglages, et les clips en cours de jeu. Ces cellules représentent, de gauche à droite :
 - (a) Affichage du numéro de la configuration courante (c.f. 5.4.4 page 108).
 - (b) Référence au player et au clip en mode édition sous la forme [player►clip].
 - (c) Mode de transfert des paramètres.
 - (d) Valeur du paramètre « FLM1 » du préréglage courant (en hexadécimal).
 - (e) Valeur du paramètre « FLM2 » du préréglage courant (en hexadécimal).

- (f) Valeur du masque de transfert « And ».
- (g) Valeur du masque de transfert « Or ».
- (h) Nom du flag pointé par la souris dans *L'éditeur de drapeaux*.
- 2. *L'éditeur de drapeaux*, pad contenant 28 cellules circulaires, groupées 4 par 4, permettant d'éditer aisément un ensemble de drapeaux (flags) présenté sous forme d'un nombre hexadécimal.
- 3. *L'afficheur de paramètres*. Quatre sont visibles simultanément, les touches ◀ et ▶ pouvant les faire défiler.
- 4. Un *slider*, pour éditer le paramètre choisi.
- 5. Le *pad des préréglages*, proposant 64 préréglages différents, numérotés de 1 à 64.

5.4.1 Manipulation des préréglages

L'outil dont l'utilité immédiate est la plus évidente est probablement le *pad des préréglages*. Il suffit d'un clic gauche dans une case inutilisée (grise) pour capturer l'ensemble des paramètres dynamiques. La case change de couleur pour indiquer qu'elle a été utilisée. Ultérieurement, un clic droit dans cette case va recharger l'ensemble des paramètres dynamiques à partir de cette sauvegarde, et donc modifier le comportement de tous les players.

- « *cmd-alt-cg* » va libérer le préréglage, permettant de réaliser ultérieurement une nouvelle capture.
- « *cmd-cg* », force la capture des paramètres dynamiques dans cette case.
- « *cg* » sur un préréglage va, dans tous les cas, définir la configuration correspondante comme « configuration courante ». Le numéro (de 1 à 64) de cette configuration s'affiche dans la première cellule du *pad principal*.
- « *c* » (touche du clavier) permet de copier (dans le préréglage 65) le préréglage associé à la case (1 à 64) pointée par la souris.
- « *v* » (touche du clavier) permet de coller dans le préréglage associé à la case (1 à 64) pointée par la souris la valeur du préréglage 65.

5.4.2 Randomisation

Cette opération, dite parfois « *agitation* » consiste à modifier légèrement (ou non), au hasard, les valeurs des paramètres dynamiques pour obtenir de nouveaux sons inattendus (ou inentendus). Elle s'opère par l'une des touches « A », « Z », « E », « R », « T » ou « Y », qui ne diffèrent par leur effet que par l'ampleur croissante, de « A » à « Y », des variations obtenues. Comme on l'a vu au § précédent, cette modification s'applique directement aux players actifs, et ces nouveaux réglages peuvent être conservés dans n'importe quelle case du *pad des préréglages*.

5.4.3 Interpolation

Cette opération va permettre de naviguer d'un preset à un autre, de manière continue, par interpolation des valeurs de paramètres associés à ces deux presets. L'opération consiste à choisir un premier preset (par exemple, le 23) par un « cg », puis un second preset (par exemple, le 36) par un « cmd-cd ». Les réglages du second preset deviennent alors actifs, et le slider d'édition affiche alors « Interp. 23/36 1.000 ». L'indication « 23/36 » correspond donc aux presets entre lesquels on réalise l'interpolation, et la valeur du slider peut évoluer entre -1 et 1, -1 correspondant aux valeurs des paramètres du preset 23, 1 à celles du preset 36, et toute valeur comprise entre -1 et 1 correspond à une interpolation entre ces deux réglages.

Il est possible, en tapant « X », de passer en mode *extrapolation*, les bornes de variations passant alors de [-1 1] à [-3 3]. On revient au mode interpolation en tapant « I ». Les variations sont interactives, et à tout moment il est possible de capturer un réglage intéressant par un « cg » dans une case du pad des préréglages.

5.4.4 Visualisation et édition de préréglages

5.4.4.1 Utilisation

L'*afficheur de paramètres* permet de consulter les valeurs des paramètres sauvegardés dans le préréglage courant. Ce dernier est choisi :

- par un « clic gauche » dans une cellule du pad des préréglages (pour les préréglages 1 à 64)
- par une interaction avec la cellule « Config. » du *pad principal*. La touche « + » (ou « = ») incrémente ce numéro, la touche « - » (ou « _ » ou « : ») décrémente ce numéro. « z » passe ce numéro à 0, « d » (ou « p ») choisit « 68 » qui est le numéro correspondant aux préréglages par défaut, transmis aux lecteurs lors du lancement d'un clip. Les touches « q », « r » et « s » affichent respectivement les préréglages 66, 67 et 68.
- par le choix d'un player, dans le module « *Sound Units* ». Ce choix s'effectue par un « cg » dans l'une des cellules du module, ou encore par « s » qui sélectionne la case désignée à l'écran par le curseur. Le numéro du player, et le numéro du clip correspondant s'affichent alors dans la case 2 du *pad principal*, sous la forme [player▷clip], avec la couleur (bleu ou orange) du mode de jeu associé au clip. Notons que ce choix peut concerner un player qui vient de cesser de jouer. Ce choix est valide jusqu'à une nouvelle utilisation du même player. Il suffit alors d'un « cg » dans cette case 2 pour sélectionner la configuration associée au clip, dont le numéro s'affiche dans la case 1 sous la forme « -P », où P est le numéro du player.

La « configuration courante » est donc repérée par un numéro, qui désigne :

- 0 : les paramètres dynamiques, sur lesquels il est possible d’agir immédiatement.
- 1 à 64 : l’une des configurations du pad des préréglages.
- 65 : la dernière configuration choisie par un « copier », qui joue donc le rôle d’un clipboard.
- 66 : la précédente configuration avant un « coller »
- 67 : la configuration courante, avant toute modification, quand elle vient d’être choisie pour édition.
- 68 : la configuration « par défaut », transmise à tout clip nouvellement joué.
- -1, -2, etc : la configuration du clip joué par le lecteur 1, 2, etc.

Quand une configuration a été choisie, son numéro s’inscrit dans la case 1 du *pad principal*, et tous les paramètres de cette configuration sont affichables dans l’*afficheur de paramètres*, et éditables. Faire défiler les différents paramètres au moyen des flèches gauche et droite, cliquer sur la case désirée, et le slider du paramètre s’affiche. Notons que ce slider va permettre de modifier le paramètre correspondant de la configuration *courante*. Si le numéro de celle-ci est 0, le slider modifie directement le *paramètre dynamique*, sinon seule la valeur correspondante de la configuration courante est modifiée.

Les valeurs « FLM1 » et « FLM2 » qui sont des ensemble de drapeaux ne sont pas aisément éditables au moyen d’un slider, sauf si l’on se contente de choisir les positions extrêmes, « 0 » (aucun drapeau n’est sélectionné) ou « 0xFFFF » (tous les drapeaux sont sélectionnés). Le module propose donc un mode d’édition supplémentaire pour ces deux valeurs. Le fonctionnement est le suivant :

Lorsqu’une configuration est sélectionnée, ces deux valeurs s’affichent (en orange et vert respectivement, sous forme hexadécimale) dans les cellules 4 et 5 du *pad principal*. On sélectionne l’une de ces deux cellules par un « cg », et la valeur correspondante s’affiche sous forme d’une suite de bits dans les 28 cellules de l’*éditeur de drapeaux*. Chaque cellule de l’éditeur va refléter, par sa couleur, la valeur du bit correspondant. Il est possible, par un « cg » dans l’un des cercles, d’inverser le bit correspondant, et la valeur modifiée s’affiche, toujours en hexadécimal, dans cette cellule 4 ou 5. Signalons que les valeurs des cellules 6 et 7, repérées par des teintes de bleu différentes et les noms « And » et « Or », et dont l’utilité sera expliquée ultérieurement, vont pouvoir être éditées exactement de la même manière.

Notons encore que lorsque le pointeur de la souris passe au-dessus de l’un des cercles, le nom du paramètre affecté par le drapeau correspondant est affiché dans la cellule 8 du *pad principal*. Ceci permet de s’assurer que le drapeau que l’on va modifier est bien celui qui nous intéresse. La couleur de cette cellule est identique à celle de la cellule (4, 5, 6 ou 7) en cours d’édition si le drapeau est à 1, et est différente dans le cas contraire.

Il est possible de *désolidariser* la valeur affichée dans l’éditeur de drapeau de celles des cellules 4 à 7 par un « cd » dans l’une de ces cellules. La case 8 prend

alors une teinte foncée, indiquant que les actions dans l'éditeur de drapeau n'ont plus d'effet sur les valeurs de ces cellules.

Un « *cg* » dans la cellule 8 permet de passer du mode « affichage du nom d'un drapeau particulier » au mode « affichage de l'ensemble des drapeaux en hexadécimal ».

Un « *cd* » dans la cellule 8 montre le résultat de la transformation des valeurs des drapeaux après passage au filtre « And/Or ». Très précisément, lorsque le mode de copie des paramètres affiché dans la cellule 3 du *pad principal* est « Copy Mod », une valeur « FLM » (« FLM1 » ou « FLM2 ») est copiée par :

$$FLM_{dest} = (FLM_{source} \& And) | Or$$

Concrètement, la valeur de « And » va nous permettre de choisir les drapeaux de la valeur initiale que l'on veut conserver dans la destination, la valeur de « Or » va nous permettre d'en ajouter de nouveaux.

Illustrons par quelques cas concrets. On suppose que la case « Config » affiche « 0 », indiquant que l'on travaille directement avec les paramètres courants :

- toutes les valeurs des paramètres conviennent parfaitement, et l'on veut qu'aucun ne soit modifié dynamiquement. Il suffit de passer la valeur de « FLM1 » à « 0 », et d'utiliser le mode « copy all ». Ou encore, de ne pas modifier « FLM1 », de passer « And » et « Or » à « 0 », et d'utiliser le mode « copy mod ».
- l'ensemble des réglages nous convient, mais on veut pouvoir « explorer » dynamiquement l'influence de la *position* de la lecture des grains dans l'ensemble du clip. On a probablement déjà réglé « Dyn.Pos.var » à une valeur faible (entre 0 et 15%), et l'on veut que la position dynamique « Dyn. Position » puisse varier dynamiquement. Le plus simple est d'afficher « Or » (« *cg* » sur la case 7 du *pad principal*), de remettre à 0 sa valeur si nécessaire (« *z* » dans la case), de déplacer le pointeur dans l'éditeur de drapeaux, jusqu'à sélection du drapeau « Direction », et par un « *cg* » de passer sa valeur à « 1 ». La case 7 affiche alors « Or : 000100 ». Passer « And » à « 0 », et utiliser le mode « copy mod ». On peut aussi choisir « FLM1 », passer sa valeur à 0, déplacer le pointeur dans l'éditeur de drapeaux, jusqu'à sélection du drapeau « Direction », et par un « *cg* », passer sa valeur à « 1 ». Après avoir effectué la même manipulation pour « FLM2 », utiliser le mode « copy all ».

Après ces manipulations la commande « *c* » dans la case « configuration » copie la configuration courante (0, donc) dans le pré-réglage 65. Il est possible de « coller » cette configuration dans le pad des pré-réglages (faire « *v* » au-dessus de l'entrée désirée), ou dans un player. Choisir (« *cg* ») l'un des players, même inactif. Sa dernière configuration utilisée s'affiche dans la case 2 (c.f. 5.2 page 106), et c'est cette configuration qui va être utilisée. Faire alors « *v* » au-dessus de cette case.

Il est aussi possible d'éditer directement la configuration d'un clip joué ou en cours de jeu dans le player lui-même. Après avoir choisi l'entrée désirée dans le module « *Sound Units* », faire un « *cg* » sur la seconde case du *pad principal*. Le numéro du player apparaît (sous la forme « -1 » pour le player 1), et il est possible d'éditer tous les paramètres associés à ce jeu spécifique. Le player est verrouillé durant l'édition. Il est libéré automatiquement dès qu'une autre configuration est choisie.

Notons enfin qu'il est possible de relancer la lecture du clip sélectionné par un « *cd* » sur la case 2, tant que le player n'a pas été réutilisé.

Enfin, « *i* » dans cette case 2 « installe » les modifications dans l'historique du play log : le clip sera rejoué avec ces valeurs spécifiques des paramètres, mais ne sera plus influencé par les paramètres dynamiques (il sera toujours possible de l'éditer à nouveau ultérieurement).

5.4.4.2 Récapitulatif des commandes

Ce paragraphe décrit sommairement les commandes s'appliquant aux cinq zones d'édition du module « *Dynamic Play Parameters* »

Pad Principal

Cellule 1 « *cg* » passe à l'édition du préréglage suivant. « *cd* » passe à l'édition du préréglage précédent. « *+* » (ou « *=* ») passe à l'édition du préréglage suivant. « *-* » (ou « *_* » ou « *:* ») passe à l'édition du préréglage précédent. « *z* » choisit le préréglage 0. « *d* » choisit le préréglage 68. « *p* », « *q* », « *r* » ou « *s* » choisissent les préréglages 65, 66, 67 ou 68. « *c* » copie le préréglage courant en 65. « *v* » colle le préréglage 65 dans le préréglage courant.

Cellule 2 « *cg* » sélectionne la configuration associée au [player▷clip] de cette cellule 2 comme « configuration courante ». « *cd* » relance la lecture du clip, en tenant compte des modifications effectuées. « *c* » copie la configuration [player▷clip] en 65. « *v* » colle la configuration 65 dans le clip. « *s* » sauvegarde la configuration associée au [player▷clip] dans l'historique. « *i* » effectue la même opération, mais en passant « *FLM1* » et « *FLM2* » à 0, ce qui assure que le jeu du clip sera basé exclusivement sur la configuration locale enregistrée, et non les paramètres dynamiques. Enfin « *a* » abandonne l'édition : les modifications exécutées ne seront pas inscrites dans l'historique.

Cellule 3 Détermine le mode de copie : « *Copy All* » indique que paramètres et drapeaux sont copiés dans leur intégralité. « *Copy Par* » indique que seuls les paramètres sont copiés. « *Copy Mod* » fonctionne comme « *Copy All* », mais les drapeaux passent au travers du filtre décrit ci-dessus (§ 5.4.4.1 page 110). « *cg* », « *cd* », « *+* », « *a* », « *p* », « *m* » permettent de naviguer entre ces modes.

Cellules 4, 5, 6 et 7 « *cg* » sélectionne la cellule pour édition dans l'éditeur de drapeaux. « *cd* » désolidarise les cellules de cet éditeur. « *z* » remet la cellule à 0. « *f* » remet à cellule à « FFFFF ».

Cellule 7 « *cg* » ou « *t* » alternent entre vision par drapeau et vision de l'ensemble des drapeaux. « *cd* » ou « *x* » alternent entre vision directe des « FLM » et vision modifiée par les options de recopie (§ 5.4.4.1 page 110).

Editeur de drapeaux Lorsque l'on a sélectionné l'une des cellules 4 ou 5 du *pad principal*, cette zone affiche dans les 28 cercles les drapeaux correspondants des valeurs « FLM1 » ou « FLM2 ». La cellule 8 du *pad principal* affiche le nom du flag sur lequel le pointeur de la souris est positionné. Un « *cg* » permet alors d'inverser ce flag (pour rappel, lorsque le flag est à 1, il est de la couleur du champ « FLM1 » ou « FLM2 » choisi ; ce flag à 1 implique que la valeur correspondante du paramètre vient du paramètre dynamique correspondant. Dans le cas contraire, cette valeur vient du champ fixe conservé dans le préréglage). Un « *cd* » sélectionne directement ce paramètre, dont la valeur est éditable au moyen du « *slider d'édition* ».

L'afficheur de paramètres Cette zone affiche 4 paramètres dynamiques, avec leurs noms et leurs valeurs courantes. Les touches « ◀ » et « ▶ » permettent de les faire défiler vers la gauche ou vers la droite. Cliquer (« *cg* ») dans l'une des quatre cases permet d'éditer la valeur correspondante, grâce au « *slider d'édition* »

Le slider d'édition Lorsqu'il a été lié à l'un des paramètres dynamiques (c.f. les deux § ci-dessus), ce slider permet d'éditer le paramètre correspondant.

Le pad des préréglages Ce pad affiche 64 préréglages. Une sélection (par la *cellule 1* du *pad principal*) ou un « *cg* » sur une case copie dans le préréglage correspondant (si la case est initialement vierge) les valeurs des paramètres dynamiques. Si la case est déjà utilisée, le « *cg* » a pour seul effet de sélectionner ce préréglage comme « préréglage courant ». « *cmd-cg* » force l'écriture des paramètres dynamiques, même si la case est déjà utilisée. « *cmd-alt-cg* » efface le préréglage correspondant. Un « *cd* » affecte aux paramètres dynamiques les valeurs conservées dans le préréglage. « *c* » copie les valeurs de ce préréglage dans le préréglage 65. « *v* » colle dans ce préréglage les valeurs du préréglage 65.

Deuxième partie

Le langage de script

Chapitre 6

Programmer avec mSL

Dans cette seconde partie, nous aborderons *mSL*, (pour « *micro Script Language* »), le langage de script intégré au Game Master. Nous nous intéresserons d'abord dans ce chapitre aux concepts les plus simples de la programmation, tels qu'ils sont incarnés en *mSL*. Un autre chapitre décrira plus formellement le langage. Puis nous nous intéresserons aux interactions possibles entre *mSL* et le *Game Master*, pour terminer par les aspects les plus évolués : multiprogrammation, processus, messages, etc. Dans toutes ces parties, des exemples, directement exécutables, seront disponibles pour mettre en application les concepts exposés.

Ce premier chapitre a donc pour ambition de présenter assez brièvement une introduction à la programmation, destinée au programmeur débutant qui désire écrire ses premiers scripts en *mSL*. Il ne propose rien de compliqué ni de révolutionnaire, mais gageons que même le programmeur chevronné y apprendra quelque chose à l'occasion... Celui-ci peut aussi passer immédiatement au chapitre suivant, où *mSL* est présenté plus en détail et plus formellement.

Sa lecture est cependant conseillée à tous. Assurez-vous de comprendre les codes sources proposés et les explications correspondantes. Ne sautez pas d'étapes ! Chaque exemple, chaque paragraphe vise à vous apporter quelques concepts supplémentaires...

6.1 Introduction

Une vision ancienne, mais toujours d'actualité, est de considérer qu'un ordinateur est composé d'une « *Mémoire Centrale* » (« MC »), comportant un nombre élevé, mais fini, de *cellules* pouvant contenir des valeurs numériques, et d'une « *Unité Centrale* » (« UC ») pouvant exécuter des « *programmes* », réalisant des « *algorithmes* », dont le rôle est de modifier les valeurs des cellules de la mémoire, jusqu'à obtention du résultat attendu.

Cette vision est présentée ici car elle est explicitement intégrée à *mSL* (et à *eel2*, aka « *JS* », son langage d'implémentation). Chaque cellule de la « *MC* » est désignée par son adresse, un nombre entier allant de 0 jusqu'au numéro de la dernière cellule, par exemple 16777215 (2 puissance 24 moins 1), ou encore 4294967295 (2 puissance 32 moins 1), etc. Le langage « *mSL* », et sa base d'implantation « *JS* », assimilent la mémoire à un tableau de nombres décimaux, chaque cellule de la mémoire pouvant contenir n'importe quel nombre décimal compris entre « -1.7976931348623158E+308 » et « 1.7976931348623158E+308¹ ».

En utilisant *mSL*, il est possible de lire ou d'écrire n'importe quel mot de cette mémoire, au moyen d'une séquence d'instructions similaires aux suivantes² :

```
var M = 0;
M[7231];
M[537328] = 7;
```

La première ligne déclare une variable (déclaration « *var* ») de nom « *M* », et lui donne, par l'opérateur « = », la valeur numérique « 0 ». La seconde ligne permet de lire la cellule de mémoire d'adresse 7231. Dans la troisième ligne, la cellule de mémoire située à l'adresse 537328 est modifiée pour contenir la valeur 7. Toutes les instructions *mSL* se terminent par le caractère « ; », qui marque la fin de l'instruction. Notons que les espaces (blanc, tabulation, passage à la ligne) sont facultatifs, sauf lorsqu'ils sont utilisés pour séparer deux séquences de lettres (ici, « *var* » et « *M* »).

Note : accès à la mémoire Ce n'est bien sûr pas un hasard que la variable « *M* » soit initialisée à 0. La *sémantique* de cette construction « *A[B]* », dite *indexation*, est de consulter la cellule de mémoire d'adresse *A+B*. Il est équivalent d'écrire *B[A]*, voire *0[A+B]* ou *(A+B)[0]*. *M[7231]* est donc équivalent à *0[7231]*, ou encore à *7001[230]* ! De même, « *A[B]=C* » copie dans la cellule de mémoire de numéro *A+B* la valeur *C*.

En général, l'on n'a pas besoin de lire ou d'écrire spécifiquement une adresse particulière de la mémoire, mais pour programmer un algorithme, il est très souvent nécessaire d'utiliser plusieurs quantités différentes, que l'on puisse aisément distinguer les unes des autres. Pour ce faire, nous allons demander à *mSL* d'associer pour nous une cellule de la mémoire de son choix, à un nom que nous avons choisi. Cette association (*nom* et *cellule de mémoire*) est dite « *variable* ». Cette demande se fait sous la forme « *var toto;* » dans laquelle *var* est un *mot clef* du langage (on ne

1. Cette notation se lit « 1 virgule 7976931348623158 multiplié par 10 élevé à la puissance 308 », ce qui est un très grand nombre, 1 suivi de 308 zéros – pour comparaison, on évalue actuellement le nombre d'atomes de l'univers à « seulement » 2 multiplié par 10 élevé à la puissance 79, soit, en décimal, 2 suivi de 79 zéros.

2. Nous utiliserons cette typographie particulière chaque fois que nous ferons référence à des éléments du langage *mSL*, identificateurs, valeurs numériques, fragments de code, etc.

peut l'utiliser *que* pour déclarer une variable), et `toto` est le nom que nous avons choisi pour cette variable. L'affectation va nous permettre de changer la valeur d'une variable.

Illustrons ces concepts par quelques exemples simples.

Exemple 1 On veut calculer la somme des 10 premiers nombres entiers. On peut bien entendu écrire (*algo 1.1*) :

```
var s = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
```

qui répond parfaitement à la question, ou encore (*algo 1.2*) :

```
var s = (10 * (10 + 1)) / 2;
```

qui fait appel à la formule bien connue, $\frac{N \times (N+1)}{2}$, instanciée ici pour $N = 10$.

Dans ces deux écritures, qui toutes deux vont associer la valeur 55 à la variable `s`, on trouve une *déclaration* de variable, « `var s;` » et une *affectation*, « `s = expression;` » qui consiste à modifier la valeur de la cellule de mémoire désignée par le nom placé à gauche du symbole d'affectation, « `=` », pour la remplacer par le résultat de l'expression située à droite de ce signe. Ici, la déclaration et l'affectation sont combinées en une expression unique.

Dans les *expressions*, l'on utilise des symboles traditionnels de l'informatique, « `+` », « `*` », « `/` » pour représenter l'*addition*, la *multiplication* et la *division*. On parle en général d'*opérateurs* pour désigner ces symboles qui font référence aux opérations mathématiques classiques. Les parenthèses permettent d'indiquer qu'une sous-expression doit être calculée en priorité, ici « `(10 + 1)` ». Ajoutons la *soustraction*, « `-` », et nous avons fait le tour des principales opérations arithmétiques. Enfin, vous l'avez remarqué, les instructions se terminent par un *point-virgule*, « `;` ».

Note : déclarations Dans nombre de langages de programmation, le concept de *déclaration* est associé à celui de « *type* ». Ainsi, le langage « C » distingue les *entiers*, déclarés par « `int` », nombres sans partie décimale, des *réels*, déclarés par « `float` », qui peuvent avoir une partie décimale. L'une des raisons est que cette distinction correspond au fait que les machines sur lesquelles existent des compilateur « C » proposent différents types de représentations internes des données, destinés à optimiser l'écriture et l'efficacité des programmes. Le langage « *eel2* » (ou « JSFX ») ne propose pour sa part qu'un type unique de représentation des données, le « *flottant double* ». Cette restriction se retrouve de fait dans le langage *mSL*. Dès lors, pourquoi imposer une déclaration des variables ? « *eel2* » considère que tout identificateur rencontré dans un programme est implicitement déclaré comme variable. Ceci simplifie les envolées lyriques dans lesquelles le programmeur laisse les instructions couler à flots, et ne désire pas, en outre, s'ennuyer à déclarer les variables qu'il invente au fil de la plume... Mais est aussi extrêmement propice à la création de bugs souvent difficiles

à détecter, dans lesquelles une faute de frappe fait utiliser une autre variable que celle que l'on avait prévue ! C'est pour cette raison que j'ai choisi de rendre obligatoire la déclaration de toutes les variables utilisées, ce qui, rétroactivement, me semble aujourd'hui avoir été une sage décision.

6.2 Premiers éléments

Un programme va consister en une séquence d'instructions. Nous les écrirons à raison d'une par ligne, et elles vont s'exécuter dans l'ordre, l'une après l'autre, de haut en bas. Ainsi, dans l'exemple suivant, tentez de calculer après chaque instruction, les valeurs des deux variables « a » et « b ». Que constatez-vous ?

```
var a, b;  
a = 17;  
b = 11;  
a = a + b;  
b = a - b;  
a = a - b;
```

Après la déclaration des variables et leur initialisation, les trois dernières lignes constituent un « petit algorithme » qui, vous l'avez deviné, échange les valeurs de « a » et « b ».

Revenons à notre somme des dix premiers entiers, et proposons encore une autre approche, moins appropriée dans ce cas précis puisque des solutions plus simples existent, mais intéressante par sa généralité (*algo 1.3*).

```
var s;  
var i;  
s = 0;  
i = 0;  
while (i < 10) (  
    i = i + 1;  
    s = s + i;  
);
```

L'écriture semble compliquée, mais nous abordons ici le coeur de l'algorithmique : construire l'équivalent d'une « recette de cuisine », compréhensible par la machine, qui puisse aboutir au résultat désiré.

Les quatre premières lignes sont des opérations qui sont maintenant familières : déclarations et *initialisations* de variables. La cinquième ligne propose une construction nouvelle, qui est une *répétition d'instructions*, souvent dite *boucle*. Sa syntaxe est la suivante :

```
while ( condition ) ( instructions );
```

Le mot-clef « *while* » détermine la nature de l'instruction, les parenthèses et le point-virgule final sont obligatoires. Ces éléments, dans cet ordre précis, constituent la *syntaxe* de l'instruction. La *sémantique* de cette instruction est la suivante : tant que le calcul de l'expression « *condition* » donne le résultat *vrai*, l'instruction *while* va exécuter les « *instructions* » placées dans le second couple de parenthèses. Rappelons ici que le format des instructions est libre : un espace est nécessaire pour séparer deux identificateurs (par exemple le *mot-clef* *var* et l'*identificateur* *s*), mais les espaces sont facultatifs lorsqu'il n'y a pas d'ambiguïté. On peut en placer autant que l'on le souhaite, insérer des sauts de ligne, etc. ; ainsi l'instruction *while* est ici écrite sur quatre lignes.

La condition de l'instruction *while*, dans notre cas, est l'expression « *i* < 10 » qui se lit « *i* inférieur à 10 », dont la valeur est *vrai* si la variable « *i* » est plus petite que 10, et *faux* sinon. Notons que lorsqu'une variable est utilisée dans une expression, elle est remplacée par le contenu de la cellule qu'elle désigne.

Soyons conscient également de l'aspect *temporel* lié à un programme. La machine évalue *dans l'ordre* du texte source les instructions successives du programme. Après la déclaration des variables *s* et *i*, *s* reçoit la valeur 0, puis *i* reçoit la valeur 0, puis l'instruction *while* s'exécute.

Lors de l'exécution du *while*, « *i* < 10 » est calculée une première fois. Il s'agit d'une comparaison, que l'on peut lire « *i* est-il strictement inférieur à 10 ? ». Pour ce faire, la variable *i* est remplacée par sa valeur, 0, et comme 0 est plus petit que 10, le résultat de la comparaison est *vrai*. Ce résultat implique que le calcul des instructions situées entre les parenthèses du *while* va être effectué. La première expression, « *i* + 1 » calcule donc « 0 + 1 », et le résultat est mis « dans » *i*, qui vaut maintenant 1. La seconde expression est « *s* + *i* », les variables ont pour valeurs 0 pour *s*, 1 pour *i*, et le résultat est 1, qui est placé dans *s*.

Le premier calcul de la condition ayant fourni la valeur *vrai*, la boucle *while* va donc être exécutée à nouveau. Le nouveau calcul de « *i* < 10 » se résume à « 1 < 10 » qui est *vrai*, ce qui va entraîner une nouvelle exécution des instructions internes du *while*. Celles-ci vont affecter « 1 + 1 », soit 2, à *i*, puis « 1 + 2 », soit 3, à *s*. La condition du *while* étant toujours *vraie*, les calculs vont maintenant changer *i* en 3, et *s* en 6. A l'itération suivante, *i* va devenir 4 et *s* 10. Puis les variables vont prendre les valeurs 5 et 15, 6 et 21, 7 et 28, 8 et 36, 9 et 45, puis 10 et 55. A ce moment là, le test de la condition « *i* < 10 » calcule « 10 < 10 », qui donne le résultat *faux*, ce qui interrompt l'exécution de la boucle. Le résultat attendu, 55, est dès lors disponible dans la variable *s*.

L'intérêt de cette approche est qu'elle nous fournit un *mécanisme*, dans lequel une variable, *i*, prend toutes les valeurs entre 1 et 10 (ou entre 1 et un nombre quelconque), à partir duquel on peut résoudre différents problèmes similaires.

Par exemple, si nous transformons la seconde instruction de la boucle en (*algo 1.4*) :

```
s = s + (i * i);
```

nous calculons non pas la somme des nombres de 1 à 10, mais la somme des *carrés* de ces nombres.

De même, « `s = s + sqrt(i)` » va nous donner la somme des *racines carrées* des nombres de 1 à 10. Il est fait cette fois appel à la fonction prédéfinie `sqrt` (*square root*). Signalons à cette occasion que la *syntaxe* d'un appel de fonction est :

```
nom_de_la_fonction ( paramètres )
```

Les parenthèses suivant le nom de la fonction sont partie intégrante de la syntaxe d'appel de la fonction. C'est entre celles-ci que l'on place les valeurs à partir desquelles la fonction va effectuer son travail. Une fonction utilise habituellement une ou deux valeurs, mais certaines peuvent en attendre un grand nombre, séparées les unes des autres par des virgules, ou encore... aucune ! Une fonction est un « *petit algorithme* » qui s'applique à des données – le terme précis est « *paramètre* ». Une fonction va utiliser un seul paramètre (comme `sqrt`), ou, comme nous le verrons bientôt, deux (`min` ou `max`), plus (`sprintf`), voir aucun (`stralloc`). Ces paramètres sont placés entre parenthèses, séparés par des virgules, à la suite du nom de la fonction. Cette *construction syntaxique* constitue un *appel de fonction*. Tout comme `A[B]` indique une indexation, `F(X)`, `G()` ou `H(X, Y, Z)` dénotent des appels de fonctions, avec 1, 0 et 3 paramètres respectivement.

Enfin, écrire « `s = s * i` » (à condition d'avoir initialisé `s` à 1 et non à 0) va nous fournir le produit des dix premiers entiers, c'est-à-dire la *factorielle* de 10.

Exemple 2 Continuons avec notre travail sur les boucles. Nous allons cette fois faire appel à un *tableau*. Alors que les variables que nous avons utilisées jusqu'à présent (telles `s` et `i`) sont associées à un unique élément, qui est une cellule de la mémoire, les tableaux font référence à une séquence d'éléments, c'est-à-dire plusieurs cellules de mémoire consécutives, désignées par leur numéro dans la séquence, de 0 à $N - 1$.

Le problème que l'on se pose est de trouver le plus grand élément, dans un tableau de dix valeurs. Nous allons donner le nom de « *liste* » à notre tableau, et le définir par :

```
var liste = data (7 11 43 0 2 1 1 637 -7 13);
```

La variable « *liste* » est bien sûr déclarée par « `var` », et reçoit comme valeur un tableau de dix éléments, construit par l'opération « `data` ». Le résultat de cette opération est l'adresse du premier élément de ce tableau. La valeur 7 est donc accessible par l'expression « `liste[0]` », la valeur 11 par « `liste[1]` », etc. (*algo 2.1*) :


```

var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s;
var i;
s = 0;
i = 0;
while (i < 10) (
    s = max(s, liste[i]);
    i = i + 1;
);

```

L'écriture, là encore, est similaire à la boucle précédente. Nous faisons appel à l'indexation, symbolisée par les crochets, pour consulter les valeurs des éléments successifs de « *liste* », ainsi qu'à la fonction « *max* ». Celle-ci attend deux valeurs (ou, plus généralement, deux expressions), et nous renvoie la plus grande de ces deux valeurs. A chaque cycle, la plus grande valeur trouvée jusqu'à présent, *s*, est comparée avec l'élément courant, *liste[i]*, et la plus grande de ces deux valeurs est conservée. En utilisant la fonction « *min* » (minimum de deux valeurs) au lieu de « *max* », on obtient encore un nouvel algorithme, recherchant, lui, le plus petit élément du vecteur.

Précisons que « *data* », « *liste* », « *i* », « *s* », « *max* », « *while* », etc., au-delà de tout rôle sémantique particulier, sont des *identificateurs*. Syntaxiquement, un identificateur se construit comme une suite de lettres et de chiffres, débutant par une lettre, et pouvant comprendre le caractère « *_* » assimilé à une lettre.

6.3 Premiers essais

Il est temps de passer à la pratique... Et tout d'abord, se pose la question : comment rédiger un programme *mSL* ?

Le Game Master ne contient pas d'éditeur de texte. Le langage d'implantation, « *JS* », se prête mal à ce genre d'exercice, et il existe par ailleurs nombre d'outils, quels que soient la machine et le système utilisés, qui permettent cette activité. L'approche proposée est donc de créer, de manière indépendante de REAPER et du Game Master, un fichier texte contenant le programme *mSL* à tester, puis de charger ce programme dans le Game Master. Pour ma part, j'opère sur Mac, et j'utilise *VS Code* (*Visual Studio Code*), mais à l'occasion *TextEdit* ou *Vim* conviennent parfaitement.

Essayons donc notre premier programme. Lancer *TextEdit* (ou l'équivalent sur votre machine), et dans la fenêtre qui s'affiche (intitulée « Sans Titre »), taper :

```

var s = (10 * (10 + 1)) / 2;

```

sauvegarder sous le nom « *test1.mSL* ». Le premier programme est écrit – car oui, cette expression est aussi un programme complet, prêt à être exécuté. Il nous faut

maintenant le soumettre au GM. Que va-t-il se passer ?

Lorsque le GM reçoit un « programme » (c'est le cas si on lui propose un fichier dont le nom se termine par l'extension « .mSL »), il considère que ce fichier doit être exécuté. Dans un premier temps, le texte du fichier est soumis à un module spécialisé, le compilateur *mSL*, qui va analyser ce texte, et, si la syntaxe en est correcte, va générer une structure de donnée, le « code objet » représentant ce programme prêt à être exécuté (c.f. § 6.7 page 140). Dans le cas contraire, il imprime un message d'erreur, indiquant un type d'erreur, un numéro de ligne, et une position dans la ligne. Si tout s'est bien passé, le GM va construire un processus destiné à l'exécution du programme, processus qui est confié à un interprète qui va se charger d'exécuter le code ainsi mis en forme. Là aussi, une erreur peut être détectée, qui provoquera l'arrêt de l'exécution.

Soumettons donc notre programme au GM. Pour ceci, afficher dans le Finder le répertoire dans lequel vous avez effectué la sauvegarde, faire un clic-gauche sur le nom du fichier, et, en gardant cette touche enfoncée, glisser le fichier sur la barre horizontale des tabulations, en haut de la fenêtre du Game Master (ai-je précisé qu'il fallait d'abord lancer l'exécution de celui-ci ?). Relâcher la touche de la souris : le programme vient de s'exécuter, dans la plus parfaite discrétion.

Le but d'un script étant d'être le plus discret possible et de se faire oublier, on ne peut bien sûr rien reprocher au système. Pourtant, dans cette phase d'apprentissage, il serait bien de disposer un petit retour du système. Pour ceci, nous allons utiliser la fonction « GM_Log » qui va permettre de visualiser un message dans le journal du Game Master, affiché par le module « System Log ». Ajoutons une seconde ligne :

```
GM_Log("ça marche !");
```

Nous faisons ici appel à la fonction « GM_Log » – nous avons déjà rencontré d'autres fonctions comme *sqrt*, *max*, etc. – mais ici la donnée transmise à la fonction (on parle de *paramètre*) est un nouvel objet : une *chaîne de caractères*, qui est une suite de lettres, chiffres, espaces et tous ces autres signes bizarres du clavier, suite placée entre doubles quotes, « " ».

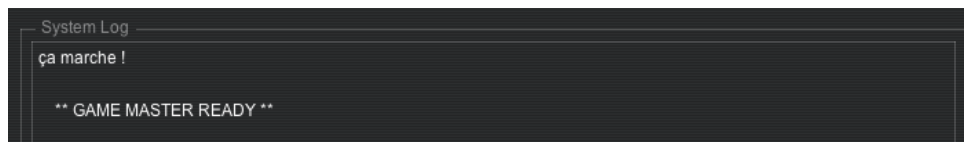


FIGURE 6.1 – Le message du premier programme

Exécutons le programme ainsi modifié : notre message s'affiche dans le module « System Log » (c.f. fig. 6.1). Nous voici donc rassurés sur le fait que le calcul se soit exécuté.

Bien sûr, il n'est pas très satisfaisant de ne pas voir le résultat du programme. Transformons le en ceci :

```
GM_Log("Calcul de la somme des dix premiers entiers.");  
var s = (10 * (10 + 1)) / 2;  
var str = stralloc();  
sprintf(str, "s vaut %d", s);  
GM_Log(str);
```

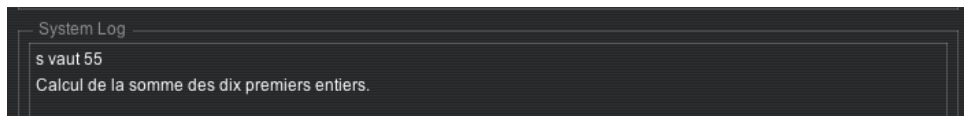


FIGURE 6.2 – Les messages du second programme

La figure 6.2 nous montre le résultat de cette nouvelle exécution. Cette fois-ci, un premier message indique la raison d'être du programme. Puis vient le calcul proprement dit, et enfin trois lignes qui ne servent qu'à afficher le résultat.

Analysons-les en détail. «`var str = stralloc();`» déclare une nouvelle variable, de nom «`str`», et lui affecte le résultat de l'appel de la fonction «`stralloc`». Celle-ci ne prend pas de données, mais fournit à chaque appel une nouvelle *chaîne de caractères*, qui va pouvoir être modifiée par le programme. La ligne suivante nous montre un appel d'une fonction très spéciale, `sprintf`, dont le rôle est de représenter, sous la forme d'une séquence de caractères, des valeurs numériques quelconques. Le premier paramètre est la chaîne que nous venons d'obtenir, à l'intérieur de laquelle va s'exécuter l'édition. Le second paramètre est le *format* d'édition. Dans cet exemple, la séquence de deux caractères «`%d`» indique qu'un nombre va être affiché sous la forme d'un *entier*. Cette description de format de représentation peut être associée à d'autres caractères, qui vont être reproduits tels quels. Cette «*spécification de sortie*» est représentée par une chaîne de caractères, qui est le second élément de l'appel de la fonction. Le premier élément est une chaîne de caractères, à l'intérieur de laquelle va être éditée la valeur numérique, et le troisième élément de l'appel est la valeur à éditer. Enfin, la ligne suivante imprime la valeur de la variable `str`, qui désigne la chaîne de caractères qui sert de support à l'édition. On notera que le «*System Log*» édite le dernier message reçu en haut de sa fenêtre, «*poussant*» les autres vers le bas. Les messages venus d'un programme se lisent donc de haut en bas en ordre inverse de leur arrivée.

Transformons de même notre programme de recherche du plus grand entier d'un tableau :

```
GM_Log("Recherche du plus grand élément d'un tableau.");
```

```

var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s; var i;
s = 0; i = 0;
while (i < 10) (
    s = max(s, liste[i]);
    i = i + 1;
);
var str = stralloc();
sprintf(str, "s vaut %d", s);
GM_Log(str);

```

Nous obtenons dans le *log* la sortie suivante :

```

s vaut 637
Recherche du plus grand élément d'un tableau.

```

Ce qui nous conforte dans l'idée que notre programme est correct. Mais est-ce bien certain ? Le fait d'avoir un programme qui s'exécute sans erreur et fournit la valeur attendue n'implique pas forcément qu'il soit juste... Reprenons notre exemple, calculons cette fois avec d'autres données le *plus petit* élément, en utilisant la fonction `min`. Le texte devient :

```

GM_Log("Recherche du plus petit élément d'un tableau.");
var liste = data (5 7 328 2 110 220 380 1138 334 17);
var s; var i;
s = 0; i = 0;
while (i < 10) (
    s = min(s, liste[i]);
    i = i + 1;
);
var str = stralloc();
sprintf(str, "s vaut %d", s);
GM_Log(str);

```

L'exécution ne laisse pas de nous surprendre :

```

s vaut 0
Recherche du plus petit élément d'un tableau.

```

La raison, nous la découvrons assez vite : nous avons choisi « 0 » comme valeur initiale de « s », un nombre plus petit que tous ceux du tableau ! Au fil des itérations de notre boucle *while*, du fait que les éléments du tableau sont tous strictement positifs, et donc plus grands que notre point de départ, 0, la variable `s` va garder cette dernière valeur. Il est clair que « 0 » n'était pas le meilleur choix comme valeur initiale.

Revenons sur nos premiers algorithmes. Dans chacun, une variable, qui va être le résultat, part d'une valeur initiale, et évolue au fur et à mesure que l'opération progresse jusqu'à contenir le résultat. La formulation même de la question, ou de la finalité de l'algorithme, nous fournit une indication précise : calcul de la « somme » des dix premiers entiers, du « produit » de ces entiers, du « plus petit » ou du « plus grand » des éléments d'un tableau. La fonction essentielle impliquée dans ces algorithmes est l'addition, la multiplication, le maximum ou encore le minimum. Dans ces cas là, il est souvent pertinent de choisir comme « valeur résultat » de départ *l'élément neutre* de la fonction concernée : « 0 » et « 1 » respectivement pour l'addition et la multiplication. Plus généralement, l'élément neutre e d'une fonction f définie sur un ensemble E est tel que $\forall x \in E, f(x, e) = f(e, x) = x$. Pour le maximum, cet élément est le plus petit nombre représentable sur la machine, soit $-\infty$ si ce concept est accessible dans le langage, ou encore, dans notre cas, $-1.7976931348623158E + 308$, puisque *mSL* travaille avec des flottants 64 bits à la norme IEEE 754. Pour le minimum, l'élément neutre est la valeur opposée, $1.7976931348623158E + 308$. Pour faire fonctionner correctement notre programme, il suffit donc de remplacer « $s = 0;$ » par « $s = 1.7976931348623158E+308;$ ».

Signalons une autre approche : lorsque le concept est de calculer une somme, un produit, un maximum ou un minimum d'un ensemble, on peut utiliser n'importe quel élément de l'ensemble comme point de départ. Le plus simple est souvent de prendre le premier élément, et de débiter l'algorithme à partir de la seconde position. Voici un exemple ainsi modifié :

```
GM_Log("Recherche du plus grand élément d'un tableau.");
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s; var i;
s = liste[0]; i = 1;
while (i < 10) (
    s = max(s, liste[i]);
    i = i + 1;
);
var str = stralloc();
sprintf(str, "s vaut %d", s);
GM_Log(str);
```

Notons que « s » prend maintenant la valeur du premier élément, de numéro « 0 », et que la boucle débute à l'indice « $i = 1;$ ».

L'algorithme suivant va nous permettre de trouver la position du plus petit élément d'un tableau :

```
GM_Log("Indice du plus petit élément d'un tableau.");
```

```

var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s, i;
s = 0; i = 1;
while (i < 10) (
    ((liste[i] < liste[s]) ? (s = i););
    i = i + 1;
);
var str = stralloc();
sprintf(str, "s vaut %d", s);
GM_Log(str);

```

Ligne 3, nous utilisons l'écriture « `var s, i;` » pour déclarer les deux variables en une seule instruction. Nous faisons également appel, ligne 6, à une construction nouvelle, *l'instruction conditionnelle*. Celle-ci obéit à l'une des deux syntaxes suivantes :

```

( ( condition ) ? ( si_vrai_alors_ceci ) ; );
( ( condition ) ? ( si_vrai_alors_ceci ) : ( sinon_celà ) ; );

```

Avouons immédiatement que toutes les parenthèses ne sont pas indispensables. J'utilise ici les parenthèses externes pour m'assurer que le tout constitue une instruction, et les parenthèses internes pour m'assurer de délimiter correctement la « *condition* », la partie « *alors_ceci* » et la partie « *sinon_celà* », cette dernière pouvant être facultative (première syntaxe).

Les éléments importants sont le « `?` » qui termine la condition, et le « `:` » qui introduit le « *sinon* ».

Dans notre algorithme, « `s` » représente l'indice (le numéro) du plus petit élément trouvé jusqu'alors. Lorsque le test « `liste[i] < liste[s]` » nous dit que le nouvel élément courant de la liste, d'indice « `i` », est plus petit que celui déjà trouvé, on change la valeur de « `s` » pour désigner ce nouvel élément.

6.4 Opérateurs, fonctions, expressions et instructions

Nous avons jusqu'à présent manipulé des variables, qui peuvent référencer un élément unique de la mémoire (on parle de variables *simples* ou de *scalaires*), ou une séquence d'éléments, les *tableaux*.

Les calculs s'effectuent sur ces données au moyen d'*opérateurs* (« `+` », « `-` », etc.) ou de *fonctions* (« `max` », « `min` », « `sqrt` », etc.) qui sont des objets similaires, aux détails de syntaxe près. Cette syntaxe, justement, a été choisie pour rendre plus familière l'écriture des programmes. Les signes les plus proches de la représentation mathématique usuelle ont été choisis pour représenter les opérations correspondantes.

Les caractères disponibles sur le clavier étant en nombre limité, des *noms* (ou *identificateurs*) sont utilisés pour représenter les autres fonctions.

De même, la notation mathématique traditionnelle utilise des conventions spécifiques pour décider quelle opération est exécutée avant telle autre. Ainsi, dans la fameuse écriture $ax^2 + bc + c$, on se souvient que x^2 est calculé en priorité, que ax^2 et bx contiennent des multiplications implicites (pour $a \times x^2$ et $b \times x$), qui s'exécutent sous la forme des produits de a par x^2 et de b par x , et que viennent enfin les additions.

On retrouve de même dans le langage *mSL* des priorités entre les opérateurs. Dans une expression qui contient des additions et des multiplications, les multiplications s'effectuent en priorité. Notre trinôme du second degré s'écrit ainsi en *mSL* « $a * x * x + b * x + c$ », où « $x * x$ » traduit x^2 , en utilisant les fonctions que nous connaissons déjà, mais on peut aussi écrire, d'une manière encore plus proche de l'originale, « $a * x^2 + b * x + c$ ». Ici, l'opérateur « $^$ » représente l'élévation à la puissance, et est plus prioritaire que la multiplication. « x^2 » est donc calculé en priorité, puis viennent les produits, puis les additions. Si plusieurs opérateurs de même priorité sont utilisés dans une expression (« $1+2+3+4+5+6+7+8+9+10$ »), les calculs s'effectuent en général de gauche à droite ; ici, on ajoute au résultat de « $1+2$ » la valeur « 3 », puis à « 6 » la valeur « 4 », etc. Dans le doute, ou au besoin, ajoutons des parenthèses autour du calcul que l'on veut prioritaire. Ainsi « $1+2*3$ » vaut « 7 », mais « $(1+2)*3$ » vaut « 9 ».

Il existe en *mSL* de multiples niveaux de priorités, supposés incarner ceux que l'on retrouve en mathématique, et censés alléger l'écriture en évitant de multiplier les parenthèses. Ce n'est pas faux, mais souvenons nous que les parenthèses, même surnuméraires, ne coûtent rien si elles permettent de se conforter dans l'idée que ce que l'on a écrit est bien ce que l'on désirait obtenir. Notons que l'affectation, « $=$ » est considérée comme un opérateur de très basse priorité. « $y = a * x^2 + b * x + c$ » calcule la valeur du trinôme avant de l'affecter à la variable « y ».

Le problème des priorités ne se pose pas avec les fonctions : une fonction s'applique aux données encloses entre parenthèses placées à sa suite. Ainsi, dans la traduction de $r1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$:

```
r1 = (-b + sqrt (b^2 - 4 * a * c) ) / (2 * a)
```

il est nécessaire d'utiliser des parenthèses pour délimiter les arguments de la division, mais la fonction « *sqrt* » s'applique directement au résultat de l'expression entre parenthèses qui la suit (nous introduisons ici, en catimini, l'opérateur *opposé*, noté « $-$ » qui s'applique à la seule donnée située à sa droite — on peut considérer que « $-b$ » est une notation raccourcie pour « $0-b$ »).

Une expression est une combinaison « syntaxiquement correcte » de valeurs, de variables, d'opérateurs, d'appels de fonctions, et de parenthèses. On remettra à plus tard la définition formelle du terme « syntaxiquement correct ». Il est bon que l'intuition du programmeur se développe peu à peu : s'il comprend ce qu'il vient d'écrire,

la syntaxe est assez probablement correcte ! La plupart des erreurs sont dues à l'oubli d'une parenthèse ou d'un point-virgule...

Une instruction est une expression suivie d'un point-virgule « ; ». Alors que l'ordre précis d'exécution d'une expression dépend des opérateurs, fonctions, parenthèses et autres règles du langage, les instructions d'un programme sont exécutées les unes à la suite des autres, dans l'ordre d'écriture du programme.

Notons encore que l'on peut aussi rajouter des parenthèses autour d'une expression sans changer sa signification, et qu'une instruction, placée entre parenthèses, est assimilée à une expression. Remarquons encore l'indentation des lignes de code pour les expressions situées à l'intérieur des boucles, indentation qui n'a rien d'obligatoire mais qui permet au relecteur de mieux appréhender la structure d'un programme.

6.5 Recherche et Tri d'éléments

Effectuer une recherche ou un tri d'élément sont des opérations fréquentes en informatique. Nous savons déjà comment rechercher le plus grand ou le plus petit élément dans un tableau. Que faire si nous recherchons maintenant les deux plus petits éléments ? Reprenons notre algorithme, réécrit pour utiliser l'élément neutre de l'opération minimum :

```
GM_Log("Indice du plus petit élément d'un tableau.");
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s, e, i;
s = -1; e = 1.7976931348623158E+308; i = 0;
while (i < 10) (
    ((liste[i] < e) ? (s = i; e = liste[i];));
    i = i + 1;
);
var str = stralloc();
sprintf(str, "s vaut %d", s);
GM_Log(str);
```

Dans notre algorithme, nous gardons maintenant dans « s » l'indice du plus petit élément, et dans « e » sa valeur. Nous pourrions imaginer deux boucles successives pour rechercher nos deux plus petits éléments (nommons les « s » et « t ») :

```
GM_Log("Indice du plus petit élément d'un tableau.");
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s, t, e, f, i;
s = -1; e = 1.7976931348623158E+308; i = 0;
```



```

while (i < 10) (
    ((liste[i] < e) ? (s = i; e = liste[i])););
    i = i + 1;
);
t = -1; f = 1.7976931348623158E+308; i = 0;
while (i < 10) (
    ((liste[i] < f) ? (t = i; f = liste[i])););
    i = i + 1;
);
var str = stralloc();
sprintf(str, "liste[%d] vaut %d", s, e);
GM_Log(str);
sprintf(str, "liste[%d] vaut %d", t, f);
GM_Log(str);

```

Bien sûr, cette version découvre deux fois de suite la même valeur ! Pour être assuré que « t » va recevoir une valeur différente de celle de « s », il suffit, dans la seconde boucle, de rajouter la condition que le « i » courant soit différent de l'indice du plus petit élément précédemment trouvé, « s » :

```

(((liste[i] < f) && (i != s)) ? (t = i; f = liste[i])););

```

La condition que nous ajoutons, « *i est différent de s* », s'écrit « i!=s », où « != » (sans espace entre les deux caractères) représente l'opération de comparaison « différent ». Les deux conditions testées sont reliées entre elles par l'opérateur « && », « *et* », qui rend « *vrai* » si ses deux opérandes sont vrais. Nous utilisons ici des parenthèses (redondantes) autour de ces deux conditions, et autour de l'ensemble de la condition.

Cette fois, nous obtenons correctement dans « s » l'indice du plus petit élément (8), et dans « t » celui du second plus petit (3).

Peut-on découvrir les deux éléments « s » et « t » en une seule boucle, au lieu d'opérer avec deux boucles successives ? Par exemple, que donnerait ceci, et pourquoi ?

```

var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s, t, e, f, i;
s = t = -1;
e = f = 1.7976931348623158E+308;
i = 0;
while (i < 10) (
    ((liste[i] < e) ? (s = i; e = liste[i])););
    (((liste[i] < f) && (i != s)) ? (t = i; f = liste[i])););
    i = i + 1;
);

```

```

        i = i + 1;
    );
    var str = stralloc();
    sprintf(str, "liste[%d] vaut %d", s, e);
    GM_Log(str);
    sprintf(str, "liste[%d] vaut %d", t, f);
    GM_Log(str);

```

On pourrait modifier *ad vitam* ce programme pour rechercher les trois plus petits éléments, puis les quatre plus petits, etc... Après avoir réalisé que l'on ne peut découvrir le plus petit élément de rang « N » sans avoir découvert les « N-1 » précédents.

Introduisons maintenant une autre stratégie : nous allons *modifier* le tableau initial, pour introduire en tête le plus petit élément. Pour ceci, on cherche la position du plus petit élément, puis on échange celui-ci avec le premier. Voici le programme :

```

GM_Log("Placer en tête le plus petit élément.");
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s, e, i;
s = -1; e = 1.7976931348623158E+308; i = 0;
while (i < 10) (
    ((liste[i] < e) ? (s = i; e = liste[i])););
    i = i + 1;
);
// échange, par l'intermédiaire d'une variable « x »
var x; x = liste[0]; liste[0] = liste[s]; liste[s] = x;
var str = stralloc();
sprintf(str, "minimum : %d", liste[0]);
GM_Log(str);

```

La modification que nous avons introduite est l'échange proprement dit; celui-ci est réalisé par la ligne qui suit le commentaire – ligne débutant par le symbole « // », dont le contenu est ignoré par l'exécutant du script et destiné aux seuls lecteurs humains du programme. Le fait d'avoir placé en tête le plus petit élément nous assure que tous ceux qui suivent sont supérieurs ou égaux à celui-ci.

On peut effectuer la même opération, pour placer le second plus petit élément à la suite du premier : il suffit cette fois d'ajouter une seconde boucle, qui va débiter à l'indice suivant, 1 au lieu de 0 :

```

GM_Log("Placer en tête les plus petits éléments.");
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var s, e, i;

```

```

s = -1; e = 1.7976931348623158E+308; i = 0;
while (i < 10) (
    ((liste[i] < e) ? (s = i; e = liste[i])););
    i = i + 1;
);
// échange, par l'intermédiaire d'une variable « x »
var x; x = liste[0]; liste[0] = liste[s]; liste[s] = x;
s = -1; e = 1.7976931348623158E+308; i = 1;
while (i < 10) (
    ((liste[i] < e) ? (s = i; e = liste[i])););
    i = i + 1;
);
// échange, par l'intermédiaire d'une variable « x »
var x; x = liste[1]; liste[1] = liste[s]; liste[s] = x;
var str = stralloc();
sprintf(str, "minima : %d %d", liste[0], liste[1]);
GM_Log(str);

```

Sous cette forme, on voit émerger un schéma : placer à la suite les uns des autres les éléments découverts, jusqu'à obtenir le nombre d'éléments désirés : 2, 3, 5, 10... Sautons les étapes intermédiaires, et, au lieu de placer les boucles numéro 3, 4, etc. les unes à la suite des autres, construisons une nouvelle boucle *englobant* la recherche initiale, dans laquelle, pour généraliser notre approche, on désigne par « N » le nombre d'éléments du tableau, et « P » le nombre d'éléments recherchés :

```

GM_Log("Placer en tête les plus petits éléments.");
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var N = 10;
var P = 5;
var s, e, i, j, x;
j = 0;
while (j < P) (
    s = -1; e = 1.7976931348623158E+308; i = j;
    while (i < N) (
        ((liste[i] < e) ? (s = i; e = liste[i])););
        i = i + 1;
    );
    // échange, par l'intermédiaire d'une variable «
x »
    x = liste[j]; liste[j] = liste[s]; liste[s] = x;
    j = j + 1;

```

```
);
var str = stralloc();
sprintf(str, "minima : %d %d %d %d %d",
        liste[0], liste[1],
        liste[2], liste[3], liste[4]);
GM_Log(str);
```

L'exécution nous fournit le résultat attendu : « minima : -7 0 1 1 2 ». Mais comment ce programme fonctionne-t-il exactement ? Il se compose de deux boucles imbriquées, c'est-à-dire que l'une contient l'autre. La boucle la plus externe utilise « j » comme variable de contrôle :

```
while (j < P) (
    // ici se situe l'intérieur de la boucle
    j = j + 1;
);
```

A l'intérieur de la boucle, « j » prend les valeurs successives « 0 », « 1 », « 2 », etc., jusqu'à « P-1 ». Dans la boucle interne, qui est répétée « P » fois, le point de départ de la boucle est « j ». Au fur et à mesure de la progression de « j » dans la boucle externe, la boucle interne va décrire des segments de plus en plus petits du tableau : de 0 à N-1, puis de 1 à N-1, puis de 2 à N-1, jusqu'à la dernière valeur de « j », de N-1 à N-1, le segment ne contenant plus alors qu'une seule valeur.

Si l'on va jusqu'au bout, en positionnant « P » à 10, on obtient dans l'ordre les 10 plus petites valeurs du tableau, c'est-à-dire que l'on a, en fait, réalisé un *tri ascendant* de ce tableau. Il faut bien sûr modifier l'ordre d'écriture des différentes valeurs, qui devient :

```
sprintf(str, "minima : %d %d %d %d %d %d %d %d %d %d",
        liste[0], liste[1], liste[2], liste[3], liste[4],
        liste[5], liste[6], liste[7], liste[8], liste[9]);
```

et l'on obtient alors :

```
minima : -7 0 1 1 2 7 11 13 43 637
```

C'est satisfaisant, mais avouons le, cet algorithme n'est pas le plus efficace possible. Pour N éléments, la boucle externe va s'exécuter N fois, et, la taille des segments décrémentant régulièrement, la boucle interne va tester d'abord N éléments, puis $N-1$ éléments, puis $N-2$, etc. jusqu'à la dernière qui ne testera qu'un élément. Soit, au total, $\frac{N \times (N+1)}{2}$, formule déjà rencontrée. Lorsque le nombre N d'élément devient élevé, le temps nécessité par l'algorithme devient, en gros, proportionnel au carré du nombre d'éléments. L'on dit que la *complexité* de l'algorithme est $O(N^2)$ que l'on prononce « grand O de N^2 ». Pour des tableaux de petite taille, ce coût

reste acceptable, et, à l'intérieur du *Game Master*, nombre de tris s'adressant à des tableaux de quelques dizaines ou centaines d'éléments sont programmés en $O(N^2)$. Nous reviendrons ultérieurement sur ces concepts...

6.6 Tableaux

On l'a compris, manipuler des tableaux va devenir une opération fréquente dans nos activités de programmeur. Il se trouve qu'en *mSL* toute variable se comporte potentiellement comme un tableau, au travers de l'opération d'indexation, dans la mesure où toute variable a une valeur numérique, et que toute valeur numérique « pourrait » désigner une adresse dans la mémoire.

Cependant, *mSL* propose aussi une structure de donnée particulière, le *tableau*, ou *bloc*, que l'on peut créer au moyen des trois opérations suivantes :

B = data(...) cette construction spéciale crée un bloc renfermant toutes les valeurs numériques placées entre parenthèses. La syntaxe spécifique de cette forme est décrite en détail au § 13.2 page 200. La création est statique : même placée à l'intérieur d'une boucle, le résultat est toujours le même bloc, construit une fois pour toutes au chargement du script.

B = array(e1, e2, ..., eN) cette fonction admet un nombre arbitraire d'expressions, et fournit le tableau contenant ces valeurs. A la différence de « data », cette construction est dynamique, et chaque nouvelle exécution crée un nouveau tableau.

B = malloc(taille) cette dernière fonction crée un tableau dynamique, dont la taille est spécifiée en paramètre, et peut aller de « 0 » (tableau « vide ») à plusieurs millions d'éléments – en fonction de l'espace disponible dans la mémoire allouée au plug-in. Tous les éléments du tableau créé sont initialisés à « 0 ».

Dorénavant, nous n'utiliserons plus que des tableaux légitimement construits. Attention cependant, toutes ces opérations de construction peuvent rendre la valeur « 0 » s'il n'est pas possible d'allouer assez de mémoire pour exécuter la création du tableau demandé.

Lorsqu'une variable « v » désigne un tableau de taille « N », il est possible de lui appliquer les opérations d'indexation « v[i] » pour tout « i » compris entre « 0 » et « N-1 », ainsi que l'affectation indicée, « v[i]=expression ». On dispose en outre des fonctions suivantes :

z = isblock(v) rend « vrai » (« 1 ») ou « faux » (« 0 ») selon que le paramètre « v » est, ou non, un tableau construit par l'une des trois opérations décrites ci-dessus.

z = size(v) rend la taille (le nombre d'éléments) de « v » si « v » est un tableau construit par l'une des trois opérations décrites ci-dessus, « -1 » sinon.

Enfin, si votre algorithme n'a plus besoin d'un tableau, il peut libérer l'espace mémoire correspondant au moyen de l'opération « mfree » :

mfree(t) libère le tableau ; l'espace libéré pourra ultérieurement être réutilisé par un appel à « array » ou « malloc ».

Copie d'un tableau Un exemple typique d'utilisation de tableau dynamiquement alloué est celui d'un algorithme qui modifie le tableau qui lui est confié, comme le tri que nous venons de voir, dans une circonstance où nous ne voulons pas que ce tableau soit modifié ! Nous devons faire une *copie* de ce tableau, avant de lui appliquer notre algorithme de tri :

```
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var copie = malloc(size(liste));
var i = 0;
while (i < size(liste)) (
    copie[i] = liste[i]; i = i + 1;
);
```

Après cette opération, nous obtenons une copie conforme du tableau original, que nous pouvons dès lors modifier à notre gré.

Note : représentation des tableaux Démystifions la nature d'un tableau : ce n'est rien d'autre qu'un morceau de la mémoire centrale, dont la fonction « malloc » nous alloue, à la demande, une petite partie. Le programme ci-dessous dévoile le mystère de sa représentation :

```
var liste = data (7 0 2 637 -7 13);
var str = stralloc();
sprintf(str, "liste = %d", liste);
GM_Log(str);
sprintf(str, "contenu : %d %d %d %d %d %d %d %d %d %d",
    liste[-2], liste[-1], liste[0], liste[1], liste[2],
    liste[3], liste[4], liste[5], liste[6], liste[7]);
GM_Log(str);
```

Il ne fait rien de plus qu'imprimer le contenu du tableau de six éléments créé par « data », mais vous pouvez constater que nous avons choisi de faire débiter cette impression 2 éléments *avant* le début du tableau (indices -2 et -1), et de la terminer deux éléments *après* (indice 6 et 7). Le programme imprime les deux lignes suivantes :

```
liste = 2944173
contenu : 1684108385 6 7 0 2 637 -7 13 6 463375262
```

On y reconnaît (« contenu », valeurs n° 3 à 8) le contenu du tableau. La première et la dernière valeur constituent les « bornes » du tableau, et vont se retrouver à l'identique pour tous les objets créés par « malloc ». La seconde et l'avant-dernière valeur sont identiques, et égales au nombre d'éléments du tableau, « 6 ». Ces quatre valeurs supplémentaires permettent au système de reconnaître un tableau qu'il a alloué (fonction « isblock »), et expliquent comment la fonction « size » est réalisée : appliquée à un tableau « T », elle rend tout simplement « T[-1] ».

Quant à la valeur « liste », c'est un simple entier qui est le *numéro de la cellule mémoire*, ou *adresse*, du premier élément du tableau. Son utilisation, dans des constructions telles que « liste[i] » est donc absolument cohérente avec notre présentation de la mémoire au § 6.1 page 115.

Il ressort également de cette description qu'un tableau peut se manipuler comme une variable numérique quelconque :

```
var liste = data (7 0 2 637 -7 13);
var refer = liste;
```

Cette séquence nous donne dans « refer » la même valeur que celle de « liste ». Les éléments du tableau peuvent être manipulés aussi bien par l'intermédiaire du nom « refer » que par « liste », mais bien évidemment, toute modification du contenu des éléments sera visible par l'intermédiaire de chacun des deux noms. Dans le langage « C » nous parlerions de « pointeurs ». Nous dirons que nous sommes ici en présence de deux *références* au même tableau.

Si nous avons besoin de connaître le contenu d'un tableau à des fins de mise au point, il y a encore plus simple. Le module « Memory », décrit au § 2.21 page 82, va nous permettre d'afficher le contenu d'une partie de la mémoire. Dans la première cellule du pad, un clic droit affiche un menu. Choisissons « Memory » si cette option n'est pas encore sélectionnée. La partie basse du module affiche alors le contenu de la mémoire, à raison de 4 cellules par ligne. La colonne de gauche indique l'adresse de la première cellule de chaque ligne, et l'affichage débute par défaut en « 0 ». Pour afficher cette mémoire à partir d'une autre adresse, placer la souris sur la seconde cellule du pad, la case notée « @0 », et introduire au clavier l'adresse désirée. En cas d'erreur, taper « z » pour remettre à zéro. Indiquons l'adresse de début (moins 2) du tableau, soit « 2944171 ».

L'image de la fig. 6.3 page suivante nous montre le résultat. Ce qui est affiché (les informaticiens utilisent le terme de « *dump mémoire* ») n'est pas tout-à-fait semblable à ce que notre programme a imprimé. En effet, l'outil « Memory » tient compte des conventions internes utilisées par le Game Master, et traduit la valeur « 1684108385 » en « 'data' », et « 463375262 » en « '~' data' ~ » qui sont les noms symboliques des bornes de début et de fin de données.

Memory	@2944171	↔	0	2944279	0	36	1
2944171 :	'data'	6	7	0			
2944175 :	2	637	-7	13			
2944179 :	6	~'data'~	'ctxn'	4096			
2944183 :	639	2944173	5129	4			

FIGURE 6.3 – Représentation d'un tableau

De manière générale, tous les « blocs » de mémoire gérés par le système ont la même structure : une valeur spécifique précisant le type du bloc, la taille utile du bloc, le contenu, la taille encore, et une autre valeur spécifique marquant la fin du bloc.

Attention ! Il s'agit ici d'un affichage *post mortem*, c'est-à-dire que notre programme a déjà terminé son exécution au moment où nous effectuons ce *dump*. Si d'autres programmes sont actifs, la mémoire a pu avoir été réutilisée à d'autres fins, et ne plus contenir ce que nous attendons...

6.7 Fonctions définies

Toutes les opérations que nous avons vues jusqu'alors sont, nous l'avons signalé, des programmations d'algorithmes, qui s'appliquent à un petit nombre d'objets. Réciproquement, nous pouvons transformer un algorithme en une fonction, qu'il nous sera dès lors possible d'utiliser à notre guise, sans devoir le reprogrammer ou le recopier.

Prenons l'exemple de l'opération de copie de tableau que nous venons de voir au paragraphe précédent. Il nous faut analyser son interface, c'est-à-dire comprendre à quelles données elle s'applique, quel est son résultat, quelles sont les données intermédiaires qu'elle utilise, mais qui ne sont pas des entrées ou des sorties de l'algorithme.

Ces réponses sont faciles à apporter dans ce cas : l'opération s'applique à un tableau en entrée, génère un tableau en sortie, et utilise pour ses besoins propres une seule variable, « *i* ».

La syntaxe nous permettant de définir une fonction est la suivante :

```
function nom ( paramètres )
local ( variables )
( instructions ) ;
```

Ici, « *nom* » est un identificateur, qui nous permettra par la suite de désigner notre fonction. « *function* » et « *local* » sont des mots-clefs qui indiquent respectivement que l'on va définir une fonction et fournir la liste des variables locales, propres

à cette fonction. « *paramètres* » et « *variables* » sont des listes d'identificateurs. Enfin, « *instructions* » est la séquence d'instructions qui va permettre à la fonction de construire le résultat. Celui-ci n'apparaît pas explicitement dans cette construction sous la forme d'une variable : c'est la *dernière* valeur calculée par la fonction qui est fournie comme résultat de la fonction. Précisons que la partie « `local (variables)` » peut être omise s'il n'y a pas de variables locales à déclarer.

Dans notre cas précis, il y a un seul paramètre en entrée, pour lequel nous allons choisir un nom tel que « *source* », une variable locale, « *i* ». Il est souvent pratique d'utiliser une variable locale supplémentaire pour désigner le futur résultat. Prenons « *res* ». Voici notre algorithme, directement transcrit sous la forme d'une fonction :

```
function dupliquer (source)
local (i, res)
(
    res = malloc(size(source));
    i = 0;
    while (i < size(source)) (
        res[i] = source[i];
        i = i + 1;
    );
    res;
);
```

Le code reprend, au nom des variables près, l'algorithme déjà écrit et testé. Nous avons juste ajouté une dernière instruction, contenant la seule variable « *res* » afin de fournir le résultat de la fonction. Il est dès lors possible de faire appel à celle-ci :

```
var liste = data (7 11 43 0 2 1 1 637 -7 13);
var copie = dupliquer(liste);
```

Ecrivons de même une fonction telle que la somme des éléments d'un tableau :

```
function somme (T) local (i, s, t) (
    i = s = 0; t = size(T);
    while (i < t) ( s = s + T[i]; i = i+1; );
    s;
);
```

Notons que « *T* » et « *t* » sont bien des variables différentes, et que là aussi, nous devons placer la variable représentant le résultat comme dernière instruction de la fonction. Le fait de déclarer locales les variables « *i* », « *s* » et « *t* » fait que toute modification de ces variables reste purement locale à la fonction, et n'a aucun influence sur d'autres variables *externes* (dites aussi *globales*) qui porteraient le même

nom. De ce point de vue, les paramètres de la fonction, ici « T » se comportent comme des variables locales. Il est possible de modifier une telle variable, mais cela n'aura aucun effet sur la variable passée en argument.

Construisons encore une fonction, plus complexe, le *tri* d'un tableau, à partir de la dernière version décrite de notre algorithme :

```
function sort(tab)
local (N, i, j, e, s, x)
(
  N = size(tab);
  j = 0;
  while (j < N) (
    s = -1; e = 1.7976931348623158E+308;
    i = j;
    while (i < N) (
      ((tab[i] < e) ? (s = i; e = tab[i];));
      i = i + 1;
    );
    // échange par la variable «x»
    x = tab[j]; tab[j] = tab[s]; tab[s] = x;
    j = j + 1;
  );
  tab;
);
```

La fonction fournit comme résultat le tableau d'entrée, « tab », mais le contenu de celui-ci a été modifié par l'algorithme pour que ses éléments se retrouvent rangés par ordre croissant.

Tout ceci est bel et bien. Mais dans certains cas, plus de subtilité est nécessaire pour décider des spécifications d'une fonction. Par exemple, imaginons que l'on désire énumérer les éléments d'un tableau, c'est-à-dire les obtenir successivement, mais seulement quand la valeur suivante est nécessaire. Soit « tab » ce tableau. Utilisons une variable, « index » qui désignera le prochain élément à fournir. On débute bien sûr à l'index 0 :

```
var index = 0;
```

Puis, à chaque demande d'un élément suivant du tableau, on écrit :

```
v = tab[index]; index = index + 1;
```

La variable « v » ayant été antérieurement déclarée par « var ». Il est possible d'encapsuler cette ligne dans une fonction :

```
function suivant () local (x)
  ( x = tab[index]; index = index + 1; x);
```

et d'utiliser notre fonction sous la forme :

```
v = suivant();
```

C'est un peu plus pratique, il y a moins de risque d'erreur de recopie, et c'est plus facile à mettre à jour si, par exemple, on décide de changer le nom du tableau.

Que faire si la nécessité se généralise ? Si l'on doit manipuler simultanément plusieurs tableaux, voire plusieurs dizaines de tels tableaux ? Il serait possible de transmettre à la fonction « suivant » le tableau qui nous intéresse, éventuellement la valeur de l'index courant. Cependant, si les paramètres d'une fonction et ses variables locales peuvent être modifiés à *l'intérieur* de la fonction, ces modifications n'ont aucune répercussion à *l'extérieur* de celle-ci. La valeur transmise à la fonction de l'index serait modifiée dans le corps de la fonction, mais sa valeur externe demurerait inchangée, avec comme résultat pratique que la fonction fournirait toujours la même valeur.

Une fonction ne peut pas modifier les valeurs des variables passées en argument, mais, on l'a vu, elle peut modifier le *contenu* d'un tableau. On peut imaginer de transmettre à « suivant » comme paramètre unique un tableau, qui contiendrait l'état de notre système, à savoir, quel est le tableau concerné, quelle est la taille de ce tableau, quel est l'index en cours, et toute autre variable pertinente. Pour énumérer les éléments successifs d'un tableau, décidons que nous utiliserons l'adresse du tableau, sa taille, et un indice dans le tableau, qui désignera le prochain élément à fournir. Cette structure peut être construite dynamiquement par une fonction telle que celle-ci :

```
function initial(tableau)
(
  array(tableau, size(tableau), 0);
);
```

La fonction qui utilise une telle structure pour énumérer les éléments peut s'écrire :

```
function next(s)
local (i, r, t)
(
  i = s[2]; // indice
  t = s[0]; // tableau
  r = t[i]; // élément courant
  i = i + 1; // suivant
  (i >= s[1]) ? (i = 0); // si le dernier a été fourni
  s[2] = i; // conserver le nouvel indice
```

```

        r; // résultat
    );

```

Notre code peut dès lors s'écrire :

```

var w = initial(data(7 3 5 180 3 6 8 9 43 100 0 -6 2));
var str = stralloc();
sprintf(str, "suivant : %d", next(w));
GM_Log(str);
sprintf(str, "suivant : %d", next(w));
GM_Log(str);
sprintf(str, "suivant : %d", next(w));
GM_Log(str);

```

A l'exécution, les trois appels successifs à « next » nous délivrent, dans l'ordre, les valeurs 7, 3, puis 5.

En résumé de ce qui a été vu jusqu'alors, on peut dire que *mSL* fournit à ses utilisateurs une structure de données, simplissime, mais extrêmement puissante, le tableau, ainsi que la possibilité de manipuler ces tableaux au moyen de fonctions, à la syntaxe simple et sans surprise. Ces deux qualités en font un langage *impératif* classique, puissant et souple. Il nous est dès à présent possible de traduire en *mSL* tout algorithme traditionnel, et d'obtenir rapidement des résultats satisfaisants.

Note : représentation des fonctions Pendant que nous y sommes, démystifions également la nature d'une fonction : ce n'est ni plus ni moins qu'un segment de mémoire, un tableau contenant la description, sous une forme propre au *Game Master*, des caractéristiques de la fonction : le nom, les paramètres, les variables locales, et le *code* de cette fonction. Le code lui-même n'est pas représenté par des instructions machine comme dans le cas d'un programme « C » compilé, mais par des instructions « symboliques », une série de valeurs numériques qui n'ont de sens que pour l'évaluateur du Game Master qui leur associe des actions précises à effectuer (on parle alors de « *byte-code* »). Cette séquence de valeurs traduit donc dans cette convention la sémantique du code source de la fonction.

Pour le fun, définissons une fonction simple, et regardons sa représentation interne :

```

function fun (a, b) (
    a + 1/b;
);
var str = stralloc();
sprintf(str, "fun = %d", fun);
GM_Log(str);

```

La fonction calcule tout simplement la somme de son opérande gauche et de l'inverse de son opérande droit. L'exécution du programme nous donne :

```
fun = 2966928
```

Memory	@2966926	←→	0	2967046	0	47	1
2966926 :	'ccod'	29		cflags		0	
2966930 :	ccode	23		ccodesiz		6	
2966934 :	ccname	fun		ccparcnt		2	
2966938 :	ccloccnt	0		ccglbent		0	
2966942 :	ccglbtbl	2967000		ccowntbl		2967020	
2966946 :	ccptrtbl	2967032		_Mark_		a	
2966950 :	b	1030		1040		2054	
2966954 :	32	43		2		29	
2966958 :	~'ccod'~	'ccod'		35		cflags	

FIGURE 6.4 – Représentation d'une fonction

Muni de cette précieuse information, nous pouvons faire appel à « Memory », là encore en choisissant l'adresse 2966928 - 2. La figure 6.4 nous dévoile le contenu de la fonction : Nous avons affaire à une structure un peu complexe, semblable à un tableau (c.f. note 6.6 page 134), dont les bornes ont pour identifications symboliques « 'ccod' » et « ~'ccod'~ », block de mémoire de 29 éléments. Sans entrer dans les détails, l'outil nous indique les noms symboliques des différents champs qui définissent la fonction : « cflags » vaut 0 : c'est une fonction classique, « ccode » son code débute à l'adresse relative 23, ici « 2966928+23 », soit « 2966951 », « ccodesiz » : la fonction comporte 6 instructions, son nom est « fun », elle a 2 paramètres, 0 variables locales. Quelques informations supplémentaires désignent des tables du système, potentiellement utilisées par la fonction. « _Mark_ » enfin indique le début des données brutes : la liste des noms des paramètres, des variables locales, et enfin les six instructions représentant le code machine.

Ce qu'il est intéressant de retenir est que, tout comme un tableau, une fonction est représentée par un bloc de mémoire, et que le nom qui désigne la fonction est tout simplement associé à l'adresse de la fonction.

Si, après avoir défini notre fonction « fun » ci-dessus, nous affectons sa valeur à une variable « v » :

```
var v = fun;
```

nous obtenons dans « v » une nouvelle *référence* au même bloc de mémoire. Cette variable va donc se comporter comme la fonction, et, par exemple, l'expression « v(3,2) » va nous fournir « 3.5 » comme attendu.

Notons aussi qu'une fonction peut être reconnue par la fonction « isfun » :

z = isfun(v) rend « vrai » (« 1 ») ou « faux » (« 0 ») selon que le paramètre « v » est, ou non, une fonction primitive ou définie.

Cette valeur « fonctionnelle » est donc un simple nombre, qui peut également être conservée dans un élément d'un tableau, passée en paramètre à une fonction, etc. Cet ensemble de propriétés font que dans le langage mSL les fonctions sont, selon le terme parfois employé par les informaticiens, des « *first class citizen* », ou « *citoyens de première classe* », à l'instar des tableaux et des « simples » nombres.

Objets ? Reprenons l'exemple de notre fonction « next ». Nous réalisons maintenant qu'il est également possible d'encapsuler, dans notre structure de données, la fonction avec son état. Supposons que notre but soit d'obtenir un « round-robin » sur un tableau de nombres, qui fournisse à chaque appel le nombre suivant du tableau. Décidons que notre structure, comportant maintenant quatre éléments, va représenter, dans l'ordre : la fonction qui fournit l'élément suivant, l'adresse du tableau auquel elle va s'appliquer, sa taille, et un indice dans le tableau, qui désignera le prochain élément à fournir. Soit « gen_RR » cette fonction de génération de l'élément suivant. Elle reçoit, en unique paramètre, cette structure de quatre éléments. Son écriture est identique à la version antérieure, au décalage des indices près, pour laisser libre l'indice « 0 » du tableau.

```
function gen_RR(S)
local (i, r, t)
(
    i = S[3]; // indice
    t = S[1]; // tableau
    r = t[i]; // élément courant
    i = i + 1; // suivant
    (i >= S[2]) ? (i = 0); // si le dernier a été fourni
    S[3] = i; // conserver le futur indice
    r; // résultat
);
```

Rédigeons maintenant la fonction qui construit notre nouvelle structure de données. Elle a besoin, en entrée, du tableau sur lequel la fonction « gen_RR » va opérer, mais va également conserver la *référence* à cette fonction comme premier élément :

```
function make_RR(tableau)
(
    array(gen_RR, tableau, size(tableau), 0);
);
```

Construire un nouveau générateur, « gen1 », s'écrit tout simplement :

```
var gen1 = make_RR(data(7 3 5 180 3 6 8 9 43 100 0 -6 2));
```

Voici enfin la nouvelle version de notre opération « next » qui fournit l'élément suivant à partir d'un générateur, en appelant tout simplement la fonction associée au générateur :

```
function next(gen)
local (f)
(
    f = gen[0];
    f(gen);
);
```

Imprimons les résultats de quatre appels successifs à « next » :

```
sprintf(str, "suivants : %d %d %d %d",
    next(gen1), next(gen1), next(gen1), next(gen1));
GM_Log(str);
```

Nous obtenons les valeurs attendues :

```
suivants : 7 3 5 180
```

Cette nouvelle version nous semble tout aussi opérationnelle que la précédente, mais ne paraît guère apporter d'amélioration. Il y a eu pourtant un grand pas en avant de réalisé, lié à l'*abstraction* de la méthode de production. L'opération « next » n'a plus à savoir *comment* fonctionne le générateur, mais doit simplement lui *demande*r de fournir l'élément suivant. Le programmeur chevronné réalisera que nous avons créé un *objet*, au sens de la programmation objet. Nous pouvons non seulement créer d'autres générateurs de la même classe :

```
var gen2 = make_RR(data(8801 8802 8833 8834 1002 1007));
```

mais encore, par la même approche, créer des générateurs utilisant des algorithmes différents. Ainsi, voici tout ce dont nous avons besoin pour créer un générateur d'entiers successifs à partir d'une valeur « N » :

```
function gen_int(S) local(r) (r = S[1]; S[1] = r+1; r);
function make_int(N) (array(gen_int, N));
var gen3 = make_int(100);
```

La même fonction, « next », nous permet maintenant de demander à « gen3 » de nous fournir ses éléments suivants, et nous obtenons, comme attendu, « 100 101 102 103 etc. ».

Il est légitime de qualifier *d'objets* les données comme « gen1 », « gen2 », etc. que nous pouvons créer de la sorte, dans la mesure où elles satisfont les concepts *d'abstraction* (l'implémentation est cachée), *d'encapsulation* (l'objet dispose de ses propres données et de ses propres fonctions), et de *polymorphisme* (même interface, « next », pour des implantations différentes, comme « gen1 » et « gen3 »). Nous pourrions ajouter le concept d'héritage, mais il faudrait réécrire un certain nombre d'opérations. Ceci pour dire qu'il sera possible, au besoin, de transposer en *mSL* des approches propres à la technologie objet.

Fonctionnelles Reprenons un de nos anciens exercices, la somme des éléments d'un tableau, en faisant cette fois « abstraction » de la fonction. Définissons ceci :

```
function reduce(A, f, N)
local (r, i, s)
(
  s = size(A);
  r = N;
  i = 0;
  while (i < s) (
    r = f(r, A[i]);
    i = i + 1;
  );
  r;
);
```

Cette fonction accepte trois paramètres : un tableau, « A », une fonction, « f », et une valeur « N », que l'on transmet comme « *élément neutre* » de la fonction. La fonction « reduce » a pour finalité de « *réduire* » le tableau « A » par la fonction « f », c'est-à-dire, symboliquement, d'appliquer cette fonction « f » « *entre* » les éléments de « A » – imaginons que le tableau contienne les valeurs « 6 2 5 3 »; appliquer l'addition « *entre* » les éléments de ce tableau revient à calculer « 6 + 2 + 5 + 3 ».

On donne parfois le nom de « *fonctionnelles* » aux fonctions qui acceptent d'autres fonctions comme paramètres. A ce titre, l'opération « reduce » est relativement classique. Notons que sa programmation gère implicitement le cas particulier où le tableau est *vide* (c'est-à-dire ne contient aucun élément) : le résultat est alors l'élément neutre de la fonction. Dans tous les autres cas, la fonction « f » est appliquée à chaque élément du tableau.

Pour effectuer, grâce à « reduce », le calcul de la somme des éléments d'un vecteur, on peut lui transmettre une fonction qui effectue cette opération d'addition, par exemple :

```
function plus(a,b) (a+b);
```


On peut dès lors définir un tableau :

```
var T = data(2 7 11 30 3 8 9 -2 -6 4 -2 5);
```

et en calculer la somme des éléments :

```
var r = reduce(T, plus, 0);
```

Le troisième élément de l'appel est « 0 », élément neutre de l'addition. L'impression par :

```
sprintf(str, "Réduction = %.4f", reduce(T, plus, 0));
GM_Log(str);
```

nous donne³ :

```
Réduction = 69.0000
```

Notons qu'il est possible de transmettre également les fonctions primitives du langage, telles « max » et « min », associées à leurs éléments neutres :

```
sprintf(str, "max = %.4f",
    reduce(T, max, -1.7976931348623158E+308));
GM_Log(str);
sprintf(str, "min = %.4f",
    reduce(T, min, 1.7976931348623158E+308));
GM_Log(str);
```

Nous fournit :

```
max = 30.0000
min = -6.0000
```

Encore mieux, il est même possible d'utiliser comme fonctions les *opérateurs* du langage, en les plaçant entre parenthèses :

```
sprintf(str, "sum = %.4f", reduce(T, (+), 0));
GM_Log(str);
sprintf(str, "prod = %.4f", reduce(T, (*), 1));
GM_Log(str);
```

expressions qui nous donnent :

```
sum = 69.0000
prod = -4790001600.0000
```

3. Nous utilisons ici un autre format d'impression, « %.4f » dont le résultat est d'imprimer la valeur sous forme décimale (« f » pour « *floating point* »), avec 4 chiffres décimaux après le point.

Notons que l'on peut obtenir nombre d'opérations utiles par un choix approprié de la fonction transmise à « `reduce` » :

```
function f1(a,b) (a+1/b) ;
function f2(a,b) (a+b*b) ;
function f3(a,b) (a+abs(b)) ;
function f4(a,b) (max(a,abs(b))) ;
```

nous permettent de calculer respectivement la somme des inverses des éléments d'un tableau, la somme des carrés des éléments, la somme des valeurs absolues de ces éléments, et enfin le plus grand, en valeur absolue, des éléments du tableau. Il convient à chaque fois de passer l'élément neutre approprié, selon les cas, aux opérations d'addition et de maximum.

Exemple 4 Une autre fonctionnelle, bien classique, est l'opération « `map` » qui applique une même fonction « `f` » à chacun des éléments d'un tableau. Ainsi, « `map(T, abs)` » va appliquer l'opération « *valeur absolue* » à tous les éléments de « `T` ». Elle est fort simple à écrire, travail laissé au lecteur.

Remarque Ces exemples montrent qu'il est tout aussi légitime de qualifier de « *fonctionnel* » le langage `mSL`, dans la mesure où les fonctions sont des objets de première classe, et, nous le verrons ultérieurement, qu'elles ont d'autres propriétés comme la détection et l'optimisation des récursions terminales.

6.8 Chaînes de caractères

Revenons sur les chaînes de caractères brièvement introduites au début du chapitre. En `mSL`, ce sont des objets désignés par un nombre entier⁴. Ce sont des données opaques, qui ne sont manipulables que par l'intermédiaire de fonctions spécialisées. Les fonctions de base sont les suivantes :

`z = stralloc()` fournit une nouvelle chaîne de caractères.

`z = "une suite de caracteres"` est une autre manière de définir une chaîne de caractères.

`strfree(s)` libère la chaîne de caractères fournie par « `stralloc` ».

`z = strtmp()` fournit une chaîne de caractères temporaire. Elle n'a pas besoin d'être libérée après usage, mais peut éventuellement (au bout d'un long moment) être réutilisée par le système.

4. pour la petite histoire, ce nombre est compris entre 90000 et 100000.

Les fonctions de manipulation de chaînes de caractères sont nombreuses. Nous avons déjà rencontré « `GM_Log` », opération d'impression d'un message dans le *log* du *Game Master*, et « `sprintf` », fonction d'édition d'une chaîne de caractères. « `sprintf` » est une fonction aux spécifications relativement complexes. Elle est directement héritée de sa base JSFX, laquelle la tient en droite ligne du langage « C ». Il n'est pas d'un grand intérêt d'en recopier ici la description, que l'on retrouvera aisément sur internet, dans le manuel du langage JSFX : [5].

6.9 Erreurs

Une introduction à la programmation ne serait pas complète si l'on n'introduisait pas le concept *d'erreur*. Un programme est erroné si on ne parvient pas à le compiler (une erreur est signalée lorsque l'on tente de le charger dans le GM), ou si son exécution ne nous fournit pas les résultats attendus. Ce sont deux types d'erreurs de natures différentes : dans le premier cas, la syntaxe est incorrecte, dans le second cas l'algorithme que l'on a programmé ne résout pas le problème.

Dans le premier cas, le compilateur tente, peu ou prou, de signaler l'erreur :

```
// Un programme avec une erreur :  
// parenthèse manquante...  
var a = 2 + 3 * (5^2;  
a + 8;
```

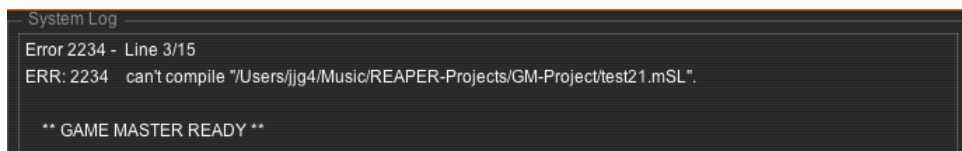


FIGURE 6.5 – Signalement d'une erreur

Le message obtenu (c.f. fig. 6.5) va indiquer un numéro d'erreur (c.f. chapitre 17 page 249), un numéro de ligne et une position dans la ligne.

6.10 Une première conclusion

Ce chapitre vous a proposé un premier tour d'horizon du langage *mSL*, en présentant de manière relativement concrète, et parfois fort détaillée, les notions de mémoire, variable, tableau, fonction, algorithme, etc., sans vouloir prétendre à l'exhaustivité des opérations et particularités du langage. N'hésitez pas à le relire si certains aspects vous semblent encore flous !

Chapitre 7

Présentation détaillée du langage mSL

Cette partie décrit plus extensivement les spécifications du mini-langage de script, *mSL*, intégré au Game Master (mais plus généralement destiné à être incorporé à des effets écrits en *JSFX*, fonctionnant sous REAPER [1]).

7.1 Concepts généraux

Cette section décrit *mSL*, un langage de programmation qui peut être intégré à l'intérieur d'un plug-in écrit en JS. *mSL* (pour « *micro Script Language* ») est lui-même écrit en JS (*jesusonic*, aka *eel2*), et est disponible sous la forme de fichiers d'importation, *mSL_Clib.jsfx-inc* et *mSL_Xlib.jsfx-inc*. Le système se compose d'un compilateur, générant un *byte-code* pour une machine à pile, d'un interprète pour ce *byte-code*, et d'un ensemble de procédures run-time.

Le run-time est relativement élaboré, puisqu'il propose une *gestion dynamique de la mémoire*, avec *allocation et désallocation explicites ou implicites*, par utilisation d'un *garbage collector conservatif*, une *gestion de processus légers*, un *scheduler préemptif*, une gestion de *messages interprocessus*, et un ensemble d'API facilitant l'*introspection* et les *interactions* avec le Game Master.

Les spécifications de *mSL* sont très proches de celles de *eel2*. Dans la mesure où *mSL* peut uniquement être utilisé dans un plug-in JSFX, ou dans un logiciel qui supporte JS, l'utilisateur potentiel¹ est susceptible d'être familier avec la syntaxe JSFX. Cette partie se concentre donc, après une brève présentation des spécificités de *mSL*, sur l'utilisation du langage et son intégration au sein d'un plug-in JSFX.

1. Au sens relativement restrictif de : « une personne qui écrit un plug-in en « JSFX » et désire y incorporer le langage *mSL* ».

L'utilisateur débutant (voire confirmé) en programmation trouvera une introduction très informelle à *mSL* au chapitre précédent.

7.1.1 Le langage

Le langage *mSL* est très similaire à *eel2*. Comme lui, il ne propose comme unique type de données que des *nombres*, sous la forme de *flottants 64 bits*, permet l'accès à la mémoire de travail *JS*, par indexation, et à la mémoire partagée, au moyen de la construction « `gmem[x]` ». La plupart des opérations et des fonctions de JSFX sont disponibles avec des noms, syntaxes et sémantiques équivalents. Il existe cependant des différences, dues à des impossibilités de programmation ou à des choix de l'implémenteur, dont la plupart sont documentées dans les paragraphes suivants.

7.1.1.1 Le langage, et ses différences avec *eel2*

Variables

1. Les noms de variables débutent avec une lettre ou le caractère « *souligné* », « `_` », et ne contiennent que des lettres, chiffres et soulignés. Les majuscules et les minuscules sont *distinctes*, ce qui fait, par exemple, que `abc`, `Abc` et `ABC` désignent trois variables différentes. Il est possible d'utiliser des symboles de plus de 8 caractères. Le compilateur remplace alors le symbole initial par un symbole « compacté » de 8 caractères. Comme il y a nécessairement une perte d'information, il est possible que deux symboles « longs » génèrent le même symbole compacté. A titre d'exemple, voici quelques symboles « longs » et leur représentation compactée :

```
Color_Dark_Gray => 6lCLyDRr
configuration   => 5fcoviuF
free_blocks_chaining => 0FDf63CG
processus_status => 7gprouYS
```

Les représentations compactées débutent toutes par un chiffre. Dans la pratique, ceci implique que deux noms différents, un court (8 caractères, ou moins) et un long (plus de 8 caractères), utilisés dans le même programme, ne peuvent pas désigner accidentellement la même variable. Le risque de collision existe cependant pour deux identificateurs longs. Notons cependant que l'algorithme utilisé permet de générer 218340105584896 symboles distincts, et que les risques de collisions accidentelles sont donc extrêmement réduits. Qui plus est, les variables devant toutes être déclarée, l'erreur « identificateur déjà déclaré » signalerait ce cas à l'utilisateur.

2. Toutes les variables *doivent être déclarées*, avec le mot-clef `var` ou encore `loc`, similaire – nous verrons ultérieurement la nuance. Voici un exemple d’une telle déclaration :

```
var a=10, b, c=a+5, d=2*$pi;
```

3. Cependant, les déclarations portent sur l’ensemble du texte source, et peuvent être placées après d’autres instructions, les variables pouvant même être utilisées avant leur déclaration, à condition qu’elles soient déclarées quelque part dans le texte source.

```
a = 3;
b = 5<< a;
c = b | 3; //returns 43
// declarations may follow
var a, b, c;
```

Constantes Toutes les valeurs utilisées dans les programmes sont des valeurs flottantes en 64 bits. Il existe différentes notations pour représenter ces valeurs, en fonction de l’usage qui va en être fait.

1. La notation numérique « classique ». Il s’agit des notations entière et décimale, de la notation hexadécimale, qui débute par `0x` ou `0X`, suivi d’une séquence de chiffres hexadécimaux (0 à 9 et A à F), ou encore de la notation binaire, qui débute par `0b` ou `0B`, et ne comporte que des 0 et des 1.
2. Les caractères, et les caractères multi-octets, qui sont constitués d’une séquence de 1 à 4 caractères placés entre apostrophe, comme `'A'` ou `'u2+', '`. Cette notation a la même signification qu’en *JS*.
3. Les *symboles* constituent une notation compacte pour les identificateurs, tels que les noms de variables, débutant par une lettre ou un souligné, et ne comportant que des lettres, des chiffres, et le caractère souligné (c.f. discussion au § 7.1.1.1 page ci-contre). Cette notation est introduite par le caractère apostrophe renversée (backquote), comme dans ``symbol`. Cette notation génère des valeurs entières comprises entre `0x02C00000000000` et `0x10000000000000`.
4. Les constantes symboliques débutent par le caractère `$`, suivi d’un symbole. Les constantes symboliques `$pi` et `$e`, représentant des constantes mathématiques bien connues, sont prédéfinies, et une application peut en ajouter de nouvelles.

Opérations primitives La sémantique des opérations est similaires dans les deux langages. Notez cependant les différences suivantes :

1. En *JS*, les opérations `&&` et `||` fournissent une valeur logique, 0 ou 1. *mSL* utilise la sémantique du langage “C”, et l’expression `2 && 3` qui fournit 1 en *JS*, fournit 3 en *mSL*. Les deux résultats correspondent à une valeur « vrai », si bien que dans un test, l’expression a la même signification dans les deux langages.
2. En *mSL*, les opérateurs d’affectation (`=`, `+=`, etc.) ont une priorité plus basse que la condition `? :`. Il peut donc être nécessaire, dans certains cas, d’ajouter des parenthèses pour obtenir la sémantique désirée.

Le tableau ci-dessous résume les différentes priorités des opérateurs du langage. En cas de doute, ou pour rendre plus lisibles les programmes, ne pas hésiter à ajouter des parenthèses autour des sous-expressions.

Niveau	Opérations	Priorité	Nature
1	() et []	-	Parenthèses, fonctions, indexation
2	- + ! ~	Droite à gauche	Opérations unaires
3	^	Gauche à droite	Elévation à la puissance
4	* / %	Gauche à droite	Opérations multiplicatives
5	+ -	Gauche à droite	Opérations additives
6	<< >>	Gauche à droite	Décalages
7	< <= >= >	Gauche à droite	Comparaisons non associatives
8	=== == != !==	Gauche à droite	Comparaisons non associatives
9	& ~	Gauche à droite	« et », « ou exclusif » bit à bit
10		Gauche à droite	« ou » bit à bit
11	&&	Gauche à droite	« et » logique
12		Gauche à droite	« ou » logique
13	? :	Droite à gauche	Forme conditionnelle
14	= += -= *= etc.	Droite à gauche	Affectation
15	;	Gauche à droite	Séquence d’instructions
16	,	Gauche à droite	Séquence de valeurs

Boucles Les constructions `loop` et `while` sont disponibles, avec la même sémantique qu’en *JS*. Cependant, seule la forme *while* de syntaxe « `while` (condition) (corps) ; » est implémentée.

Fonctions définies

1. La syntaxe de la définition des fonctions est identique en *mSL* et en *JS*. Cependant, les virgules séparant les arguments et les variables locales sont obli-

gatoires en *mSL*, alors qu'elles sont facultatives en *JS*. Contrairement à *JS*, les fonctions peuvent être récursives, la récursivité croisée est permise, et les fonctions peuvent être référencées avant leur définition :

```
fib(25); //returns 75025
function fib(n)
(
    (n < 2) ? n : fib(n-1) + fib(n-2);
);
```

2. Les paramètres et les variables locales sont alloués dans la pile d'exécution, à chaque appel de fonction. Ces variables ne sont pas attachées à la fonction, et ne sont donc pas rémanentes comme en JS, ce qui n'aurait d'ailleurs pas de sens pour des fonctions récursives. De même, les variables locales ne sont pas initialisées lors d'un appel de fonction, et reçoivent les valeurs qui se trouvent être dans la pile à ce moment là. Juste pour montrer ce fait (et ce n'est en rien un exemple à suivre, mais une remarque à oublier!), dans la fonction *f* ci-dessous, il se trouve que la troisième variable locale, *z*, capture la valeur du troisième élément de la pile, 1331, qui est le résultat du calcul 11^3 laissé par l'exécution de l'expression précédant l'appel de la fonction :

```
function f()
local (x, y, z)
( z; );
7+8*11^3;
f(); // returns 1331
```

3. L'*arité*, c'est-à-dire le nombre d'arguments des fonctions définies, est fixe. Une fonction sans argument est dite *niladique*, une fonction à un argument est *monadique*, une fonction à deux arguments est *dyadique*, etc. Il n'est pas possible de *surcharger* un nom de fonction, c'est-à-dire d'utiliser le même nom pour désigner, par exemple, une fonction à un argument et une fonction à trois arguments. Cependant, il est possible de définir des fonctions *variadiques*, c'est-à-dire des fonctions qui acceptent un nombre arbitraire d'éléments. Une telle fonction se définit avec, dans son en-tête, deux paramètres séparés non pas par une *virgule*, mais par un *point-virgule*. La sémantique est similaire à celle de l'en-tête de la procédure principale d'un programme "C" : le premier paramètre est le nombre effectif d'arguments, le second paramètre est la référence d'un tableau contenant ces arguments. Avec une définition telle que `function f(x ; y)`, et l'appel suivant : `f(12, 8, 5)`, *x* est égal à 3, car il y a trois paramètres dans cet appel, *y*[0] vaut 12, *y*[1] vaut 8, et *y*[2] vaut 5. En voici un exemple :

```
// variadic function
function f(a;b)
local (s,i)
(
    s = i = 0;
    loop (a, s += b[i]; i+= 1; );
    s;
);
f(2, 3, 5, 6) + f(5, 7) + f(); // returns 28
```

Notons que le tableau des arguments, *b* dans l'exemple ci-dessus, est un objet temporaire, alloué dans la pile, qui est valide à l'intérieur de la fonction, mais n'est pas utilisable hors de celle-ci.

4. Les fonctions sont des *objets de première classe*, ce qui signifie qu'elles peuvent être affectées à des variables, passées comme paramètres, rendues comme résultat d'une fonction, etc.

```
// functions as values
function toto(a, b, c) (a+2*b^c);
var fun = toto;
fun(3, 5, 2); //returns 53
// functions as arguments
function f1(x,y) (x+3*y);
function f2(x,y) (x*y+7);
function g(f, a, b) ( 1+f(a,b); );
g(f1, 3, 5) + g(f2, 5, 6); //returns 57
```

Les opérateurs et fonctions primitives sont également des objets de première classe. Une fonction primitive est directement manipulable dans le langage. Un opérateur, placé entre parenthèses, se comporte comme une telle fonction :

```
var f1 = sin; f1(2.2); // returns 0.808496
var f2 = (+); f2(62,39); // returns 101
```

5. La fonction primitive `return` permet de sortir d'une fonction, en rendant son résultat comme résultat de la fonction.

```
function f(u) (
    u == 3 ? return(31);
    2*u + 5;
);
f(2)+f(3); //returns 40
```

6. L'interprète détecte et optimise les cas de *récurtivité terminale*, c'est-à-dire les cas où un appel de fonction est immédiatement suivi d'une instruction `return` implicite ou explicite. L'exemple suivant montre une addition, outrancièrement complexe, de deux entiers :

```
function f1(x, y) (
    x <= 0 ? return(y);
    f2(x, y);
);
function f2(u, v)
local (x, y, z, t) (
    x = u; y = v;
    z = x-1; t = 1;
    f3(z, y, t);
);
function f3(a, b, c) (
    f1(a, b+c);
);
f1 (5000000, 5000000);
```

Dans la dernière ligne, l'appel de la fonction `f1` fournit la valeur attendue, 10000000, et effectue un total de 15 millions d'appels récursifs des fonctions `f1`, `f2` et `f3`, mais ne consomme aucun espace dans la pile d'exécution, chaque appel se traduisant par un « simple » branchement.

Fonctions du système Certaines des opérations primitives du langage JS sont disponibles dans *mSL* sous le même nom, et avec la même sémantique. Dans d'autres cas, le nom peut être différent, ceci étant dû au choix de limiter ces noms à 8 caractères.

- L'opération `time_precise()` est disponible sous le nom `ptime()`
- L'opération `__memtop()` est disponible sous le nom `memtop()`
- Les variables systèmes telles que `srate`, `num_ch`, `tempo` sont disponibles sous la forme des fonctions niladiques, `srate()`, `num_ch()`, `tempo()`, etc.

Notons que *mSL* ne propose pas de fonctions graphiques.

Chaînes de caractères Les chaînes de caractères en *mSL* sont allouées depuis un petit ensemble de chaînes prédéfinies, soit de manière implicite par le compilateur, lorsqu'une constante de type chaîne est utilisée dans un programme, soit de manière explicite par le programme, au moyen de la fonction spécifique d'allocation, `stralloc`. Les chaînes sont référencées à l'instar des autres valeurs en *mSL*, au moyen de variables ordinaires.

```
var s1 = "This is a string";  
var s2 = stralloc();
```

Ces chaînes peuvent être libérées explicitement par les programmes lorsqu'elles ne sont plus utilisées, au moyen de la fonction `strfree` :

```
strfree(s1);  
strfree(s2);
```

Le langage *mSL* ne propose qu'une partie des opérations sur chaînes de caractères disponibles en *JSFX* : `strlen`, `strcpy`, `strcat`, `strcmp`, `stricmp` et `sprintf` sont implémentées. Les fonctions *JSFX* `str_getchar` et `str_setchar` sont disponibles sous les formes `strgetch` et `strsetch`. Enfin, une nouvelle fonction, `compile`, permet de compiler une chaîne représentant une séquence d'instructions mSL, et fournit le résultat de cette compilation sous la forme d'une fonction niladique, qui peut être exécutée ultérieurement :

```
var c = compile("var a=3, b=5; 2*a+b;");  
c(); // returns 11
```

Une fonction particulière, `strtmp`, permet d'allouer temporairement une chaîne de caractères, à l'instar de « # » de *JSFX*. La valeur obtenue n'a pas besoin d'être libérée, et est réputée valide pour les quelques instructions mSL qui suivent.

Accès à la mémoire *mSL* fournit l'opérateur d'indexation, représenté par les crochets, « [] », qui a la même sémantique qu'en *JSFX*. Une séquence telle que `a=100;` `a[7];` a le même effet dans les deux langages, et fournit la valeur du mot mémoire situé en 107. Ceci est vrai également pour des opérations telles que `gmem[x];` et `gmem[x]=y;`.

En outre, *mSL* utilise une partie de la mémoire du plug-in, qu'il gère en mémoire dynamique. Les allocations dans cette mémoire sont réalisées par l'opération `malloc`, et les libérations de cette mémoire par `mfree`.

Importations *mSL* peut importer des fichiers externes contenant des instructions mSL. La syntaxe de cette opération consiste en un mot-clef spécifique, `import`, suivi d'une chaîne de caractères contenant le nom du fichier à importer, qui doit avoir l'une des extensions « .txt » ou « .mSL », et se termine par un point-virgule. Ex :

```
import "file.txt";
```

Les instructions d'importation peuvent être mélangées avec les autres instructions.

7.2 Quelques caractéristiques de l'implémentation

7.2.1 Gestion de la mémoire

mSL propose une gestion dynamique de la mémoire, au travers des opérations « `malloc()` » et « `mfree()` ». Il existe aussi un « *garbage collector* », qui permet de récupérer certains blocs (mémoire partagée entre différents processus) qui ne peuvent pas aisément être libérés explicitement. Le « *garbage collector* » est de type conservatif, c'est-à-dire qu'il essaye de ne libérer que les blocs qui ne peuvent pas être accessibles au travers d'une structure ou d'une autre.

7.2.2 Gestion des chaînes de caractères

La gestion des chaînes de caractères est également dynamique. Le système propose un petit nombre de chaînes de caractères, qui sont allouées implicitement (utilisation de constantes de type « chaîne de caractères » dans un programme), ou explicitement par la procédure `stralloc()`, et libérées par la procédure `strfree()`. Les chaînes ainsi allouées sont assurées de perdurer durant toute l'exécution du programme. Notons cependant qu'il n'y a pas actuellement de récupération automatique des chaînes allouées, devenues inutilisées, et non libérées.

7.2.3 Compilation dynamique

La procédure « `compile()` » permet de compiler la chaîne de caractères passée en paramètre. Le résultat est un code exécutable, assimilé à une fonction *niladique* anonyme. La chaîne de caractères peut comporter des déclarations, des insertions de fichiers extérieurs, etc. Elle est compilée dans le contexte du code appelant, ce qui veut dire qu'elle peut faire référence aux fonctions et variables définies par ce code.

7.2.4 Blocs

mSL définit une unique structure de données, dite *mBlock* (pour “Memory Block”), ou plus simplement, *block*, qui est juste un segment de mémoire, avec un *en-tête* et une *terminaison* spécifique. Par bien des aspects, un tel bloc se comporte comme un tableau dans un langage de programmation classique. Un bloc peut être créé au sein d'un programme *JS* par la procédure `mSL_make_block` à partir d'une adresse mémoire fournie à la procédure, ou encore, dynamiquement, par un appel à l'allocateur de mémoire `mSL_malloc`. Dans un programme *mSL*, une création dynamique peut être réalisée par un appel à `malloc()` ou par la procédure `array()`. Un bloc peut aussi être créé, au moment de la compilation, mais toujours par allocation dy-

namique de mémoire, au moyen de la directive `data`, dont la syntaxe comporte le mot-clef « `data` » suivi d'une liste de constantes entre parenthèses :

```
var bl = data(1,2,3.4, hello, world, 'ABCD',
             0x3ffa27342c5, $pi, 'x');
```

Dans cette expression, toutes les valeurs sont des nombres, et, par exemple, « `hello` » a pour valeur la codification du symbole correspondant, sous la forme d'un nombre flottant 64 bits. Notons aussi que les *virgules* qui séparent les valeurs entre elles sont facultatives, et peuvent être remplacées par des espaces ou des passages à la ligne.

Les Blocs sont utilisés pour représenter toutes les structures internes du système *mSL*. Un type leur est associé, sous la forme d'une valeur multi-chars de 4 octets, mais ce type est utilisé comme un simple étiquetage du bloc. Les programmes *mSL* peuvent accéder aisément aux informations concernant un bloc : si `bl` est un tel bloc, `bl[-2]` est son type, `bl[-1]` son nombre d'éléments, N , et les expressions `bl[0]` à `bl[N-1]` fournissent l'accès, en lecture ou écriture, à ses éléments. Le type d'un bloc, tel qu'il est par exemple défini par une directive `data()`, est par défaut « `'data'` » s'il n'est pas explicitement modifié.

Les blocs peuvent également être utilisés en *mémoire associative*, dans laquelle les éléments de rang pair représentent les clefs, et les éléments de rang impair les valeurs associées. Ainsi :

```
var tab = data(item1 1 item2 2 hello 3.4 world 333
              'ABCD' 0x3ffa27342c5 last 'x');
```

L'expression :

```
tab['hello]
```

a comme valeur :

```
3.4
```

et :

```
tab['ABCD']
```

vaut :

```
0x3ffa27342c5 // qui s'écrira en sortie : 4396477006533
```

Notons que la directive « `data` » peut aussi être remplacé par le caractère « ``` » : ``(1 2 3)` est strictement équivalent à `data(1 2 3)`.

7.2.5 Tables des symboles

Lors de l'exécution, différentes tables de symboles entrent en jeu :

1. la table des variables globales, qui rassemble les variables déclarées par le mot clef « `var` » et les fonctions définies.
2. la table des variables « semi-locales », qui contient les variables définies par le mot-clef « `loc` ».
3. la table des pointeurs, qui contient les variables définies par le mot-clef « `ptr` ». Ces dernières variables sont en fait des variables « indirectes », qui contiennent une adresse mémoire, et c'est cette adresse mémoire qui est en fait consultée ou vérifiée.
4. la table des variables locales à une fonction, qui contient les valeurs des paramètres et des variables déclarées par « *local* » dans cette fonction.

Les trois premières sont des « blocs » classiques, de types respectifs '`vars`', '`locs`' et '`ptrs`'. La dernière est virtuelle, étant représentée temporairement par une zone de la pile d'exécution.

7.2.6 Processus

mSL permet de définir des *processus légers* (« *light weight processes* »), qui peuvent s'exécuter en parallèle, et sans interférer avec la gestion de l'audio. Il peut y avoir des centaines de processus actifs simultanément. Tous les processus partagent le même espace d'adressage global, qui est celui du plug-in, mais peuvent avoir des espaces de variables privés séparés. A un instant donné, un seul processus est actif. La commutation entre différents processus s'effectue automatiquement (le mécanisme est préemptif, ce qui assure qu'aucun processus ne va bloquer la machine), ou par une requête explicite du processus, ou d'un autre processus.

La création de nouveaux processus peut être réalisée par le Game Master, ou explicitement, par un appel d'une fonction spécialisée.

A l'heure actuelle, les fonctions suivantes sont disponibles :

wait(sec) permet de placer le processus courant en attente pour une durée de « sec » secondes.

yield() permet à un processus de « céder la main » au processus suivant de la liste d'attente. Si le processus courant est unique, il est donc réactivé immédiatement.

exit(r) arrête définitivement le processus. La valeur du paramètre est fournie comme « exit value », et est récupérable par un autre processus.

Chaque processus venant d'être activé est assuré de disposer d'une durée minimale d'exécution donnée (plus de 4000 instructions), ce qui permet, après un *yield()* ou un

`wait()`, d'être assuré, durant cette période, d'un accès exclusif à la mémoire et aux variables manipulées, même si elles sont partagées avec d'autres threads.

Un *thread* est créé par le lancement, au moyen de la primitive `thread()`, d'une fonction sans paramètres ni variable locale. Un *thread* créé lors de l'exécution d'un script partage avec les autres *threads* créés lors de l'exécution du même *thread* toutes les variables déclarées par « `var` », mais les variables déclarées par « `loc` » sont locales à chacun des *threads*.

7.3 *mSL*, en résumé

1. *mSL* est très similaire à *JS*.
2. Les noms des variables ne sont pas limités en nombre de caractères (mais voir à ce sujet le § 7.1.1.1 page 150), sont sensibles à la casse, et les variables *doivent* être déclarées avec l'un des mots-clefs `var` ou `loc`.
3. Les opérateurs standard : `+`, `-`, `*`, `/`, `%`, `<`, `<=`, `=`, `!=`, `>=`, `>`, `==`, `!==`, `&`, `|`, `&&`, `||`, `!`, `^`, `~`, `?:`, `=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=` ont la même sémantique.
4. Les fonctions standard : `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `max`, `min`, `atan2`, `pow`, `sqr`, `sqrt`, `exp`, `log`, `abs`, `sign`, `log10`, `floor`, `ceil`, `invsqrt`, `rand`, `time`, `strlen` sont identiques dans les deux langages.
5. Certaines fonctions standard sont définies avec un nom ou une syntaxe différente : `ptime()` pour `time_precise()`, `srate()`, `num_ch()`, `tempo()`, `memtop()` pour `__memtop()`.
6. Nouvelles fonctions : `int(x)` (conversion de `x` en un entier), `irand(x)` (entier au hasard dans `[0 x-1]`), `isint(x)` (teste si `x` est un entier), `isfun(x)` (teste si `x` est une fonction), `return(v)` (sortie d'une fonction), `strcode(string)` (transformation d'une chaîne de caractères en un symbole), `sum(x1, x2, ... xn)` (somme de ses paramètres), `compile(string)` (compilation d'une chaîne de caractères sous la forme d'une fonction niladique), `error(code)` (signalement d'une erreur, et arrêt de l'exécution du processus), `malloc(size [, type])` (allocation de mémoire dynamique), `isblock(x)` (teste si `x` est un bloc de mémoire alloué par `malloc`), `mfree(mref)` (libération d'un bloc alloué), `stralloc()` (allocation dynamique d'une chaîne de caractères), `strfree(str)` (libération d'une chaîne allouée), `get(t, x)` (accès à la propriété `x` d'une table `t`), `set(t, x, v)` (modification de la valeur `v` de la propriété `x` d'une table `t`).
7. Les fonctions définies acceptent les récursions simples et croisées, et l'exécution optimise les récursions terminales. Les fonctions sont des « citoyens de première classe », c'est-à-dire qu'elles peuvent être assignées à une variable,

passées comme paramètres à une fonction, et fournies comme résultat d'une fonction.

Chapitre 8

Integration de mSL au sein du Game Master

Ce chapitre décrit plus spécifiquement l'intégration du langage *mSL* dans le plugin, et de quelle manière celui-ci est utilisé pour étendre les possibilités du logiciel.

8.1 Fonctions spécifiques

8.1.1 La procédure « **action** »

L'accès aux opérations du *Game Master* s'effectue au travers d'un certain nombre de fonctions. L'une d'entre elles permet d'exécuter les *actions* décrites au chapitre 12 page 189 depuis un script *mSL* ;

action(bloc) lorsque le paramètre est un bloc de données, ces données sont interprétées comme une séquences d'actions (c.f. chapitre 12 page 189). Il n'est pas nécessaire de préciser le nombre d'éléments constituant cette liste, information qui est fournie par le bloc lui-même.

action(num, p1, p2, ...) lorsque les paramètres sont des valeurs numériques, ils représentent le code d'une action (« num »), suivi des paramètres de l'action.

8.1.2 La procédure « **table** »

Cette procédure est en fait un analyseur syntaxique, dont le rôle est de prendre une entrée (un bloc mémoire défini en *mSL*), et de construire à partir des valeurs rencontrées certaines structures utilisées par le logiciel. Son appel est le suivant :

```
table(bloc) ;
```

Le paramètre est un bloc, qui est typiquement construit par un « data ».

La première ligne est ainsi « table (data (» et la dernière «)) ; ». Ex :

```
table(data(
    Table Sensors Clear
    MMode 0
        Enter 324 3 323 7
        DefSeq 1 DoKmd 324 9 350 7100
        DefSeq 2 DoKmd 335 0.5 349 7100
        DefSeq 3 DoKmd 301 334 60 0 302
    MMode 1
        DefSeq 1 DoKmd 350, 3001
    End
)) ;
```

L'analyse est basée sur un automate déterministe à pile et à états fini, dans la mesure où toutes les structures de tables utilisées sont simples et non récursives.

Un préambule est constitué par une séquence :

```
Table « nom_de_table »
```

Il est suivi d'une description du contenu, description dont la syntaxe, simple, est décrite pour chaque table au chapitre 13 page 199. Il peut être judicieux d'ajouter « Clear » après le nom de la table si l'on veut s'assurer que celle-ci est vierge de tout contenu, faute de quoi les nouvelles informations sont ajoutées au contenu actuel de la table.

A noter que les commentaires sont du type « C », c'est-à-dire « // » pour un commentaire de fin de ligne, et « /* » et « */ » encadrant un commentaire de plusieurs lignes.

Une description se termine en général par une indication de fin :

```
End
```

La procédure admet la description consécutive de plusieurs tables, chacune encadrée par « Table » ... « End ». Le format d'entrée des données est souple : les données peuvent être séparées par une virgule (optionnelle) et/ou des espaces ou des passages à la ligne, et des commentaires peuvent être placés n'importe où.

Les tables prises en compte sont les suivantes :

Banks	Description des groupes, banques et partiels.
Clips	Description des ajustements des clips.
ClipSets	Description des ensembles de clips.
HPConf	Configurations des haut-parleurs de l'installation.
Macros	Définition de macro commandes.

PlModes Description des modes de jeux.

Sensors Réglages des commandes pour les sensors.

SpModes Descriptions des modes d'espace.

Les syntaxes utilisées pour les descriptions de tables sont similaires à celles qui étaient utilisées dans les « anciens » fichiers texte, à la différence des commentaires (« // » remplaçant le « ; »). Notons aussi que « End » ne sert plus qu'à indiquer la fin d'une description de table. Il n'est donc plus utilisé dans la description des modes de jeu, des modes d'espace et des banques.

De nombreux exemples d'utilisation de l'opération « table » sont donnés au chapitre 13 page 199.

8.1.2.1 Macro commandes

La table « Macros » a pour finalité de conserver les macro-commandes définies par l'utilisateur. Une macro commande est introduite par le mot-clef « DefM » suivi du nom de la macro, suivi enfin d'une séquence de valeurs, qui sont des actions associées à leurs paramètres. Les actions peuvent être représentées soit par leur valeur numérique, soit par leur identifiant, tel qu'il est décrit au chapitre 12 page 189. Ainsi, les deux séquences ci-dessous sont-elles identiques :

```
326, 200034, 328, -4, 350, 6901
TempBnk, 200034, TempVol, -4, PlayUnmd, 6901
```

Pour définir une macro, de nom « Jeu », représentant l'enchaînement de ces actions, on écrira :

```
DefM Jeu 326, 200034, 328, -4, 350, 6901
```

ou encore

```
DefM Jeu TempBnk 200034, TempVol -4, PlayUnmd 6901
```

Une macro définition se termine soit par le mot « End », soit par « DefM » débutant une nouvelle définition. Signalons aussi que, comme dans tout bloc « data », les « virgules » sont autorisées, mais facultatives dans cette syntaxe.

Cette macro peut dès lors, dans la table « Sensors », s'utiliser pour représenter le comportement d'un senseur, sous une forme telle que :

```
DefSeq, 7, DoKmd, Jeu
```

Chaque fois que le capteur « 7 » sera activé, le logiciel jouera le clip « 6901 », affecté d'un volume de « -4dB », avec les réglages de jeu de la banque « 34 » du groupe « 200 ».

Notons qu'un nom de macro peut apparaître à chaque endroit où une commande est attendue, mais aussi qu'une telle macro est un simple remplacement textuel, avant l'interprétation effective de la commande. Si l'on définit notre macro sous la forme :

```
DefM Jouer TempBnk 200034 TempVol -4 PlayUnmd
```

c'est à dire en ne précisant pas le dernier élément, qui est le numéro du clip à jouer, celui-ci devra être indiqué dans la commande, à la suite du nom de la macro. « Jouer » devient ainsi une commande signifiant « jouer un clip à -4db, avec les réglages de la banque 34 du groupe 200 », ce qui va permettre des écritures telles que :

```
DefSeq 1 DoKmd Jouer 7210
DefSeq 2 DoKmd Jouer 7217
DefSeq 7 DoKmd Jouer 6901
```

Les macro commandes ne sont pour l'instant acceptées que dans la description de la table « Sensors », et dans la fonction « action ».

8.1.3 Opérations « get/set »

Deux opérations, `get()` et `set()`, permettent l'accès à nombre de tableaux et de variables utilisées dans le « Game Master ».

get(x) Accès à l'objet de nom « x ». Les paramètres possibles sont les identificateurs suivants :

vars tableau des variables globales du thread - ex : « `get('vars')` ». Le résultat est l'adresse de la table correspondante.

locs tableau des variables semi-locales du thread.

get(x, id) Accès à l'objet de nom « id » de l'ensemble « x ». Les valeurs acceptées pour « x » sont les suivantes :

pnum numéro d'un paramètre spécifique. « id » est le nom du paramètre. Ex : « `get('pnum', 'GenNbr')` » fournit le numéro interne du paramètre correspondant au nombre de générateurs actifs.

pval valeur d'un paramètre spécifique. « id » est le nom du paramètre. Ex : « `get('pval', 'GenNbr')` ». Si le numéro interne du paramètre est déjà connu (obtenu, par exemple, par « `get('pnum', 'GenNbr')` »), cette valeur peut être utilisée comme identification. Ex : « `get('pval', 20)` »

gm accès à une variable spécifique du Game Master. Plus d'une centaine de variables « pertinentes » du logiciel sont ainsi accessibles par des scripts. Ex : « `get('gm', 'mSL_errX')` » fournit le numéro de la dernière erreur survenue en exécution.

set(x, id, val) Affectation de la valeur « val » à l'objet de nom « id » de l'ensemble « x ». Les valeurs acceptées de « x » sont les suivantes :

pval valeur d'un paramètre spécifique. « id » est le nom ou le numéro du paramètre. Ex : « set ('pval, 'GenNbr, 10) » affecte la valeur 10 au nombre de générateurs actifs.

gm modification d'une variable spécifique du Game Master. Plus d'une cinquantaine de variables « pertinentes » du logiciel sont ainsi accessibles en écriture par des scripts.

8.1.4 Opération « call »

La fonction « call » permet l'accès à certaines fonctions du Game Master. La syntaxe est :

call('name, paramètres...) Le nom de la fonction est un symbole, les paramètres de la fonction suivent ce nom. Voici quelques unes des fonctions implantées :

'strclean, str, flags suppressions dans la chaîne passée comme second paramètre des espaces de début (si « flags & 1 »), de fin (si « flags & 2 »), ou d'une chaîne de fin constituée d'un « . » suivi de zéros (si « flags & 4 »).

'SensorSize [, newSize] indique la taille du *pad* des sensors, ou modifie celle-ci si un paramètre est fourni. Cette valeur est 2 ou 3 (pads 2 × 2 ou 3 × 3).

'clear, block [, valeur] (ré)initialise le bloc passé comme premier paramètre à 0, ou à « valeur » si celle-ci est fournie.

8.1.5 Extensions

Il est relativement simple d'ajouter de nouvelles fonctions primitives au langage mSL. A titre d'exemple, le fichier source « mSL_Extend.jsfx-inc » définit trois nouvelles fonctions, de noms « prod », « GM_Log » et « GM_Addr ».

prod accepte un nombre arbitraire de paramètres et calcule leur produit. On peut s'en inspirer aisément pour réaliser une fonction calculant la somme, le maximum, le minimum, etc., d'une liste de nombres.

GM_Log accepte des chaînes de caractères, et les ajoute au log du Game Master.

GM_Addr accepte un mot clef, et fournit une adresse en mémoire relative à ce mot-clef.

De manière générale, les *byte-codes* internes de numéros 132 à 255 sont disponibles pour de tels ajouts. Les trois fonctions décrites ci-dessus correspondent aux numéros 255, 254 et 253. Pour définir une nouvelle action primitive, il faut choisir un nom (par exemple «newfun»), un numéro disponible (par exemple, 200), et insérer dans le fichier «mSL_Extend.jsfx-inc» le nouveau code source de la fonction «mSL_200», actuellement définie dans le fichier «mSL_Clib.jsfx-inc» sous la forme : «function mSL_200(c,v) (c);».

On écrira, par exemple :

```
function mSL_200(c,v)
(
...
);
```

et l'on ajoutera dans la fonction «mSL_define_extensions» la ligne :

```
mSL_define_opcode("newfun", 200);
```

Lors d'un appel de la nouvelle fonction tel que «newfun(2, 3, 5, 8)», la fonction «mSL_200» va recevoir dans son premier argument, «c», le nombre de paramètres de l'appel de «newfun», 4, et dans son second argument, «v», l'adresse en mémoire d'un tableau contenant les 4 paramètres de l'appel.

8.2 Lancement d'un script

A l'heure actuelle, le lancement d'un script s'effectue depuis le module «*Script Manager*». Effectuer un clic droit sur la case «*Load*», ce qui affiche un menu proposant des scripts susceptibles d'être chargés. Ceux-ci doivent être déclarés d'une manière ou d'une autre : c.f. § 1.11 page 40.

Les fichiers Conf03.mSL à Conf08.mSL contiennent des exemples de définitions des différentes tables du logiciel.

Une autre manière d'ouvrir un script *mSL* est de le sélectionner (depuis le bureau, ou une fenêtre du finder) et de le déposer dans la barre de tabulation de la fenêtre du Game Master, qui l'exécute alors immédiatement.

Chapitre 9

Processus

L'une des forces du GameMaster est son langage de script intégré, « `ee12` », et la possibilité de l'utiliser pour gérer des processus légers au sein du plug-in.

9.1 Introduction

Il n'est pas inutile de décrire ici les interactions entre REAPER, le GameMaster, les scripts *mSL* et les processus que l'on vient d'évoquer.

9.1.1 REAPER

Le logiciel REAPER, en plus de son travail interne, de la gestion de son interface utilisateur et des scripts ReaScripts, a pour tâche d'exécuter régulièrement les différents plug-ins utilisés dans la session, que ce soient des VST, VST3, etc., ou des JSFX.

REAPER gère l'audio par blocs d'échantillons, d'une taille fixe mais modifiable, comprise entre 32 et 4096 par canal. A chaque seconde, REAPER doit assurer, pour chaque piste, la production d'une quantité d'échantillons égale au produit de la fréquence d'échantillonnage (44.1kHz, 48kHz, 96kHz, etc.) par le nombre de canaux (2, 6, 16, jusqu'à 128 à l'heure actuelle).

Prenons un exemple quasi concret : le projet utilise 20 pistes, et travaille à 48kHz en 5.1, soit 6 canaux. Chaque piste comporte deux plug-ins, et la taille du bloc est de 1024 échantillons. A chaque seconde, pour chaque piste, REAPER doit fournir 48000×6 échantillons. Chaque plug-in disposerait donc, au mieux, de $1s / (20 \text{ pistes} \times 2 \text{ plug-ins})$ pour traiter ces 288000 échantillons, soit environ 0.5 millisecondes par bloc de 1024×6 échantillons. Ajoutons que nombre d'autres processus s'exécutent sur la même machine, que REAPER utilise lui-même de la CPU, si bien qu'il est douteux qu'un plug-in dispose de beaucoup plus que 0.1 millisecondes pour effectuer

son travail, ce qui est bien peu. Précisons, dans le cas d'un JSFX, et toujours pour le même exemple, que ceci implique l'exécution de la section « @block » et de 1024 fois la section « @sample » qui elle-même doit calculer six échantillons. La programmation de tous ces codes doit donc s'assurer que chaque exécution reste en deçà des durées autorisées, afin d'éviter des « *drop out* » du son en sortie.

Dans le même temps, REAPER gère des processus de plus faible priorité que les processus audio, qui sont les processus de gestion de l'interface graphique. REAPER maintient sa propre interface, et donne la main à chaque plug-in pour que celui-ci s'occupe de son interface, ceci une trentaine de fois par seconde. Dans le cas de JSFX, il s'agit du code situé dans la section « @gfx ». La encore, pour maintenir cette cadence d'une trentaine d'exécution par seconde par chaque plug-in, ces processus se doivent d'être programmés de la façon la plus efficace possible. Les processus graphiques sont donc moins prioritaires que les processus audio, mais peuvent occasionnellement se permettre de dépasser largement le temps qui leur est en principe accordé, REAPER diminuant alors leur fréquence d'exécution. L'interface graphique peut ainsi se figer durant quelques secondes sans provoquer de « drop-out » audio.

9.1.2 Le GameMaster

Revenons au GameMaster. Considérons un projet comportant le seul GM et ses players, tournant sur une machine n'utilisant pas d'autres applications coûteuses en temps. On peut imaginer que REAPER consomme au total la moitié de la puissance disponible (par exemple, 4 cœurs de processeurs sur 8), et que le GM va avoir droit à un cœur pour lui tout seul – vue probablement optimiste.

Le GM (toujours en 48kHz, blocs de 1024) dispose donc, au mieux, de 1024/48000, soit environ 21ms pour assurer son travail. Notons ici que le GM, n'ayant pas de section « @sample », n'exécute que la section « @block », et que la durée de celle-ci est pratiquement indépendante de la taille du bloc. Augmenter la taille de celui-ci (2048 ou 4096) diminue donc la consommation CPU totale, puisque le GM est exécuté 2 ou 4 fois moins souvent, mais également la réactivité, et réciproquement.

Le GM doit donc accomplir en un temps limité tout un ensemble de tâches diverses. Une partie de celles-ci est consacrée aux aspects graphiques, et est exécutée dans la section « @gfx » du plug-in. Les autres tâches vont s'exécuter dans une apparente simultanéité, dans la section « @block » du plug-in. Décrivons en quelques unes :

- Gérer les événements du « playback », lorsque le GM est en train de rejouer une séquence enregistrée.
- Converser avec le ReaScript écrit en « eel » qui permet les interactions poussées avec REAPER.
- Gérer les interactions avec les différents players, qui sont des plug-ins situés sur des pistes audio indépendantes.

- Gérer tous les calculs résultant des interactions de l'utilisateur avec l'interface graphique.
- Gérer les événements MIDI reçus par des plug-ins spécialisés et transmis au GM.
- Gérer les différents aspects du mode « jeu automatique ».
- Maintenir diverses statistiques concernant l'utilisation des ressources.
- Gérer le chargement et l'exécution des différents scripts de configuration.

Pour prendre en compte ces différentes tâches, le GM analyse les données appropriées, et réalise les opérations nécessaires au travers « d'Actions » ou de séquences d'actions, à l'instar de ce que pratique REAPER. Certaines de ces actions vont être immédiates, mais d'autres vont être retardées dans le temps, ou conditionnées à des événements attendus. Dans tous les cas, sachant qu'à chaque exécution de la section « @block » le GM n'a droit qu'à quelques millisecondes de temps CPU, il est clair que ces actions sont obligatoirement de durées très courtes, ou encore vont devoir être segmentées en parties de courtes durées, s'exécutant au cours de « @blocks » successifs.

A cette fin, les différents travaux vont être organisés sous la forme de ce que l'on nomme « processus légers ».

9.1.3 Processus légers

Ces processus vont être créés pour différentes raisons. Prenons l'exemple du jeu d'un clip. La décision de jouer un clip peut résulter de l'exécution du mode « jeu automatique » du GM. L'utilisateur peut aussi cliquer dans une case de l'un des modules, tels que le « Studio Play Pad » ou le « Clip Selection ». Un clip peut être joué en réponse à une interaction avec un périphérique MIDI, voir être lancé par une action spécifique automatique, ou encore un script *mSL*. Ces événements peuvent se produire simultanément, mais le mécanisme de jeu d'un clip, qui est relativement complexe, ne peut en traiter qu'un à la fois. Dans tous les cas, le GM va créer, en quelques dizaines d'instructions, un processus, qui va être inséré dans la liste des processus prêts à être exécutés. De même, une tâche telle que « modifier le volume sonore dans 30 secondes » va se traduire par la création d'un processus, qui va être placé dans une file d'attente, et qui, au bon moment, sera à son tour inséré dans la liste des processus prêts à être exécutés.

Ces processus sont dits « légers », car ils sont représentés par des structures informatiques de petite taille (une vingtaine de mots, décrivant la consigne du processus, c'est-à-dire le type de travail à réaliser, et quelques paramètres, structure informatique dite « SCB », Scheduler Control Block), et que le passage d'un processus à un autre est très rapide.

A un instant donné, il peut exister plusieurs processus, voire plusieurs dizaines ou centaines de processus. Comment sont-ils gérés ? Chaque processus est donc re-

présenté par un SCB, qui peut être :

- actif, le processus qu’il décrit étant en cours d’exécution.
- dans la liste des processus actifs, c’est-à-dire prêts à être exécutés dès que possible.
- dans la liste des processus en attente sur un timer.
- dans la liste des processus en attente d’un événement particulier.
- dans la liste des processus temporairement suspendus, car une ressource du système essentielle est utilisée. C’est le cas des processus « *mSL* » (ceux qui résultent de la compilation d’un script *mSL*) lorsqu’une compilation ou un garbage collector est en cours.
- dans la liste des processus verrouillés (essentiellement du fait d’une intervention de l’utilisateur, ou en tant que composante de l’algorithme d’un autre processus).

Il y a, au maximum, un seul processus actif à un moment donné. Les processus du système sont écrits pour s’exécuter en un seul tenant, ou encore segmentés en petites sections qui s’enchaînent, chaque partie étant d’une durée très courte, tout en étant assurée de l’accès exclusif aux ressources qu’elle nécessite. A la fin de cette période d’activité, le processus décide s’il a terminé sa tâche, ou s’il doit se réinscrire en queue de la liste des processus actifs, ou de l’une des listes de processus en attente.

Notons que, à chaque exécution de la section « @block », au moins un processus actif est exécuté (plus, si le temps le permet). Il n’existe pas de priorité entre les processus, et l’ensemble des processus actifs est géré en « *Round Robin* », c’est-à-dire que le premier de la liste est choisi pour exécution. Lorsque sa période d’activité s’est écoulée, s’il n’a pas terminé son travail, il est réinséré en queue de liste des processus actifs. Cette approche garantit donc que tout processus progresse à tout instant, en étant éventuellement ralenti, mais jamais bloqué par d’autres processus.

9.1.4 Le Scheduler au sein du GameMaster

Résumons ici les articulations entre le Scheduler et les diverses parties du GameMaster.

La partie « @block » du GM est exécutée à intervalles réguliers : selon la taille du bloc d’échantillons de REAPER, à 48kHz c’est entre 12 fois par seconde pour des blocks de taille 4096 à 1500 fois par seconde pour des blocks de taille 32. Cette dernière valeur est déconseillée, sauf sur une machine ultra rapide à venir. Il est conseillé de ne pas descendre en-dessous de 512.

Au cours de l’exécution de cette partie « @block », le GM gère l’ensemble de ses tâches, qu’il accomplit immédiatement pour une partie d’entre elles, ou qu’il réifie sous la forme d’un processus, qu’il introduit dans l’une des diverses listes de processus. Parmi ces tâches, l’une consiste à retirer des listes des processus en attente (d’un timer ou d’un événement) ceux dont le timer est arrivé à son terme, ou dont

l'événement attendu s'est produit, pour l'insérer dans la liste des processus actifs. Cette description montre que la réactivité du GM (délai entre deux « @block ») ne peut guère être meilleure que 10 à 100ms.

A la fin de cette section « @block », le GM évalue le temps CPU dont il peut encore disposer pour ce block, et va exécuter au moins l'un des processus actifs, voire plus en fonction du temps disponible.

Certains de ces processus correspondent à des scripts *mSL* « compilés », c'est-à-dire traduits en une séquence d'instructions élémentaires d'une machine virtuelle. Là encore, toutes les instructions de cette machine virtuelle sont rapides (calcul d'un sinus) ou très rapides (telle une addition). Certaines primitives peuvent être relativement lentes dans certains cas. Elles vont alors créer un processus nouveau pour réaliser ces actions, qui pourra se dérouler en parallèle avec le processus *mSL* (cas du jeu d'un clip), ou mettre celui-ci en attente (cas de certains processus gérant des événements). Dans tous les cas, le GM autorise le processus *mSL* à n'exécuter qu'un nombre limité d'instructions de cette machine virtuelle (au minimum, 4096, au maximum quelques centaines de milliers), avant d'interrompre celui-ci, et de le replacer en queue de liste des processus actifs.

Le Scheduler se comporte donc de manière coopérative pour les processus du système (qui sont écrits pour être très brefs, sont largement testés, et ne sont jamais interrompus, ce qui permet une programmation plus concise et plus efficace), et de manière préemptive pour les processus *mSL* (on parle ici de « time slicing »). Notons que ceux-ci peuvent aussi faire preuve d'un comportement coopératif, la fonction `yield()` ou encore les primitives d'attente sur timer ou sur événement impliquant qu'ils passent la main à un autre processus.

9.1.5 Opérations sur threads : « thread »

thread(...) fournit une série d'opérations sur des objets de type *thread*. Les opérations sont désignées par un mot-clef (introduit par une back-quote). Ce sont les suivantes :

new création d'un nouveau thread. Le second paramètre doit être le nom d'une fonction *mSL*, sans paramètres ni variables locales. Le thread est placé dans la file des threads suspendus, et est prêt à être lancé. Le résultat est l'identification du thread.

run lancement d'un thread. Le second paramètre doit être l'identification d'un thread, telle que créé par une expression « `thread('new, fun)` ».

status statut d'un thread. Le second paramètre doit être l'identification d'un thread. Le résultat est une combinaison des flags suivants :

- 1** le thread correspond à un processus *mSL*.
- 8** le thread est actif.

idt identification du thread courant, qui est son numéro de processus. Il n'y a pas d'autre paramètre. Cette identification est un nombre entier, unique au processus.

name nom du thread courant. Des threads différents peuvent porter des noms identiques, mais auront toujours des « *idt* » différents.

find recherche d'un thread par son nom et/ou son numéro de processus. Les deux valeurs peuvent être spécifiées (*idt* ou *name*), l'une pouvant être égale à 0. Une troisième valeur, optionnelle, peut être indiquée. Elle permet de définir l'information demandée : 0 pour le bloc définissant le process lui-même, 1 pour le status du process, 2 pour son *idt*, 3 pour son *name*, 4 pour son activité actuelle (0 : il est inactif, 1 : il est actif, 2 il est en attente sur timer, 3 il est en attente d'un événement, 4 il est suspendu, 5 il est verrouillé, 6 il est dans un autre état).

count le paramètre est un nom de processus ; le résultat est le nombre de processus portant ce nom.

Le Game Master est actuellement configuré pour gérer jusqu'à 4096 processus, dont une vingtaine est utilisée pour ses besoins propres, et le reste est disponible pour les scripts *mSL*.

9.1.6 Opérations sur événements : « **event** »

event (...) fournit une série d'opérations sur des objets relatifs aux événements. La syntaxe est :

event('name', paramètres...) Le nom de la fonction est un symbole, les paramètres de la fonction suivent ce nom. Voici quelques unes des fonctions implantées :

check permet de savoir si un événement a été notifié au thread courant.

wait ou getnext récupère l'événement suivant notifié au thread. Le résultat peut aussi être 0 si un time-out a été indiqué, et qu'aucun événement n'a été notifié durant ce laps de temps.

ready le paramètre est un « *idt* » d'un thread. Le résultat est un booléen (0 ou 1) indiquant si ce thread est réceptif, c'est-à-dire en capacité de recevoir un message.

find recherche d'un thread par son nom et/ou son numéro de processus. Les deux valeurs peuvent être spécifiées (*idt* ou *name*), l'une pouvant être égale à 0. Une troisième valeur, optionnelle, peut être indiquée. Elle permet de définir l'information demandée : 0 pour le bloc définissant le process lui-même, 1 pour le status du process, 2 pour son *idt*, 3 pour son

name, 4 pour son activité actuelle (0 : il est inactif, 1 : il est actif, 2 il est en attente sur timer, 3 il est en attente d'un événement, 4 il est suspendu, 5 il est verrouillé, 6 il est dans un autre état).

notify construction et envoi d'un message. Cette fonction admet 13 paramètres, avec la même signification que la commande « *build* ».

build construction d'un nouveau message. La fonction admet 13 paramètres, qui sont, dans l'ordre :

1. Le bloc destiné à recevoir le message. Il doit être de taille au minimum égale à 16, mais on peut aussi transmettre la valeur 0 pour demander à ce que l'opération fournisse elle-même le bloc destiné à recevoir le message.
2. un « *idt* » de l'expéditeur. On peut aussi passer la valeur 0, pour que l'opération infère elle-même l'*idt* de l'expéditeur.
3. le « *name* » de l'expéditeur. Là aussi, on peut passer la valeur 0, pour que l'opération infère elle-même cette information.
4. l'« *idt* » du destinataire.
5. le « *name* » du destinataire. L'opération tentera d'inférer les valeurs exactes de ces deux paramètres en fonction des informations fournies.
6. le nom de l'événement notifié au destinataire. C'est typiquement un symbole.
7. Une durée à attendre (exprimée en secondes) avant l'envoi du message.
8. Une durée à attendre (exprimée en secondes), après l'envoi du message, et avant destruction automatique de celui-ci s'il n'a pas été récupéré par le destinataire. Cette durée est approximative, le message étant conservé au moins aussi longtemps qu'indiqué.
9. Indicateurs.
10. Code.
11. Sequence.
12. Par1.
13. Par2.

Les cinq dernières valeurs ne sont pas utilisées par le système, et peuvent être employées librement au sein du message. Le résultat est le bloc créé ou initialisé par l'opération.

send envoi d'un message tel qu'il a pu être construit par la commande « *build* ».

count Compte le nombre de message en attente pour ce processus. Si un paramètre est précisé (sous la forme d'un symbole), compte le nombre de messages en attente portant ce nom.

recycle Avec un paramètre, qui doit être un message devenu inutilisé, cette opération « *recycle* » le message, qui est intégré à une liste de messages « *disponibles* » pour les opérations « *build* » ou « *notify* ». Sans paramètre, cette opération renvoie la taille de la liste des messages disponibles.

clearall supprime tous les messages en attente destinés au thread courant.

Chapitre 10

Jeu des clips depuis mSL

Ce chapitre aborde exclusivement les interfaces disponibles pour contrôler le jeu des clips depuis le langage mSL.

10.1 Scénario

10.2 Variables et structures de données

10.3 Opérations disponibles

10.3.1 Jeu de clips : « **play** »

La fonction « **play** () » permet de jouer un clip, en transmettant différents paramètres. Les paramètres sont des mots-clefs, qui précèdent éventuellement les valeurs correspondantes. Les mots-clefs reconnus sont les suivants :

part introduit un numéro de partiel, dont les caractéristiques vont être utilisées pour le jeu ; par défaut, le partiel 0 est sélectionné.

clip introduit un numéro de clip (0 à 9999) à jouer.

vol introduit un volume, exprimé en décibels, dans l'intervalle [-48 12].

plmode introduit un mode de jeu

spmode introduit un mode d'espace

hpent introduit un numéro de configuration de HP.

hpset introduit un tableau représentant une configuration de haut-parleurs, sous la forme décrite dans la table `HPConf` (un tableau, contenant dans l'ordre : un nombre « *n* » de HP, une liste de « *n* » HP, un zéro).

rate introduit une vitesse de lecture (dans [-10 10]).

dur introduit la durée à jouer, exprimée en secondes, dans l'intervalle [0.001 3600].

grfades durée de fade-in/fade-out pour les grains en mode loop, exprimée en millisecondes.

PMod modificateurs du mode de jeu.

unit mot-clef utilisé sans paramètre. Il demande à ce que l'adresse de l'unité utilisée pour jouer le clip soit fournie en résultat de la fonction.

Il est possible également de passer en paramètre un tableau unique. Voici deux exemples (identiques dans leur résultat) d'utilisation :

```
play(`clip, 3320, `rate, 0.65, `vol, -1.5, `plmode, 11);  
var k = data(clip 3320 rate 0.65 vol -1.5 plmode 11);  
play(k);
```

10.3.2 Gestion des players : « player »

La fonction « `player()` » permet d'envoyer des commandes à un player spécifique.

Le premier paramètre est l'adresse mémoire du bloc décrivant le player. Une telle valeur est rendue par la fonction « `play()` » dans laquelle on a précisé le paramètre « `unit` ».

Ce paramètre peut être suivi de l'un des mots-clef « `Imm` » (valeur par défaut), action immédiate ou encore « `Dly` », action temporisée. Les actions immédiates sont exécutées le plus rapidement possible au sein du processus « audio » du player, les actions temporisées sont lancées depuis un processus moins prioritaire, typiquement quelques dizaines de millisecondes plus tard. Si une demande d'action suit immédiatement le lancement d'un clip par « `play` », il vaut mieux utiliser une action temporisée. Si l'on est certain que le lancement du clip s'est effectué correctement, une action en mode immédiat est préférable.

Après la référence à l'unité (et éventuellement l'un des mots-clef décrivant la temporalité de l'action) vient un numéro d'action suivi de paramètres éventuels.

Numéro	Description	Paramètres
1	Smooth factors	Deux valeurs (dans [0 ... 1]) indiquant les valeurs de croissance et de décroissance pour le filtrage de certains paramètres.
2	Smooth period	Mise à jour des valeurs modifiées.
3	Global reset	Réinitialisation à une valeur standard des caractéristiques modifiables des players.
4	Volumes 1/2	Spécification des volumes de sortie sur les configurations de HP 1 et 2. Deux valeurs dans [0 ... 1].
5	Hold release	Lancement effectif du jeu d'un player verrouillé par un « hold »
6	Set space algo	Si aucun paramètre : reconstruction de l'algorithme courant de mise en espace. Si un paramètre est spécifié : choix d'un nouvel algorithme de mise en espace, et construction de celui-ci.
7	Repeat on	Passage en mode « repeat » pour le clip courant
8	Repeat off	Suppression du mode « repeat » pour le clip courant
9	Set gensize	Modification de la taille des tuples générés par le lecteur
10	Set par source	Modification du numéro de paramètre déterminant la source du paramètre auxiliaire des mouvements. La valeur peut être 0 (aucun paramètre) ou l'un des paramètres génériques, « X.Par 0 » à « X.Par 3 ».
11	Disconnect pan source	Déconnection du réglage de panoramique du paramètre « Dyn.X-Pan », et utilisation de l'argument pour valeur de ce paramètre.
12	Set HPSet 1	Remplacement du set de canaux HP 1 par la liste de canaux passés comme paramètres.
13	Set HPSet 2	Remplacement du set de canaux HP 2 par la liste de canaux passés comme paramètres.
14	Set update algo	Modification de l'algorithme auxiliaire, servant à la mise à jour des volumes de canaux de sortie.
15		

Troisième partie

Configuration et paramétrisation

Chapitre 11

Liste des paramètres et indicateurs

Ce chapitre décrit la liste des différents paramètres utilisables pour modifier le comportement du logiciel, et des indicateurs, consultables, mais non modifiables, décrivant des aspects du fonctionnement du logiciel. Tous les objets ici décrits sont affichables dans les blocs de contrôle de type « *sliders* », mais aussi sont accessibles, et modifiables, depuis tout programme *mSL*.

11.1 Paramètres

Chaque paramètre est défini par son étiquette, affichable dans certains visualisateurs, par son nom symbolique compacté (identificateur limité à 8 caractères), et par son numéro, qui permettent de le référencer dans les scripts. Le numéro indiqué peut varier d'une version à l'autre du GM, et n'a pas forcément été mis à jour dans ce document). Il est préférable de référencer les paramètres par leur nom symbolique, indiqué également entre crochets. Enfin, l'intervalle de variation que le paramètre peut prendre est indiqué, entre accolades, sous la forme de quatre nombres : la valeur minimale, la valeur maximale, la valeur par défaut, et le pas de variation lors des réglages par un slider.

Auto Freq. [29, `AutoFreq`] « Fréquence » à laquelle de nouveaux clips peuvent être joués en mode automatique. C'est en fait une *durée* en secondes qui indique l'intervalle de temps minimal à respecter avant qu'un nouveau clip puisse être joué. Intervalle de variation {0.0, 30, 0.01, 0.01}.

Auto Mode [25, `AutoMode`] Fonctionnement (on/off, ou marche/arrêt) de l'auto-mode, c'est à dire du lancement automatique de la lecture de clips. Intervalle de variation {0, 1, 0, 1}.

Channels # [26, `ChanNbr`] Nombre de canaux à utiliser en sortie. Permet une adaptation d'une configuration définie à une autre configuration, comportant

moins de canaux, au prix de la perte des sémantiques associées aux configurations de haut-parleurs. Intervalle de variation {2, 64, 64, 1}.

Cl.Pl.End [102, ClPEnd] Choix du point d'arrêt de lecture d'un clip en mode interactif. Intervalle de variation {0, 100, 0, 0.1}.

Cl.Pl.Start [101, ClPStart] Choix du point de départ de lecture d'un clip en mode interactif. Intervalle de variation {0, 100, 0, 0.1}.

Cl.Start.Pos [81, ClStPos] Choix du point de départ de lecture d'un clip, lorsqu'une durée de lecture plus courte que la durée du clip est choisie. Intervalle de variation {0, 100, 0, 1}. Pour une valeur de 0, la portion lue est choisie au début du clip, pour une valeur 100, la portion lue est choisie à la fin du clip, pour une valeur intermédiaire, la portion est prise entre ces bornes, avec intervention d'un aléa.

Clip.Rd.Lim [27, ClRdLim] Limitation (en secondes) de la taille d'un clip lors de sa lecture. Il existe une limitation, liée à la mémoire allouée à chaque lecteur, sur la taille maximale que l'on peut lire (de l'ordre de 4 minutes en stéréo). Intervalle de variation {1, 600, 60, 1}.

Contin.Rate [83, ContRate] Variation dynamique de la vitesse (appliquée à tous les players actifs). Intervalle de variation {0.01, 10, 1, 0.001}.

Gen min # [113, GenMinCt] Nombre minimum de générateur à utiliser en mode automatique. Intervalle de variation {0, 24, 4, 0.1}.

Gen max # [114, GenMaxCt] Nombre maximum de générateurs à utiliser en mode automatique. Intervalle de variation {0, 24, 4, 0.1}.

Generators # [20, GenNbr] Définit le nombre de générateurs simultanément actifs. Intervalle de variation {0, 64, 4, 0.01}. On notera que le nombre maximal de générateurs disponibles dans ce réglage (64) est théorique. Il est limité par le nombre de plug-ins « Players » effectivement installés sur les pistes. Ainsi, s'il n'y a que 8 players disponibles, l'intervalle effectif de variation affiché sera {2, 8}. Il est possible de demander un nombre fractionnaire de générateurs. Ainsi, régler sur 2.5 indique qu'il y aura, en moyenne, de 2 à 3 générateurs actifs, et que sur un moyen (quelques minutes) ou long terme (quelques dizaines de minutes), il y aura, en moyenne, 2.5 générateurs actifs.

G# Switch Time [109, PSwtchT] Intervalle de temps moyen, en mode automatique, séparant deux changements consécutifs du nombre de générateurs. Intervalle de variation {2, 1200, 180, 1}.

G# Variation [110, PVariaT] Mode de variation du nombre de générateur en mode « jeu automatique ». Intervalle de variation {0, 100, 10, 1}.

Gr.Density [100, GrDensty] Nombre de grains simultanément actifs dans un looper. Peut être modifié dynamiquement. Intervalle de variation {1, 24, 1, 1}.

- Gr.Fade.In** [87, GrFadIn] Durée moyenne du fade-in des grains générés en mode « loop », exprimée en ms. Intervalle de variation {1, 3000, 1000, 1}. Pour des valeurs élevées et des grains de petites tailles, les fade-in et fade-out sont réduits, de manière proportionnelle à leur valeurs respectives, pour que leur somme ne dépasse pas la durée du grain, permettant à celui-ci d'atteindre son amplitude maximale lors du jeu.
- Gr.Fade.Out** [88, GrFadOut] Durée moyenne du fade-out des grains générés en mode « loop », exprimée en ms. Même remarque que pour « Gr.Fade.In ».
- Gr.Min Size** [97, GrMnSiz] Durée minimale d'un grain, en secondes. Intervalle de variation {0.002, 30, 1, 0.001}.
- Gr.Max Size** [95, GrMxSiz] Durée maximale d'un grain, en secondes. Intervalle de variation {0.002, 30, 1, 0.001}.
- Grp Switch Time** [41, GSwTchT] Durée moyenne de passage d'un groupe au suivant, en mode auto-play. Intervalle de variation {2, 1200, 180, 1}.
- Grp Variation** [112, GVariat] Mode de variation du choix du groupe en mode « jeu automatique ». Intervalle de variation {0, 100, 10, 1}.
- HP Density** [40, HPDensty] « Densité » (en fait, pourcentage des HP indiqués dans le mode d'espace) de haut-parleurs à utiliser. S'applique à partir des prochains lancements de clips. Intervalle de variation {1, 100, 100, 1}.
- HP Used** [99, HPUsed] Nombre de HP effectivement utilisés lors de la lecture d'un clip. Ce nombre peut être dynamiquement modifié dans un displayer. Intervalle de variation {1, 64, 2, 1}.
- Loop Dur.** [31, LoopDur] Durée par défaut de jeu pour les générateurs en mode « Loop », exprimée en secondes. Intervalle de variation {0.1, 600, 30, 0.1}.
- Loop Gain** [33, LoopGain] Gain par défaut pour le jeu des générateurs en mode « Loop », exprimé en décibels. Intervalle de variation {-40, 20, 0, 0.1}.
- Loopers #** [22, LoopNbr] Nombre de générateurs simultanément actifs en mode « loop ». Intervalle de variation {0, 64, 2, 0.01}. Même remarque que pour les générateurs sur le nombre maximal et les valeurs fractionnaires de ce réglage.
- Lp.Fade.In** [36, LpFadIn] Durée, par défaut, de fade-in pour les loopers. Intervalle de variation {0.002, 30, 1, 0.001}.
- Lp.Fade.Out** [37, LpFadOut] Durée, par défaut, de fade-out pour les loopers. Intervalle de variation {0.003, 30, 1, 0.001}.
- Mass** [86, Mass] Intervalle de variation {0.002, 30, 1, 0.001}. *Non opérationnel.*
- Max players** [129, MxPlayer] Limite globale du nombre de générateurs simultanément actifs, au-delà de laquelle le déclenchement d'un nouveau générateur provoque l'arrêt du plus anciennement actif. Intervalle de variation {1, 40, 40, 1}.

- Pl.Fade.In** [34, PlFadIn] Durée, par défaut, de fade-in pour les players. Intervalle de variation {0.002, 30, 1, 0.001}.
- Pl.Fade.Out** [35, PlFadOut] Durée, par défaut, de fade-out pour les players. Intervalle de variation {0.003, 30, 1, 0.001}.
- Play Dur.** [30, PlayDur] Durée par défaut de jeu pour les générateurs en mode « Play », exprimée en secondes. Intervalle de variation {0.1, 600, 20, 0.1}.
- Play Gain** [32, PlayGain] Gain par défaut pour le jeu des générateurs en mode « Play », exprimé en décibels. Intervalle de variation {-40, 20, 0, 0.1}.
- Players #** [21, PlayNbr] Nombre de générateurs simultanément actifs en mode « play ». Intervalle de variation {0, 64, 2, 0.01}. Même remarque que pour les générateurs sur le nombre maximal et les valeurs fractionnaires de ce réglage.
- Randomness** [65, RandNess] Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rate Initial** [28, RateInit] Normalement réglée à 1, ce paramètre permet de modifier la vitesse de lecture des prochains clips lus. Cette variation s'ajoute (en fait, c'est une multiplication) avec les variations de vitesse décrites dans le mode de jeu utilisé pour le clip. Intervalle de variation {0.1, 4, 1, 0.001}.
- Read Reverse** [94, RdRev] Indique si le clip courant doit être lu à l'envers ou non. Peut être utilisé dynamiquement pour modifier le sens de lecture d'un clip actif. Intervalle de variation {0, 1, 0, 1}.
- Reader/Looper** [23, RLRatio] Ratio entre générateurs en mode « play » et générateurs en mode « loop ». Intervalle de variation {-100, 100, 0, 1}. Pour un réglage à -100, il n'y aura que des players, pour un réglage à 100, il n'y aura que des loopers. Le ratio sera équilibré pour une valeur de 0, et pour une valeur de -33, il y aura en moyenne deux players pour un looper.
- Rgen 1 Amp** [1, RG1Amp] Amplitude du générateur d'aléa numéro 1. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 1 Freq** [9, RG1Freq] Période (exprimée en secondes) du générateur d'aléa numéro 1. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Rgen 2 Amp** [2, RG2Amp] Amplitude du générateur d'aléa numéro 2. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 2 Freq** [10, RG2Freq] Période (exprimée en secondes) du générateur d'aléa numéro 2. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Rgen 3 Amp** [3, RG3Amp] Amplitude du générateur d'aléa numéro 3. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 3 Freq** [11, RG3Freq] Période (exprimée en secondes) du générateur d'aléa numéro 3. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*

- Rgen 4 Amp** [4, RG4Amp] Amplitude du générateur d'aléa numéro 4. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 4 Freq** [12, RG4Freq] Période (exprimée en secondes) du générateur d'aléa numéro 4. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Rgen 5 Amp** [5, RG5Amp] Amplitude du générateur d'aléa numéro 5. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 5 Freq** [13, RG5Freq] Période (exprimée en secondes) du générateur d'aléa numéro 5. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Rgen 6 Amp** [6, RG6Amp] Amplitude du générateur d'aléa numéro 6. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 6 Freq** [14, RG6Freq] Période (exprimée en secondes) du générateur d'aléa numéro 6. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Rgen 7 Amp** [7, RG7Amp] Amplitude du générateur d'aléa numéro 7. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 7 Freq** [15, RG7Freq] Période (exprimée en secondes) du générateur d'aléa numéro 7. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Rgen 8 Amp** [8, RG8Amp] Amplitude du générateur d'aléa numéro 8. Intervalle de variation {0, 100, 0, 1}. *Non opérationnel.*
- Rgen 8 Freq** [16, RG8Freq] Période (exprimée en secondes) du générateur d'aléa numéro 8. Intervalle de variation {0.01, 3600, 1, 0.01}. *Non opérationnel.*
- Scheduler** [24, SchAct] Activité (on/off, ou marche/arrêt) du scheduler. Intervalle de variation {0, 1, 1, 1}, ce qui équivaut à un off/on. Attention, interrompre le scheduler va progressivement bloquer l'ensemble du système. Il devra très probablement être débloqué manuellement.
- Speed Var.** [103, SpeedVar] Variation de vitesse appliquée à un lecteur spécifique, par l'intermédiaire d'un contrôle placé dans un « Displayer ». Intervalle de variation {0.01, 10, 1, 0.001}.
- Synchro** [84, Synchro] Intervalle de synchronisation pour des players/loopers, exprimé en secondes. Intervalle de variation {0.001, 120, 1, 0.001}. *Non opérationnel.*
- Tempo** [93, Tempo] Intervalle de variation {0.1, 480, 60, 0.1}. *Non opérationnel.*
- Tempo Sync.** [92, TempoSync] Intervalle de variation {0, 1, 0, 1}. *Non opérationnel.*
- Transpose** [106, Trnspose] Transposition d'un clip lors de sa lecture, variation exprimée en demi-tons. Intervalle de variation {-24, 24, 0, 1}. Notons que cette valeur remplace toute autre variation de vitesse réglée par d'autres paramètres : c'est le dernier utilisé qui est pris en compte.

- UI Theme** [82, UITheme] Numéro du thème utilisé pour la présentation de l'interface graphique du Game Master. Intervalle de variation {0, 16, 0, 1}.
- Vol Switch Time** [108, VSwitchT] Intervalle de temps moyen, en mode automatique, séparant deux changements consécutifs de la valeur du volume. Intervalle de variation {2, 1200, 60, 1}.
- Vol Variation** [111, VVariat] Réglage de l'aléatoire dans les variations globales de volume. Intervalle de variation {0, 100, 10, 1}.
- Vol. Glob.** [128, VolGlob] Volume global des players. Intervalle de variation {-80, 20, 0, 0.1}.
- Vol. Max.** [116, VolMax] Volume global maximum en mode automatique. Intervalle de variation {-60, 20, 0, 0.1}.
- Vol. Min.** [115, VolMin] Volume global minimum en mode automatique. Intervalle de variation {-60, 20, 0, 0.1}.
- Vol. Mod.** [91, VolMod] Variation du volume. Intervalle de variation {-40, 40, 0, 0.1}.
- Vol.glob.bias** [117, VolGlobB] Vitesse de modification effective du volume de sortie, après une modification du volume global. 0 correspond à une variation instantanée, 10 à une variation très lente. Intervalle de variation {0, 10, 3, 0.1}.

11.2 Indicateurs

Les indicateurs peuvent être affichés (mais non modifiés) dans différents blocs de contrôle, et témoignent de l'activité du logiciel. Certains indicateurs sont relatifs aux générateurs, et indiquent donc une valeur propre au générateur sélectionné. Notons que tous les paramètres décrits ci-dessus au paragraphe 11.1 page 183 constituent également des indicateurs, dont la valeur peut être affichée dans les mêmes conditions.

Clip # Numéro du clip courant (relatif au « player courant »).

Last Kmd Dernière commande exécutée par le player courant.

Vol. bias Bias du volume du player courant.

Chapitre 12

Actions

Le *Game Master* propose un certain nombre de fonctions, qui sont accessibles, depuis certaines parties du logiciel, à travers un numéro d'action, et d'éventuels paramètres.

12.1 Liste des actions prédéfinies

Elles sont ici classées par leur numéro d'ordre, croissant, à partir de 0. Dans certains cas, un autre numéro (débutant à 300) a été associé aux mêmes actions, pour des raisons de compatibilité. Cette seconde notation disparaîtra probablement dans une prochaine version. Certaines de ces actions n'ont de sens que lorsqu'elles font partie d'une séquence, et influent sur le comportement des actions suivantes. Notons aussi que certaines de ces actions consistent à jouer un clip. Par défaut, ce clip est joué avec les paramètres du *partiel zéro*, dont il est possible de modifier le mode de jeu, le mode d'espace, etc.

Numéro	Nom	Id.	Description
0	None	None	Action neutre
1/300	Auto Play off	AutoOff	Arrêt du mode « autoplay »
2/301	Auto Play	AutoOn	Activation du mode « autoplay »
3/302	Auto Play on/off	AutoTgl	Toggle (bascule, inversion) du mode « autoplay »
4/303	Stop all Players	PlayStop	Arrêt de tous les players (le mode « autoplay » n'est pas modifié)
5/304	Stop and Release	PlStopR	Arrêt du mode « autoplay » et de tous les players.

6/305	Display pad #	ShowPad	Affichage sur APC Mini (si connecté/disponible) du numéro de mode (de 0 à 9, mais seuls 0 à 7 disponibles).
7/306, <gr>	Select Group	SelGrp	Passage au groupe « gr » (utilisé en mode « autoplay »)
8/307	Next Group	NextGrp	Passage au groupe suivant.
9/308	Temp Play Flags	TmpPlFlg	<p>Sélection temporaire d'une combinaison pour les « play flags », qui est une valeur numérique, indiquant les contraintes suivantes :</p> <ul style="list-style-type: none"> 1 jouer le clip en entier. Le player utilisera des fade-in et fade-out très courts. 2 utiliser un fade-in très court, et un fade-out long (de l'ordre de la moitié de la longueur du clip) 4 jouer à la vitesse 1 8 jouer le clip depuis son début 16 ne pas appliquer d'échange au hasard dans l'ordre des HP définis dans la configuration 32 jouer à un volume fixe (moyenne des volumes min et max indiqués dans le mode de jeu) 64 protéger la lecture de l'extinction par des clips plus récents (c.f. 331). <p>Tous ces paramètres peuvent être combinés par addition (1+4=5, jouer le clip en entier et à vitesse 1, etc.)</p>
10/309	Default Play Flags	DefPlFlg	Sélection par défaut d'une combinaison pour les « play flags » (c.f. commande 9/308).
11/310	Sensor pad 0	SelPad0	Sélection du pad MIDI 0
12/311	Sensor pad 1	SelPad1	Sélection du pad MIDI 1
13/312	Sensor pad 2	SelPad2	Sélection du pad MIDI 2
14/313	Sensor pad 3	SelPad3	Sélection du pad MIDI 3
15/314	Sensor pad 4	SelPad4	Sélection du pad MIDI 4

16/315	Sensor pad 5	SelPad5	Sélection du pad MIDI 5
17/316	Sensor pad 6	SelPad6	Sélection du pad MIDI 6
18/317	Sensor pad 7	SelPad7	Sélection du pad MIDI 7
19/318	Sensor pad next	SelNxPad	Arrêt du mode « autoplay » et passage au mode MIDI suivant, de manière circulaire (de 2 à 3, de 5 à 6, de 7 à 0).
20/319	Sensor pad previous	SelPrPad	Arrêt du mode « autoplay » et passage au mode MIDI précédent, de manière circulaire (de 2 à 1, de 5 à 4, de 0 à 7).
21/320	Play clip of group	PlayGrp	Lance le jeu d'un clip du groupe courant
22/321, <sm>	Select SM [tmp] ()	TempSMd	Sélection temporaire d'un mode d'espace. Le prochain clip à être joué utilisera non pas le mode d'espace associé au partiel zéro, mais ce mode d'espace spécifique.
23/322, <pm>	Select PM [tmp] ()	TempPMd	Sélection temporaire d'un mode de jeu. Le prochain clip à être joué utilisera non pas le mode de jeu associé au partiel zéro, mais ce mode de jeu spécifique.
24/323, <SM>	Select SM (p0) ()	SetSMd	Le paramètre est un mode d'espace (space mode) qui remplacera la valeur correspondante dans le partiel 0. Les clips joués ultérieurement le seront donc avec ce space mode spécifique.
25/324, <PM>	Select PM (p0) ()	SetPMd	Le paramètre est un mode de jeu (play mode) qui remplacera la valeur correspondante dans le partiel 0. Les clips joués ultérieurement le seront donc avec ce play mode spécifique.
26/325, <hpc>	Select HPset [tmp] ()	TempHPs	Sélection temporaire d'un numéro de configuration de haut-parleurs. Le prochain clip à être joué utilisera non pas la configuration calculée en fonction du mode d'espace associé, mais cette configuration spécifique.

27/326, <part>	Select Bank [tmp] (_)	TempBnk	Sélection temporaire d'un numéro de partiel spécifique. Le prochain clip à être joué (et seulement celui-ci) utilisera non pas la configuration du partiel 0, mais celle de ce partiel spécifique. On peut utiliser aussi un ensemble « groupe » et « banque », sous la forme « groupe*1000+banque » (ex : 201009 pour banque 9 du groupe 201)
28/327, <dur>	Select Duration [tmp] (_)	TempDur	Sélection temporaire d'une durée (secondes). Le prochain clip à être joué n'utilisera pas la valeur du paramètre correspondant, mais cette valeur spécifique.
29/328, <vol>	Select Volume [tmp] (_)	TempVol	Variation de volume pour le clip (positive ou négative, exprimée en dB).
30/329	Interrupt Cl. off	NoInterr	« Verrouillage » du prochain clip joué contre l'autostop (réglé par « max players ») par des clips plus récents
31/330, <dur>	Select Gr.Fades [tmp] (_)	TmGraFd	Sélection d'une durée (ms) pour les fade-in/fade-out des grains joués en mode loop
32/331, <n>	Set Max Players (_)	SetMaxP	Sélection de « n » players max jouant simultanément, ou, pour n=0, suppression de cette limite
33/332, <n1>, <n2>, ..., <n>, -1	Rand. Event (...)	RndEvnt	La commande tire au hasard un numéro de senseur, et insère celui-ci dans la file d'entrée des événements, comme si ce capteur venait d'être déclenché.
34/333, <par>, <val>	Set Param (_ , _)	SetParam	Affectation au paramètre de numéro <i>par</i> de la valeur <i>val</i> . La liste des paramètres accessibles, et de leurs numéros, est donnée en annexe.
35/334, , 0, [...]	Wait (_ , 0)	Wait	Le premier paramètre indique un délai en secondes (éventuellement fractionnaire), le second est la valeur 0 ; la suite est une liste de commandes, qui sera exécutée au bout du délai indiqué.
36/335, <proba>, [...]	Cond. exec (_)	CondExec	Le paramètre est une probabilité (valeur numérique entre 0 à 1) de poursuivre la séquence des commandes.

37/336	Log Message « ... »	LogMess	Impression d'un message sur le log système.
38/337	SensSet Lock	SensLokS	Verrouillage d'un ensemble de senseurs pour une période donnée. L'« ensemble » est fourni sous la forme d'une combinaison de bits. Pour « verrouiller » les senseurs 3, 5, 6 et 8 pendant 15 secondes, on peut écrire : <code>SensLokS 0/3/5/6/8 15</code>
39/337	Copy Partiel –	CopyPart	Copie d'un partiel, défini par son numéro interne, ou par « groupe » et « banque » (c.f. code « 27 »), dans le partiel « 0 ».
40/341	Unlock Sensors set	SendULkS	Déverrouillage d'un ensemble de senseurs, fourni sous la forme d'une combinaison de bits. Pour déverrouiller les capteurs 7, 4 et 8, on peut écrire : <code>SendULkS 0/7/4/8</code>
41/340	Stop and Clear	PlStopC	Arrêt du mode « autoplay » et de tous les players actifs.
42/341, <n>	Sensor pad #(_)	SensPadX	Arrêt du mode « autoplay » et changement de mode MIDI. La commande attend une valeur numérique, nombre de 0 à 7, qui est le numéro du nouveau mode.
43/342, <s>, <d>	Sensor lock (_,_)	SensLok2	« Verrouillage » d'un capteur. Le capteur de numéro « s » (1 à 9) est rendu désactivé pour une durée de « d » secondes (nombre entier ou fractionnaire).
44/343	MIDI Add Dummy	MIDAddDu	Opération sans paramètre, qui insère un événement MIDI fictif dans l'historique des détections des capteurs. Elle permet d'éviter qu'une séquence telle que [1,2] suivie de « 3 » ne soit reconnue comme [1,2,3] ou [2,3].
45/344	Sensors release	SensRel	« déverrouillage » immédiat de tous les capteurs (c.f. code 342)
46/346, <num>, <dur>	MIDI Add evnt (_,_)	MIDAddEv	« déclencher » l'événement « num » (équivalent à « le senseur num a été activé ») au bout de « dur » secondes.
47/346, <z>	Set vol. bias (_)	SetVBias	Réglage de la vitesse des « fade-out » en cas d'arrêt : de 0 (très court), à 10 (très long)

48/347, <v>	Set volume (_)	SetVol	Réglage du volume global courant (en dB)
49/348, <d>	Sensor lock (_)	SensLok1	« Verrouillage » du capteur courant pour une durée de « d » secondes.
50/349	Play clip (_)	PlayClip	Jeu d'un clip. Le clip dont le numéro est indiqué sera joué avec les paramètres associés au partiel 0, sauf si une commande « 326 » antérieure a temporairement changé le numéro de partiel à utiliser comme modèle.
51/350, <cl>	Play cl. unmod. (_)	PlayUnmd	Jeu d'un clip. Le clip dont le numéro est indiqué sera joué non modifié (vitesse de lecture 1, etc.)
52/351, <cl1>, <cl2>, ..., <clN>, -1	Play Rand. Cl. (...)	PlayRnd	La commande est suivie d'une séquence de numéros de clips (nombre compris entre 0 et 9999), terminée par la valeur « -1 ». La commande va jouer l'un des clips, tirés au hasard, dans la séquence. Le clip choisi sera joué avec les paramètres associés au partiel 0 (sauf si, comme ci-dessus, une commande « 326 » antérieure...).
53/352, <cl>, <I>	Play clip (_,FI)	PlayFlg	Jeu d'un clip, en positionnant des indicateurs spécifiques concernant les modes de jeu. « cl » est le clip à jouer; « I » est une valeur numérique (c.f. commande 9/308).
54/353, 0, <cl1>, <cl2>, ..., <clN>, -1	Play RR. Cl. (0,...)	PlayRndN	La commande est suivie d'un zéro, puis d'une séquence de numéros de clips (nombres compris entre 0 et 9999), terminée par la valeur « -1 ». La commande, à chaque exécution, va jouer le clip suivant de la séquence, avec retour au premier en fin de séquence.
55/354, 0, <cl1>, <cl2>, ..., <clN>, -1	Play Seq. Cl. (0,...)	PlaySeq	La commande est suivie d'un zéro, puis d'une séquence de numéros de clips (nombres compris entre 0 et 9999), terminée par la valeur « -1 ». La commande, à chaque exécution, va jouer le clip suivant de la séquence, mais contrairement à la commande précédente, devient inactive après le jeu du dernier clip, chaque clip de la liste ayant été joué, dans l'ordre, une fois et une seule.

56/355, <cnt>, <cl1>, <cl2>, ..., <clN>, -1	Play Ex. Cl. (0,...)	PlayExh	La commande est suivie d'un compteur (un entier positif), puis d'une séquence de numéros de clips (nombres compris entre 0 et 9999), terminée par la valeur « -1 ». La commande va jouer l'un des clips, tiré au hasard, dans la séquence. Le clip choisi sera joué avec les paramètres associés au partiel 0 (sauf si, comme ci-dessus, une commande « 326 »...). Le compteur est décrémenté à chaque jeu d'un clip, et il n'y a plus de jeu de clip lorsque le compteur devient égal à 0.
57/356, 0, ..., -111	Play 0,{clips}	PlaySet	La commande est suivie du nombre 0, puis d'une série de groupes de 4 valeurs, représentant : <ol style="list-style-type: none"> 1. un numéro de clip 2. une durée de verrouillage du senseur (si la valeur est positive) 3. une valeur du volume (en dB) 4. le numéro du partiel à utiliser pour les autres réglages La commande se termine par la valeur « -111 ». A chaque exécution, la commande joue le clip suivant, puis devient muette lorsque tous les clips ont été joués une fois.
58/357, n	Play from Clipset, n	PlFrSet	Joue un clip venant du set dont le numéro suit la commande.
59/358, n, p	Play from Clipset Limited, n, p	PlFrSetL	Joue un clip venant du set dont le numéro suit la commande. Le second paramètre indique le nombre maximal de clips à jouer, après quoi la commande devient muette.
101		ASVers	Version du script auxiliaire chargée, ou 0 si aucun script n'est chargé.
102		ASFTrc	Traces du script auxiliaire.
103	ASRetX, (X)	ASRetX	« Xème » résultat de la dernière exécution du script. Une valeur « Unavailable » indique que l'exécution n'est pas encore terminée. « X » peut être compris entre 0 et 3.

104		ASRet	(premier) Résultat de la dernière exécution du script. Une valeur « Unavailable » indique que l'exécution n'est pas encore terminée.
105		ASWDone	(bool) Vrai si la dernière requête envoyée au script est achevée.
106		ASWBusy	(bool) Vrai si le script est en mode « délai ».
107		Unavail	Fournit la valeur « Unavailable ».
111, <v>		ASSetTr	Positionne les traces d'exécution du script. Le résultat est la valeur précédente des indicateurs de trace.
112, <n>		GoNtMark	Aller au nième marqueur dans l'ordre de la lecture.
113, <m>		GoMark	Aller au marqueur de numéro « m ».
114		StrtRead	Jouer à partir de la position du curseur d'édition. Interrompt l'enregistrement s'il y en a un en cours.
115		StopRead	Interrompt la lecture ou l'enregistrement en cours.
116		PauzRead	Toggle pause/play en lecture (ou en enregistrement).
117		GetPPos	Fournit la position de la tête de lecture REAPER en secondes.
118		GetPPos2	Fournit la position en secondes du prochain bloc à lire par REAPER.
119		GetPStat	<p>Fournit l'état du mode lecture de REAPER. Combinaison de :</p> <p>1 Il est en cours de lecture</p> <p>2 Il est en pause</p> <p>4 Il est en enregistrement</p> <p>Le résultat peut donc être :</p> <p>0 A l'arrêt</p> <p>1 En lecture</p> <p>2 En pause durant la lecture</p> <p>5 En enregistrement</p> <p>6 En pause durant l'enregistrement</p>
120		EdtCurP	Fournit (en secondes dans le projet) la position du curseur d'édition de REAPER.

121,<p>		MvEdPos	Déplace la position du curseur d'édition de REAPER à la seconde <p>. Si REAPER est en lecture, celle-ci se poursuit à la position courante.
122,<p>		MvEdPos2	Déplace la position du curseur d'édition de REAPER à la seconde <p>. Si REAPER est en lecture, celle-ci se poursuit à la nouvelle position.
123		MvEdStrt	Positionne le curseur d'édition de REAPER au début du projet.
124		ProjPRat	Fournit le « play rate » courant du projet REAPER : entre 0.25 et 4, 1 pour une vitesse normale, 2 pour une vitesse double, etc.
125,<A>		ReAct	Exécute une action REAPER. <A> est le numéro de l'action, qui doit être un entier entre 0 et 65535. C.f. le menu « Actions » de REAPER pour la liste de celles-ci.
128		ASStop	Arrêt du script auxiliaire.

12.2 Séquences d'actions

Les différentes procédures qui acceptent ces commandes prennent également en compte les *séquences* d'actions. Une telle séquence est une suite de commandes, chacune avec ses paramètres, la séquence débutant par la taille effective (en nombre d'éléments) de la séquence, cette taille n'incluant pas la valeur de la taille (une séquence de 2 éléments, « A » et « B », est donc notée « 2, A, B »).

12.3 Utilisation des actions

La « programmation » des senseurs fait appel à nombre de ces commandes. On en trouvera des exemples dans le fichier de description « config-Sensors.mSL ».

Chapitre 13

Fichiers de configuration

Ce chapitre aborde les différents fichiers de configuration utiles à l'exécution du « *Game Master* ». A partir de la version 1.5.5, ces fichiers sont décrits en *mSL*, le langage de script intégré. Les fichiers de configuration écrits dans ce langage utilisent le suffixe « *.mSL* ».

Le langage *mSL*, lui-même, est décrit au chapitre 7 page 149. Rappelons simplement que l'on peut aussi placer dans un fichier de configuration des commentaires, similaires à ceux du langage « C », qui, pour un commentaire de ligne, débutent par la séquence de caractères « *//* » et se terminent à la fin de la ligne, et qui, pour un commentaire de bloc, débutent par la séquence de caractères « */** » et se terminent par la séquence de caractères « **/* ».

Tous les fichiers du système se définissent au moyen d'une même fonction, de nom « *table* », qui prend un tableau comme paramètre. En général, ce tableau est lui-même défini sous une forme symbolique, décrite ci-dessous, au moyen d'une directive « *data* ». La syntaxe des fichiers est relativement simple, et consiste en une séquence de mots-clefs et de valeurs numériques.

13.1 Les listes de fichiers de configuration

Le concept des fichiers de configuration a déjà été évoqué au § 1.11 page 40. Les deux fichiers *GM_txt_configs.txt* et *GM_mSL_scripts.txt* peuvent être générés automatiquement par le script, « *mkConfigs.sh* », ou encore peuvent être édités manuellement au moyen d'un utilitaire tel que « *TextEdit* » sous Mac. Ces fichiers peuvent dès lors être sélectionnés directement grâce au menu de choix de scripts. Une autre méthode pour lire un fichier consiste à faire un *glisser-déposer* du fichier depuis une fenêtre du *finder*/explorer jusqu'à la fenêtre du *Game Master*.

13.2 Syntaxe de la commande « **data** »

La commande « **data** » est suivie d'une liste d'éléments placés entre parenthèses. Tous ces éléments sont des *constantes* du langage « mSL », c'est-à-dire des nombres ou des chaînes de caractères.

- Les chaînes de caractères sont représentées par une suite de caractères placée entre doubles quotes « " ».
- Les nombres sont représentés :
 - par des notations traditionnelles (flottants, entiers représentés en décimal, hexadécimal ou en binaire);
 - par la codification depuis l'ASCII d'une séquence de 1 à 4 caractères, placée entre quotes simples « ' ».
 - par une codification interne représentant les identificateurs *mSL*. Ainsi, à l'intérieur d'un « data », le mot « toto » est représenté par la codification interne du symbole *toto*. Attention, la *casse* des caractères est significative !
- par certaines notations reconnues uniquement à l'intérieur d'une séquence « *data* » :
 - une valeur numérique précédée du caractère « - » est codifiée comme l'opposé de cette valeur numérique.
 - deux valeurs numériques séparées par le caractère « + » sont remplacées par leur somme.
 - deux valeurs numériques séparées par le caractère « | » sont remplacées par leur « ou » logique.
 - deux valeurs numériques séparées par le caractère « & » sont remplacées par leur « et » logique.
 - deux valeurs numériques séparées par le caractère « / » sont considérées comme des numéros de bits dans un entier. Le résultat est l'entier dont les bits correspondants sont mis à 1. Ainsi, « 6/2/0 » est calculé comme « $2^6 | 2^2 | 2^0$ », fournissant la valeur « $64 | 2 | 1$ », soit « 67 ».

Les éléments de ces listes peuvent être séparés par des virgules « , » ou des espaces (blanc, tabulation, passage à la ligne, etc.). Il est possible de placer, à l'intérieur d'une commande *data*, des commentaires de ligne (débutant par « // ») ou des commentaires de blocs (encadrés par « /* » et « */ »).

13.3 Configuration des canaux/haut-parleurs

Les configurations de canaux (et, a priori, de haut-parleurs) définissent des zones logiques ou physiques relatives au lieu d'écoute : des localisations quasi géogra-

phiques, liées à une position et une orientation supposées de l'auditeur, telles la gauche, la droite, l'avant, l'arrière, le haut, le bas, et leurs combinatoires, des localisations conceptuelles, telles un point isolé, un cluster de haut-parleurs, un groupe de haut-parleurs distants les uns des autres, ou encore des combinaisons de ces différentes approches, comme une ligne horizontale ou verticale de haut-parleurs, etc.

Ces configurations sont décrites dans un fichier externe, de nom « `config-HP.mSL` » (nom par défaut, mais tout autre nom de fichier est acceptable). Chaque configuration précise le nombre de HP qu'elle décrit (entre 1 et 64). Les configurations sont classées implicitement et informellement en catégories, repérées par un chiffre « rond » de centaine. On peut par exemple décider que « 101 » et suivants sont les haut-parleurs les plus « proches » d'un point donné, « 201 » et suivants les haut-parleurs les plus « éloignés » d'un point donné, etc.

Il est possible de décrire un nombre arbitraire de configurations, mais tous les numéros de 0 à 100000 n'ont pas nécessairement à être utilisés, et les configurations décrites n'ont pas à porter des numéros consécutifs. Par exemple, si l'on utilise une configuration non définie (disons « 683 »), le système va utiliser une configuration valide, tirée au hasard dans la même catégorie « 600 », telle que 643 ou 627. Pour rendre les choses explicites, on utilisera par exemple « 100 » pour désigner un ensemble, choisi au hasard, de HP groupés, « 700 » pour désigner un ensemble, choisi au hasard, de 8 HP dispersés, etc. – si ces valeurs correspondent à la convention que vous avez adoptée, bien sûr.

Il est ainsi possible, en fonction de ses envies, de créer nombre de configurations, classiques, ou non : cubes, quads, octophonies diverses, cercles, dômes, plans, etc.

La syntaxe du fichier « `config-HP.mSL` » est particulièrement simple. Il débute par deux lignes contenant :

```
table(data(
    Table HPConf Clear
```

et se termine par une ligne contenant :

```
End ));
```

Les deux premières lignes correspondent à un appel de la fonction « `table` » dont le rôle est de définir une structure de données du Game Master, fonction qui prend comme paramètre le résultat de la directive « `data` » qui permet de définir un tableau de nombres, correspondant à la description des données destinées à cette fonction `table`. Cette description débute par le mot-clef « `Table` », qui indique qu'une des tables du système va être modifiée, du nom de cette table, ici « `HPConf` », et enfin (c'est optionnel) du mot-clef « `Clear` », demandant à ce que cette table soit réinitialisée avant de recevoir de nouvelles données. Il est possible également de placer une commande « `FIdent` », suivi d'un nombre entier, qui permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à

99), évolution (de 0 à 99), calculé par la formule « version * 10000 + sous-version * 100 + évolution ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207. A la place de « Clear », il est possible d'utiliser « Default », qui va créer quelques entrées appropriées à une configuration Stéréo.

L'épilogue contient le mot clef « End », qui indique que l'on a terminé la description des données. La dernière ligne contient deux parenthèses fermantes et un point-virgule, nécessaires pour respecter la syntaxe du langage *mSL*.

Entre les deux, se trouve la description des configurations de haut-parleurs, dont chacune est réalisée par une séquence de nombres entiers, séparés les uns des autres par une virgule, un espace ou un passage à la ligne. Le premier de ces nombres est une identification de la configuration, valeur comprise entre 0 et une valeur maximale relativement élevée (dépendant de la configuration du « Game Master », par exemple, 2000), qui doit être unique. Le second est le nombre de haut-parleurs décrits par la configuration, un entier « *N* » compris entre 1 et 64. Il est suivi par la liste de « *N* » éléments des numéros (1 à 64) des haut-parleurs de la configuration. Enfin, un nombre supplémentaire, égal à 0, indique la fin de la description¹. Une configuration de « *N* » HP nécessite donc « *N*+3 » valeurs. Voici un exemple de 3 configurations, extraites d'un fichier décrivant une installation spécifique :

```
39, 1, 39, 0,
610, 5,
    10, 1, 5, 47, 52, 0,
1201, 4, // La configuration 1201 comporte 4 HP
    1, 3, 8, 10, // ce sont les HP 1, 3, 8 et 10
    0, // on termine par un zéro.
```

La première, de numéro « 39 », décrit une configuration d'un seul haut-parleur, le numéro « 39 », justement. La seconde, de numéro « 610 », décrit une configuration de 5 HP éloignés les uns des autres, contenant le HP 10. La dernière enfin, de numéro « 1201 », décrit une configuration contenant les HP situés aux quatre coins du bas de la salle. On notera que le format est libre, que les virgules sont facultatives, les indentations et passages à la ligne permettant de rendre plus claire la description des données. Les numéros des configurations n'ont pas à être successifs, et les configurations peuvent être décrites dans un ordre arbitraire.

Ces numéros de configuration sont utilisées dans un autre fichier, qui décrit les « Modes d'espace » (c.f. paragraphe 13.6 page 208).

Le fichier est lu à l'ouverture du « Game Master », mais peut à tout moment être modifié au moyen d'un éditeur externe et être rechargé depuis le logiciel, sans interrompre ce dernier.

1. Il y a une redondance dans le fait de préciser le nombre de haut-parleurs, avant d'en donner leur liste, et le fait d'ajouter un « 0 » après cette liste, mais cette précaution permet de déceler au plus vite les erreurs dans la description des configurations.

13.4 Configuration des clips

Les clips sont localisés dans un répertoire spécifique, de nom, par défaut, « WAVES ». Le fichier de configuration des clips va permettre d'associer à chaque clip un réglage de volume. Ces informations sont décrites dans un fichier, de nom « config-Clips.mSL ».

La syntaxe des opérations est la suivante :

Clear indique que l'on veut réinitialiser toutes les informations concernant les clips.

FIdent suivi d'un nombre entier permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à 99), évolution (de 0 à 99), calculé par la formule « version * 10000 + sous-version * 100 + indice ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207.

ClipDir suivi d'une chaîne de caractères (placée entre doubles quotes) indique le nom du répertoire où se situent les clips. Ce répertoire est en général un sous-répertoire de « Data » contenu dans le répertoire système des ressources de REAPER, ou un sous-répertoire de « Data » contenu dans le répertoire du projet courant.

ClipFmt suivi d'une chaîne de caractères (placée entre doubles quotes) décrit la syntaxe des noms des clips. Sa valeur par défaut est « "clip1%04d.%s" ». Le segment « %04d » dont la présence est obligatoire, indique que l'édition du numéro de clip se fera sur 4 chiffres, complétés par des zéros. Le suffixe « %s », qui décrit l'édition d'une chaîne de caractères, va permettre au logiciel de placer l'extension appropriée au type de fichier (« wav », « aiff », « flac », etc.).

ClVol suivi d'une valeur en décibels, définit le changement de volume appliqué à tous les numéros de clips qui suivent.

ClRange suivi de deux numéros de clips, va appliquer le changement de volume à tous les clips compris dans l'intervalle défini par ces deux numéros.

Un nombre indique un numéro de clip.

Les commandes **Clear**, **ClipDir**, **ClipFmt** et **FIdent** seront typiquement utilisées une fois au plus, et apparaîtront en tête. Voici un exemple de contenu du fichier :

```
/* Configuration des clips */
table(data(
    Table Clips Clear
        ClVol -3      // volume courant à -3 dB
        ClRange 7200 7257 // pour les clips 7200 à 7257
        ClVol -2.5 7260 7261 // volume à -2.5dB pour
```

```

7266 7281 7252 7721 7726 // ces clips-ci
7712 7742 7743 7751 // et ceux-ci également
ClVol -4 7264 33 21 325// -4 dB
ClVol 3 6602 2510 2514 2517 2518 2524 2525
ClVol -12 2358 ClVol -9 2359 2363 2364
ClVol 10 6609
// boost +9 dB des 2000 premiers clips
ClVol 9 ClRange 0 1999
ClVol 8 6611 ClVol 7 521 6604
End
));

```

On notera, comme dans la descriptions des autres fichiers de configuration, que le fichier débute par le prologue habituel :

```

table(data(
    Table Clips Clear

```

et se termine, de même, par l'épilogue :

```

End
));

```

13.5 Configuration des ClipSets

Les ClipSets sont définis par un fichier externe, de nom « config-ClSets.mSL » (nom par défaut, mais tout autre nom de fichier est acceptable). Un *ensemble* de clips (« Set ») est repéré par un numéro, compris entre 0 et 999, choisi par l'utilisateur, mais tous les numéros n'ont pas nécessairement à être utilisés, et les configurations décrites n'ont pas à porter des numéros consécutifs. La syntaxe du fichier « config-ClSets.mSL » est relativement simple. Le fichier débute par deux lignes contenant :

```

table(data(
    Table ClipSets Clear

```

et se termine par une ligne contenant :

```

End ));

```

Les deux premières lignes correspondent à un appel de la fonction « table » dont le rôle est de définir une structure de données du Game Master, fonction qui prend comme paramètre le résultat de la directive « data » qui permet de définir un tableau de nombres, correspondant à la description des données destinées à cette fonction table. Cette description débute par le mot-clef « Table », qui indique qu'une des

tables du système va être modifiée, suivi du nom de cette table, ici « `ClipSets` », et enfin (c'est optionnel) du mot-clef « `Clear` », demandant à ce que cette table soit réinitialisée avant de recevoir de nouvelles données. Il est possible également de placer une commande « `FIdent` », suivie d'un nombre entier, qui permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à 99), évolution (de 0 à 99), calculé par la formule « $\text{version} * 10000 + \text{sous-version} * 100 + \text{indice}$ ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207.

L'épilogue contient le mot clef « **End** », qui indique que l'on a terminé la description des données. La dernière ligne contient deux parenthèses fermantes et un point-virgule, nécessaires pour respecter la syntaxe du langage *mSL*.

Entre les deux, se trouve la description des configurations d'ensembles de clips. Une description débute par le mot clef « `Set` », suivi du numéro (0 à 999) du *set* que l'on va définir. Ce numéro ne doit pas avoir déjà été utilisé dans les descriptions antérieures de ce fichier. Le numéro est suivi d'un type d'algorithme de sélection d'un clip dans l'ensemble décrit. Imaginons un ensemble qui va contenir 200 clips, numérotés de 0 à 199. Les algorithmes disponibles sont désignés par un mot-clef, qui est l'un des suivants :

- « **RR** », *Round Robin* : les clips vont être délivrés en ordre séquentiel, du numéro 0 au numéro 199, puis repartir du numéro 0. Tous les clips sont donc utilisés, à une différence de 1 près, un nombre égal de fois.
- « **ARR** », *Alternate Round Robin* : les clips vont être délivrés en ordre séquentiel inverse, du numéro 199 au numéro 0, puis repartir du numéro 199. Tous les clips sont donc utilisés, à une différence de 1 près, un nombre égal de fois.
- « **RRR** », *Random Round Robin* : les clips vont être délivrés en ordre aléatoire, mais vont être utilisés, à une différence de 1 près, un nombre égal de fois. Un clip ne sera fourni une seconde fois que quand tous les autres clips auront été fournis au moins une fois. Cet ordre aléatoire sera différent pour chaque cycle recouvrant l'ensemble des clips.
- « **Random** » : Les clips sont délivrés dans un ordre purement aléatoire. La répartition suit une loi de Gauss, et ainsi, sur 1000 tirages parmi les 200 clips de notre exemple, beaucoup de clips seront tirés 5 fois, moins de clips sortiront 4 ou 6 fois, peu d'entre eux 0, 1, ou encore 9 fois ou plus.

Si le type d'algorithme n'est pas spécifié, c'est « *Random Round Robin* » qui est utilisé par défaut.

Ce mot-clef peut éventuellement être suivi de « **PLUC** » (acronyme de « *Prefer Less Used Clips* »), demandant au *clipset* de fournir de préférence les clips les moins utilisés. A quoi sert cette précision, alors que l'on dispose déjà d'algorithmes de *Round Robin* garantissant un tirage équitable des clips ? La différence est la suivante :

- Les *Round Robin* vont utiliser les clips de manière équitable, *indépendamment* du nombre de fois où ils ont été antérieurement exécutés. Ainsi, si dans

un ensemble de clips, les *min* et *max* d'utilisation sont 0 et 10, après nombre de tirages, ces *min* et *max* seront peut-être devenus 4 et 14, avec toujours le même écart entre *min* et *max*.

- Avec l'option « PLUC » et une même configuration initiale, le logiciel va tenir compte de l'*utilisation antérieure* des clips, et ces valeurs *min* et *max* deviendront peut-être 11 et 12.

Après cet en-tête, on peut trouver (en nombre et en ordre arbitraires) les commandes suivantes :

- « **Clips** » ou « Clps » : introduit une liste de clips qui feront partie de l'ensemble défini. La liste peut comporter jusqu'à 1000 clips.
- « **ClRange** » ou « Range » : introduit deux numéros de clips, *p* et *q*, tous les numéros de clips compris entre *p* et *q* inclus feront partie de l'ensemble défini.
- « **Copy** » ou « Idem » : permet de recopier dans le set courant tous les éléments du set précisé après ce mot-clef. Ce numéro du set est compris entre 0 et 999, et doit être déjà défini par une description antérieure.
- « **Xclude** » ou « Xcept » : permet d'indiquer une liste de clips qui seront explicitement exclus de l'ensemble défini. Cette exclusion s'applique aux clips déjà existant dans le set courant. Des clips exclus par cette commande peuvent être (volontairement ou non) réintroduits par une commande ultérieure.
- « **MinDur** », pour *Minimum Duration*. Ce mot-clef est suivi d'une valeur en secondes (comme 0.25 ou 15), qui indique que les clips lus à vitesse normale d'une durée inférieure à celle-ci ne seront pas conservés dans le *set*. Défaut : 0.001. Il s'agit d'une valeur globale, s'appliquant à tous les clips du set.
- « **MaxDur** », pour *Maximum Duration*. Ce mot-clef est suivi d'une valeur en secondes (comme 10.5 ou 45), qui indique que les clips lus à vitesse normale d'une durée supérieure à celle-ci ne seront pas conservés dans le *set*. Défaut : 3600. Il s'agit d'une valeur globale, s'appliquant à tous les clips du set.

La description d'un set se termine lorsque l'un des mots-clefs « Set » (annonçant la description d'un nouvel ensemble) ou « End » (terminant la description de la table des ClipSets) est rencontré. Notons que, comme le suggère le mot « Set », tous les clips d'un tel ensemble sont différents. Notons aussi qu'il n'est pas équivalent d'utiliser, dans des partiels différents (c.f. § 13.8 page 213) le même numéro de *set* ou des *set* identiques, puisque l'algorithme de *Round Robin* est associé au *set*. Dans le premier cas, les partiels se partageront des numéros de clips fournis le même *RR*, dans le second cas, les clips seront fournis à chaque partiel par des *RR* différents.

Voici un exemple de fichier de configuration :

```
/*      Configuration des ClipSets */
var s = stralloc();
var id = "Configuration des Clipsets";
```

```

sprintf(s, "%s START", id); GM_Log(s);
var z = table(data(
    //=====
    Table ClipSets Clear
        FIdent 10502
        Set 80 Random
            MinDur 8
            MaxDur 90
            ClRange 0 9999
        Set 100 RRR
            Idem 80
            Xclude 2500 2501 2502 2503
        Set 106
            Clips 3100 3103 3110 2167
            ClRange 2100 2180
            Clips 2501 2153
            ClRange 2510 2515
            ClRange 2520 2548
            Xcept 2031 2132
            ClRange 3200 3299
            Xcept 3217 3338
            ClRange 3300 3390
            Range 100 190
    End
    //=====
));
sprintf(s, "%s END - Res: %d", id, z); GM_Log(s);
strfree(s);

```

Dans cet exemple, la version du fichier de configuration est dite 1.5.2 (FIdent 10502). Le fichier définit un premier ensemble, le *Set 80*, qui recouvre tous les clips (de 0 à 9999), mais en ne sélectionnant que ceux dont la durée est comprise entre 8 et 90 secondes, et en leur assignant un type de choix « *Random* ». Le *Set 100* contient (Par « Idem 80 ») les mêmes clips, mais *sans* limites sur la durée, mais, par la commande suivante, en en excluant certains clips. Le choix du tirage des clips est *Random Round Robin*. Enfin, le *set 106* (utilisant par défaut *Random Round Robin*) est constitué d'un assemblage de différents ensembles de clips. Les instructions sont exécutées en ordre séquentiel, et l'on notera, par exemple, que le clip 3338, explicitement exclu par « Xcept 3217 3338 », est en fait réintroduit dans l'ensemble par la commande suivante.

Notes

1. Lorsque l'option « PLUC » est activée, il peut arriver que le même clip soit sélectionné deux ou trois fois de suite. En effet, une première demande de clip peut choisir le clip le moins utilisé du set. Ce numéro de clip va être transmis à un lecteur, qui va ouvrir le fichier correspondant, et, si tout se passe bien, incrémenter le compteur d'utilisation associé à ce clip. Cette demande et cette utilisation sont effectuées par des processus différents. Une nouvelle demande peut survenir avant que le lecteur n'ait effectivement accompli son travail et incrémenté le compteur d'utilisation. La nouvelle demande risque donc de se voir allouer le même numéro de clip.
2. L'algorithme est complexe, et optimise son fonctionnement par une anticipation des demandes : si un clip est demandé, plusieurs autres sont calculés dans le même temps, l'algorithme se basant sur la situation actuelle (nombre d'utilisation des clips à cet instant « T »). Si plusieurs sets différents se recouvrent partiellement ou totalement, et que l'option « PLUC » est utilisé, l'utilisation d'un set va perturber le fonctionnement des autres : des sets différents peuvent alors choisir, du fait de cette anticipation, des clips identiques.

13.6 Modes d'espace

Un « mode d'espace » définit la configuration spatiale liée à la diffusion des échantillons fournis par un player. Les différents aspects d'un mode d'espace sont définis par des mots clefs, qui sont habituellement suivis d'une ou plusieurs valeurs numériques. Les mots-clef et leurs paramètres sont les suivants :

Clear indique que l'on veut réinitialiser toutes les informations concernant les modes d'espace.

FIdent suivi d'un nombre entier permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à 99), évolution (de 0 à 99), calculé par la formule « version * 10000 + sous-version * 100 + indice ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207.

Smd Numéro d'identification associé à ce mode d'espace, un entier positif. Le nombre de modes d'espace que l'on peut définir n'est limité que par la taille allouée à leur représentation (1000 actuellement). Ce mot-clef est obligatoire et doit apparaître en premier, car il définit le numéro associé à ce mode d'espace. Les autres mots-clefs sont facultatifs, et peuvent être placés dans un ordre quelconque.

HPSet Numéro du groupe de HP associés à ce mode (un nombre de 0 à 1999, c.f. description des HP, au paragraphe 13.3 page 200).

HPCnt Nombre de HP à utiliser dans le groupe (valeurs minimale et maximale).

HPMode Modes d'utilisation des HP, un mot-clé « `Frst` », « `ButFirst` » ou « `Rndm` », indiquant que l'on veut, dans un groupe de HP, choisir les « `N` » premiers, exclure le premier et choisir les « `N` » suivants, ou encore en prendre « `N` » tirés au hasard (le nombre étant spécifié par « `HPCnt` », et éventuellement d'autres contraintes liées au mode de jeu).

HPFlags [*] (flags de modification). Un ensemble de flags, permettant de modifier l'utilisation qui est faite de ce groupe. *[non utilisé actuellement]*

HPEv [*] Evolution temporelle de l'utilisation des HP dans le groupe. *[non utilisé actuellement]*

Les différents modes d'espace sont décrits dans le fichier de nom « `config-SModes.mSL` », qui est un fichier texte, raisonnablement lisible, et pouvant être modifié par l'utilisateur. Chaque description d'un mode de jeu doit débiter par « `SMD` » suivi du numéro choisi pour ce mode de jeu. Les mots-clefs et les valeurs numériques doivent être séparés par des espaces, des virgules ou des passages à la ligne. Voici un exemple d'entrées extraites de ce fichier :

```
SMD, 1,
    HPSet, 100
    HPCnt, 2, 8
    HPMode, Frst
SMD, 2
    HPSet, 1000
    HPCnt, 8, 24
    HPMode, Rndm
```

Dans cet exemple, le mode d'espace 1 est basé sur la configuration de HP « 100 », qui représente « une configuration de 32 HP groupés choisie au hasard ». L'entrée « `HPCnt` » permet de spécifier que l'on utilisera seulement de 2 à 8 HP parmi les 32 que définit la configuration, et « `HPMode` » que l'on utilisera les premiers (mot-clé `Frst`) de la liste de 32. De même, le mode d'espace 2 indique que l'on utilisera entre 8 et 24 HP, extraits au hasard (mot-clé `Rndm`) de la configuration « 1000 », représentant elle-même « une configuration de 24 HP distants choisie au hasard ».

Les caractéristiques non spécifiées prennent des valeurs par défaut appropriées. Les modes d'espaces 1 à 12 du fichier représentent les modes d'espaces correspondant aux banques telles que définies dans l'appel à participation « Gran Lux 2020 ».

On notera que le fichier débute par un prologue semblable à ceux de fichiers de configuration déjà décrits :

```
/* Configuration des Modes d'Espace */
table(data(
    Table SpModes Clear
```

et se termine, de même, par l'épilogue :

```
End
) ) ;
```

La seule différence avec l'exemple précédent (hormis le commentaire !) est le nom de la table concernée, ici « SpModes ».

13.7 Modes de jeu

Les « modes de jeu » définissent de quelle manière un clip va être joué. Les différents aspects d'un mode de jeu sont définis par des mots clefs, qui sont habituellement suivis d'une ou plusieurs valeurs numériques. Les mots-clef et leurs paramètres sont les suivants :

Clear indique que l'on veut réinitialiser toutes les informations concernant les modes de jeu.

FIdent suivi d'un nombre entier permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à 99), évolution (de 0 à 99), calculé par la formule « version * 10000 + sous-version * 100 + indice ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207.

PMd un numéro d'identification associé à ce mode de jeu (entier positif). Le nombre de modes de jeu que l'on peut définir n'est limité que par la taille allouée à leur représentation (1000 actuellement).

Kmd Commande associée à ce mode de jeu, qui sera envoyée au *player*. Cette commande est l'un des mots-clefs « KmdPlay » or « KmdLoop ».

Speed Trois valeurs numériques : la vitesse de lecture minimale, ci-après « A », la vitesse de lecture maximale, ci-après « B », et la probabilité d'inversion du sens de la lecture, « C ». La vitesse de lecture est choisie au hasard dans l'intervalle [A, B]. Ainsi, le couple (1,1) définit une vitesse de lecture qui sera toujours la vitesse « normale » égale à 1, (0.8,1.2) une vitesse qui sera comprise entre 0.8 et 1.2, (-1,-1) pour un clip qui doit toujours être lu à l'envers, (0,3) pour un clip qui peut être lu à une vitesse comprise entre 0 et 3, etc. Rappelons que cette vitesse de lecture fonctionne à l'instar d'un magnétophone ; plus la vitesse de lecture est basse, plus le son est décalé vers le grave, plus elle est rapide, plus le son est décalé vers le haut. Notons aussi que dans ce dernier cas, aucun effort n'est fait pour gérer l'aliasing. In fine, la vitesse de lecture est ramenée à l'intervalle [-3, -0.01] ou encore [0.01, 3], selon qu'elle est négative ou positive. La valeur « C » détermine la probabilité d'inversion du sens de la lecture. Pour une valeur égale à 0, la vitesse de lecture définie

par [A,B] est conservée telle qu'elle. Pour une valeur égale à 1, elle est inversée. Pour une valeur égale à 0.35, elle est inversée dans 35% des cas. Si « C » n'est pas précisé, il est pris égal à « 0 ». Si « B » n'est pas non plus précisé, il est pris égal à « A ».

PVol Volume de lecture, une ou deux valeurs numériques, définissant les valeurs minimale « D » et maximale « E » du volume de base. Ces valeurs sont exprimées en dB. Le volume est tiré au hasard dans l'intervalle [D, E], et est contraint dans l'intervalle [-120, 12] (toujours en dB). Ainsi, pour le couple (-6, 0), le volume sera compris entre -6 et 0 dB. Si « E » n'est pas précisé, il est pris égal à « D ».

ChC Nombre de canaux à utiliser en sortie; peut être modifié en fonction du nombre de canaux source du clip joué et du mode d'espace choisi.

FLM Flags, transmis au lecteur, sous la forme de deux entiers.

Gsize Durée d'un grain (mode loop), exprimée par des valeurs minimale et maximale, en secondes, comprises entre 0.001 et 180. Ainsi, (Gsize 3 3) détermine des grains de longueur fixe, égale à 3 secondes, (Gsize 0.5 0.8) des grains dont la durée varie entre 500 et 800 millisecondes.

GDepth Polyphonie moyenne de jeu, nombre de grains simultanément actifs, sous forme de deux valeurs représentant la polyphonie minimale et la maximale. Ces valeurs sont comprises entre 1 et 24.

PDur Play Duration, valeurs minimale et maximale pour la durée de jeu associée à ce mode de jeu. Ces valeurs sont exprimées en secondes, et comprises entre 0.01 et 300.

FDin Fade In, valeurs minimale et maximale pour le fade in du clip. Ces valeurs sont exprimées en millisecondes, et comprises entre 0.01 et 60000. Des valeurs négatives, comprises entre -100 et -1, sont considérées comme un pourcentage de la durée de jeu du clip consacré au fade in.

FDout Fade Out, valeurs minimale et maximale pour le fade out du clip. Ces valeurs sont exprimées en millisecondes, et comprises entre 0.01 et 60000. Des valeurs négatives, comprises entre -100 et -1, sont considérées comme un pourcentage de la durée de jeu du clip consacré au fade out.

GrFDin Grain Fade In, valeurs minimale et maximale pour le fade in des grains du clip en mode loop. Ces valeurs sont exprimées en millisecondes, et comprises entre 0.01 et 60000. Des valeurs négatives, comprises entre -100 et -1, sont considérées comme un pourcentage de la durée de jeu du grain consacré au fade in.

GrFDout GrainFade Out, valeurs minimale et maximale pour le fade out des grains du clip en mode loop. Ces valeurs sont exprimées en millisecondes, et

comprises entre 0.01 et 60000. Des valeurs négatives, comprises entre -100 et -1, sont considérées comme un pourcentage de la durée de jeu du grain consacré au fade out.

FX1 Une ou deux valeurs numériques, définissant les valeurs minimale « F » et maximale « G » du volume d'envoi du son à la piste d'effet « 1 ». Ces valeurs sont exprimées en dB. Ce volume d'envoi est tiré au hasard dans l'intervalle [F,G], et est contraint dans l'intervalle [-120, 12] (toujours en dB). Si « G » n'est pas précisé, il est pris égal à « F ». [non utilisé]

FX2 Volume d'envoi des sons générés vers la piste d'effet 2. [non utilisé]

Les modes de jeu sont définis dans un fichier de commande, de nom « `config-PModes.mSL` », qui est un fichier texte, raisonnablement lisible, et pouvant être modifié par l'utilisateur. Chaque description d'un mode de jeu doit débiter par « `PMd` » suivi du numéro choisi pour ce mode de jeu. Les mots-clefs et les valeurs numériques doivent être séparés par des espaces, des virgules ou des passages à la ligne. Voici un exemple d'entrées extraites de ce fichier :

```
PMd, 3      // Mode n°3
      Kmd, KmdPlay
      PVol, -6, 0
PMd, 4      // Mode n°4
      Kmd, KmdLoop
      Speed, 0.5, 2, 0.5
      Gsize, 3, 3.5
      GDepth, 3, 4
```

Les caractéristiques non spécifiées (comme « `PVol` » ou « `FX1` ») prennent des valeurs par défaut appropriées. Les modes de jeux 1 à 12 du fichier représentent les modes de jeux correspondant aux banques telles que définies dans l'appel à participation « Gran Lux 2020 ».

On notera, comme au paragraphe précédent, que le fichier débute toujours par le prologue :

```
/* Configuration des Modes de Jeu */
table(data(
      Table PlModes Clear
```

et se termine, de même, par l'épilogue :

```
      End
    ));
```

La seule différence avec l'exemple précédent (hormis le commentaire !) est le nom de la table concernée, ici « `PlModes` ».

13.8 Banques

Les banques définissent une hiérarchie dans la classification des objets sonores. Une banque est constituée d'un ensemble de partiels, ainsi définis :

Grp Numéro du groupe auquel appartient la banque virtuelle, un entier positif ou nul. Le nombre de groupes que l'on peut définir n'est pas limité.

Bnk Numéro de la banque définie par ce partiel. Un entier positif ou nul. Le nombre de banques que l'on peut définir dans un groupe n'est pas limité.

Clps Numéros (entre 0 et 9999) du premier et du dernier clip de ce partiel. Il est possible de préciser deux paramètres supplémentaires, qui sont le mode de jeu et le mode d'espace. Par défaut, ces deux valeurs sont prises égales au numéro de banque. Dans la version actuelle, l'intervalle des clips indiqué est transformé en un *Set* de mode *RRR*, avec toutes les propriétés d'un tel set (c.f. § 13.5 page 204).

Set Numéro d'un set (c.f. § 13.5 page 204) décrivant l'ensemble des clips utilisés par ce partiel.

PClass [*] Classe définie par ce partiel. Un entier positif ou nul. Défaut : 0.

PWeight Poids associé aux clips de ce partiel. Cette valeur intervient lors de la sélection d'un clip dans une banque ou un groupe. Défaut : 1.

Plm (PlayMode) Numéro du mode de jeu associé à ce partiel. Défaut : le numéro de la banque.

SpM (SpaceMode) Numéro du mode d'espace associé à ce partiel. Défaut : le numéro de la banque.

prVol Deux entiers, définissant des facteurs de correction de volume min et max, exprimés en dB. Défaut : [0,0].

Notes

- La banque « *b* » du groupe « *g* » est constituée de l'union des partiels qui ont « *g* » comme numéro de groupe (*Grp*) et « *b* » comme numéro de banque (*Bnk*). Il est ainsi possible de créer des banques à partir d'ensembles de clips non contigus et d'affecter à chaque partiel des modes de jeu ou d'espace différents. On ouvre aussi la possibilité de choisir dans une banque les seuls partiels d'une certaine classe (*PClass*) « *c* », ou encore les partiels de toutes les banques avant une certaine classe... Plusieurs partiels peuvent avoir même « *g* », même « *b* » et même « *c* ».
- Les numéros de partiels sont automatiquement attribués par le système, à partir d'un numéro initial qui est 64.

- Les partiels de numéro 0 à 63 peuvent être définis par le mot-clef « `Partiel` » suivi du numéro correspondant. Seuls sont utilisés dans cette définition le *PlayMode* et le *SpaceMode*, que l'on précise par les mots-clefs « `PlM` » et « `SpM` ». Voir le § 3 page 85 pour l'utilisation de ces partiels pour l'écoute des clips définis. Les lettres étant utilisées pour déclencher cette écoute, on peut aussi utiliser, à la place des numéros 10 à 61, les notations « 'a' » à « 'z' » et « 'A' » à « 'Z' » pour désigner ces partiels.
- La *classe* permet ainsi, si on le décide, de définir les caractéristiques communes aux clips du partiel, par exemple « ce sont des sons solistes », ou encore « ils sont courts », « ils sont en 8 canaux », bref, n'importe quel type d'information susceptible d'être pertinente pour les heuristiques de jeu.
- Les options « `Clps` » et « `Set` » sont mutuellement exclusives.
- Notons encore que le nombre total de partiels que l'on peut définir n'est limité que par la taille allouée à leur représentation (5000 actuellement).
- Cependant, dans d'autres circonstances, des heuristiques de jeu peuvent s'appuyer sur des ensembles de banques « virtuelles », qui pourront être extraites de groupes virtuels différents, et filtrées sur d'autres caractéristiques, comme le numéro de banque virtuel (catégorisation), la classe, le mode de jeu ou le mode d'espace.

Comme dans les autres syntaxes, les commandes « `Clear` » et « `Fident` » sont admises en tête.

Clear indique que l'on veut réinitialiser toutes les informations concernant les banques.

Fident suivi d'un nombre entier permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à 99), évolution (de 0 à 99), calculé par la formule « version * 10000 + sous-version * 100 + indice ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207.

Les banques sont définies dans le fichier « `config-Banks.mSL` ». Voici un exemple de définition de trois groupes, 1, 2 et 4.

```
Grp 1 // Groupe 1
  Bnk, 1 // banque 1
    Clps, 0, 99
  Bnk, 2
    Clps 100, 199
Grp 2 // Groupe 2
  Bnk 1
    Clps 1000, 1099
  Bnk 4
    Clps, 1300, 1399
```

```

    Bnk, 7
        Clps 1600 1699
Grp, 4
    Bnk 1, Clps 2000, 2009, PlM 1, SpM 3
    Bnk 2, Clps 2010, 2019, PlM 14, SpM 2
    Bnk 5, Clps 2020, 2029, PWeight 3
        Clps 2020, 2022, PlM 8, SpM 11, PWeight 2
    Bnk 6, Clps 2040, 2049 PlM 1 SpM 4
Partiel 'a' SpM 1 PlM 28
Partiel 'b' SpM 1 PlM 36

```

La mise en page est libre (l'indentation est ici présente pour des raisons esthétiques), il suffit que les nombres ou les mots-clés soient séparés les uns des autres par des espaces, des virgules ou des fins de ligne. Si « Grp » ou « Bnk » ne sont pas précisés, ils conservent leur valeur antérieure, par exemple « 1 » pour « Grp », jusqu'à rencontre de « Grp, 2 ». On remarquera que la banque « 5 » du groupe « 4 » se compose de deux partiels, utilisant des clips communs (« 2020 » à « 2022 »), mais avec des modes de jeu et d'espace différents (tous deux égaux à 5 dans le premier partiel, car ils reçoivent comme valeur le numéro de la banque lorsqu'ils ne sont pas précisés). Dans la description du groupe 4, banques 1 et 2, les modes de jeu et d'espace sont explicitement précisés, après les deux valeurs représentant les bornes des clips. Le fichier se termine par la définition de deux partiels, de numéros 'a' et 'b', soit 10 et 11, qui permettront en mode interactif dans le module « Clip selection » de lire des clips avec les modes de jeu et d'espace indiqués. Précisons encore que c'est la rencontre d'une commande « Grp », « Bnk », « Partiel » ou « End » qui détermine la création d'un nouveau partiel, avec les caractéristiques définies par les commandes précédentes.

Dans cet exemple, 12 partiels sont définis, correspondant à un total de 9 banques.

On notera, comme dans les paragraphes précédents, que le fichier débute toujours par le prologue :

```

/* Configuration des Banques */
table(data(
    Table Banks Clear

```

et se termine, de même, par l'épilogue :

```

    End
));

```

La seule différence avec l'exemple précédent (hormis le commentaire !) est le nom de la table concernée, ici « Banks ».

13.9 Senseurs

Ce fichier (de nom « `config-Sensors.mSL` ») décrit la configuration des senseurs connectés en mode MIDI. Concrètement, le hardware d'une telle l'installation peut se composer d'un Arduino, connecté à des capteurs infrarouge, qui envoie les interactions sous forme de messages MIDI. Ceux-ci sont des messages « Note On », un numéro de note (de 1 à N) étant affecté chaque capteur. Le fichier va décrire les actions que doit prendre le Game Master à la réception de tels messages, sachant qu'il va reconnaître des « mots », c'est à dire des séquences de notes MIDI. Il peut ainsi réagir à des « séquences » constituées d'un événement MIDI unique, ou des séquences constituées de deux, ou plus, événements MIDI. Dans ce dernier cas, le premier sera d'abord détecté comme événement unique, puis la séquence qu'il introduit, éventuellement, comme un second événement.

Le système permet de décrire différentes configurations indépendantes, dans lesquelles des actions différentes peuvent être associées à chaque capteur. Ces configurations sont dites « Modes » (« Scène » conviendrait également), repérés par un numéro compris entre 0 et 31. Il y a donc 32 modes différents possibles. Lorsque l'on se trouve dans un mode donné, le système ne reconnaît que les séquences associées à ce mode.

La syntaxe, comme dans les autres fichiers de configuration, fait appel à des mots-clefs et des valeurs numériques, séparés par des espaces, des virgules ou des passages à la ligne. Elle utilise les éléments suivants :

Clear indique que l'on veut réinitialiser toutes les informations concernant les senseurs.

FIdent suivi d'un nombre entier permet de préciser l'indice d'évolution du fichier sous une forme hiérarchique, en version, sous-version (de 0 à 99), évolution (de 0 à 99), calculé par la formule « $\text{version} * 10000 + \text{sous-version} * 100 + \text{indice}$ ». La version 3, sous-version 2, évolution 7 se notera ainsi 30207.

MMode, <n> début de définition du mode « n », entier compris entre 0 et 31.

DefSeq, <seq> définition d'une séquence de numéros de capteurs. La séquence peut se composer de 1 à 3 numéros de capteurs, eux-mêmes compris (l'installation étant connectée à « N » capteurs) entre 1 et « N ».

DoKmd, <insts> définition d'une suite d'instructions. Chaque séquence « DoKmd » doit suivre une séquence « DefSeq », dont elle est la conséquence : les instructions « *insts* » seront exécutées si, et seulement si, la séquence « *seq* » qui précède a été reconnue.

End cet ordre termine le fichier.

Enter, <insts> les instructions qui suivent « Enter », jusqu'au prochain mot-clef, seront exécutées chaque fois que l'on passe à ce mode MIDI.

13.9.1 Instructions

Les instructions sont repérées par un nombre entier ou un mot-clef, éventuellement suivi des paramètres attendus par l’instruction. Les instructions attendent soit un nombre fixe de paramètres (0, 1, 2...), soit un nombre variable, le dernier étant alors typiquement suivi par la valeur -1. Tous les nombres utilisés doivent être séparés par des virgules ou des passages à la ligne.

Les instructions utilisables sont décrites au chapitre des Actions (c.f. chapitre 12 page 189). On s’y reportera pour une description précise de celles-ci.

13.9.2 Exemples

Dans ce premier exemple, la configuration choisie est le mode MIDI 3.

La séquence d’activation des capteurs « 1 » suivi de « 4 » exécute la commande « 341, 4 », passage au mode MIDI 4, puis « 350, 7 » déclenche le jeu du clip « 7 », et enfin « 343 » nous assure qu’un passage inopiné sous le capteur 1, immédiatement après cette séquence, ne sera pas reconnu comme une nouvelle séquence « 4, 1 ». La ligne suivante définit la séquence « 4, 1 », permettant le retour au mode MIDI 2.

```
// PAD 3
MMode, 3
DefSeq, 1, 4, DoKmd, 341, 4, 350, 7, 343 // Midi mode 4
DefSeq, 4, 1, DoKmd, 341, 2, 343 // ==> Midi mode 2
```

Nous avons ensuite deux séquences de « jeu » de clip. Le capteur « 1 » va permettre le jeu du clip 6612. Le capteur 6 celui du clip 6211. La commande « 352, 6211, 7 » précise, par son dernier paramètre (7=1+2+4), que le clip doit être joué en entier (1), avec un fade-in court et un fade-out long (2), et à la vitesse 1 (4).

```
DefSeq, 1, DoKmd, 350, 6612
DefSeq, 6, DoKmd, 352, 6211, 7
```

Dans les exemples suivants, nous modifions certains paramètres de jeu. Le capteur « 5 » va lancer le jeu du clip 6208 (« 350, 6208 »), après avoir choisi les modes de jeu et d’espace associés au partiel 7 (« 326, 7 »). De même, la séquence associée au capteur « 9 » se compose du choix du partiel 7 pour le jeu, soit « 326, 7 », d’un boost du volume de 18dB, « 328, 18 », et du choix, par un « round robin » du jeu des clips 6201, 6209 et 6210, « 353, 0, 6201, 6209, 6210, -1 » (c’est-à-dire que le capteur 9 va lancer, la première fois le clip « 6201 », la seconde fois le clip « 6209 », puis le « 6210 », puis à nouveau le « 6201 », etc). Notons que la commande similaire « 354, 0, 6201, 6209, 6210, -1 » aurait lancé le clip « 6201 », puis le clip « 6209 », et enfin le « 6210 », le capteur devenant inactif après ce dernier clip.

```
DefSeq, 5, DoKmd, 326, 7, 350, 6208
```

```
DefSeq, 9, DoKmd, 326, 7, 328, 18, 353,
0, 6201, 6209, 6210, -1
```

Dans ce dernier exemple, l'activation du capteur « 3 » va exécuter les commandes « 325, 563 », choix de la configuration de HP n°563, « 352, 6251, 5 », jeu du clip 6251 en entier, à la vitesse 1, et enfin « 342, 3, 275 », verrouillage du capteur « 3 » pendant 27.5 secondes, durée de la lecture du clip – c'est-à-dire qu'il ne sera pas possible de le jouer à nouveau avant l'expiration de ce délai.

```
DefSeq, 3, DoKmd, 325, 563, 352, 6251, 5, 342, 3, 27.5
```

13.9.3 Notes

Dans une configuration à « N » capteurs, il est possible d'avoir des séquences associées à des numéros de « capteurs » de numéros supérieurs à « N ». Ces séquences peuvent être générées par un contrôleur MIDI envoyant les numéros de notes correspondantes, mais aussi déclenchées par programme. La commande 345 décrite ci-dessus permet de lancer une action au bout d'un temps donné, en simulant l'arrivée de l'événement MIDI correspondant. Dans l'exemple ci-dessous, on associe au capteur « 8 » un arrêt immédiat de tous les sons joués (« 304 »), puis on « programme » l'événement « 12 » au bout de 20 secondes (« 345, 12, 20 »). Cet événement, défini à la ligne suivante, va lancer le clip « 6670 », dont on ne joue (commande « 327, 10 ») que les dix premières secondes.

```
DefSeq, 8, DoKmd, 304, 345, 12, 20
DefSeq, 12, DoKmd, 327, 10, 350, 6670
```

13.9.4 Identification des commandes

Depuis la version 1.5.4, les commandes peuvent être désignées aussi bien par une valeur numérique que par un mot-clef. La description de ces commandes se trouve au chapitre 12 page 189.

13.10 Le fichier de configuration général

Par défaut, le Game Player va tenter de charger les fichiers de configuration décrits dans ce chapitre, puis, en dernier, un fichier « *profile* » permettant à l'utilisateur de définir divers autres aspects de ces préférences. Ce fichier a pour nom « GM_mSL_ini.mSL ». L'utilisateur peut, par exemple, choisir de redéfinir ici certains aspects définis dans les fichiers précédents. On peut par exemple essayer de nouveaux ensembles de clips avec des modes d'espace et de jeu différents, sans avoir

à modifier la base de la configuration définie par les fichiers précédents. Il suffit pour ce faire de placer les ajouts sous une forme telle que :

```

table(data(
  Table Clips
    ClVol 6 410 411 412
  End
  Table PlModes
    PMd 32
    Kmd KmdLoop
    Speed 0.4 0.7 0.8
    Gsize 0.2 0.45
    GDepth 2 3
  End
  Table Banks
    Grp 24
    Bnk 1 PlM 32 SpM 8 Clps 410 412
    Bnk 2 PlM 8 SpM 11 Clps 1201 1207
  End
));

```

Il est possible de la sorte d'ajouter, mais aussi de redéfinir des modes d'espace, des modes de jeu, des configurations de HP. Il suffira de supprimer ceci dans le profile pour revenir aux modes de jeu antérieurs.

On peut aussi choisir de n'utiliser que ce fichier « GM_mSL_ini.mSL », en y plaçant l'ensemble des définitions des tables nécessaires au fonctionnement du logiciel. Il suffit alors de ne pas définir les autres fichiers, ou encore d'indiquer que l'on ne désire pas charger ceux-ci.

Chapitre 14

Utilitaires

On a signalé que le Game Master utilisait essentiellement deux plug-ins principaux : le *Game Player* et le *File Player*. Un petit nombre d'autres plug-ins sont également disponibles, et s'avèrent pertinents lors de l'utilisation du logiciel.

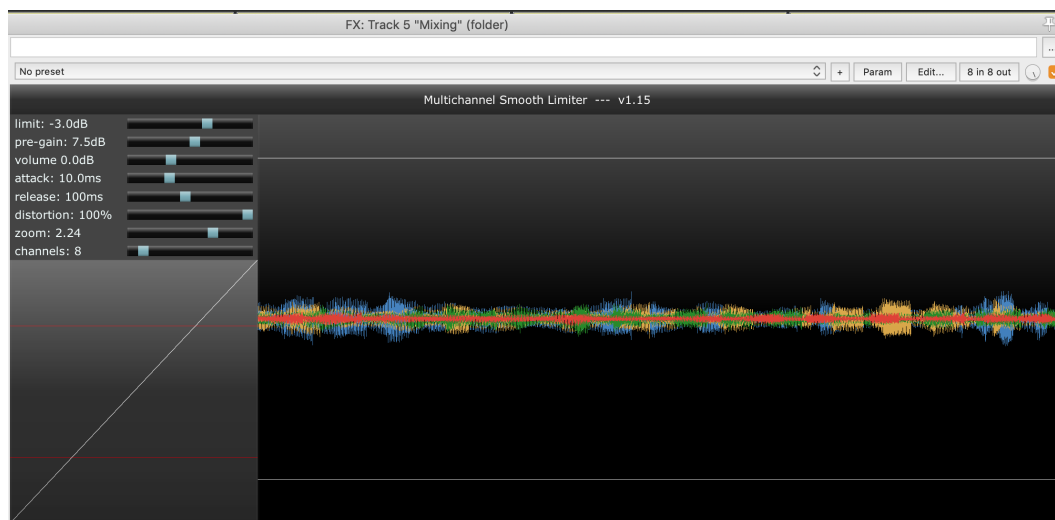


FIGURE 14.1 – Le compresseur multicanaux

14.1 Smooth Compressor

Il s'agit d'un compresseur multicanaux (C.f. fig. 14.1) dont l'utilisation est relativement simple :

limit ce réglage détermine la limite maximale du compresseur.

pre-gain permet de réduire ou booster le signal à l'entrée du compresseur.

- volume** permet d'augmenter le signal après compression. Le volume maximum en sortie sera donc (limit + volume)
- attack** durée de la rampe d'augmentation de compression à l'arrivée d'un signal supérieur à « limit ».
- release** durée de la rampe de diminution de la compression lorsque le signal d'entrée est devenu inférieur à la limite fixée.
- distortion** variable de 0 à 100%, ce réglage permet de doser une distortion douce appliquée au signal lorsque la compression devient effective
- zoom** utilisé uniquement pour l'affichage, détermine la hauteur du signal dans la fenêtre
- channels** choix du nombre de canaux en sortie. Seules les valeurs des quatre canaux les plus forts à un instant donné sont affichés en bleu, jaune, vert et rouge. Afin d'assurer une bonne visualisation des signaux, l'amplitude du signal le plus fort, affichée en bleu, est multipliée par quatre, celle du second plus fort, jaune, par trois, celle du troisième, vert, par deux, et celle du signal affiché en rouge est inchangée.

14.2 Multi Channel VU Meter

C'est l'afficheur standard multicanaux, livré avec les configurations de REAPER.

14.3 Cheap Channel Reducer

Sans aucun réglage, ce tout petit plug-in se contente de replier l'ensemble des canaux en entrée de la piste sur les canaux 1 et 2 en sortie, réalisant un « mixage » stéréo sommaire, les canaux 1, 3, 5, etc. étant dirigés vers la gauche, les 2, 4, 6, etc. vers la droite. Il remet également à zéro les sorties des autres canaux.

14.4 MIDIduino

Ce plug-in sert d'interface avec un Arduino connecté à des capteurs infra-rouge, et envoyant ses données sur sa sortie USB sous la forme de notes MIDI. Mais on peut aussi l'associer à n'importe quel clavier ou surface de contrôle MIDI. Sa particularité est de « replier » les numéros de notes MIDI reçues sur les valeurs 0 à 11 (do à si), ce qui permet de simuler un ensemble de 12 capteurs au moyen de n'importe quelle unité MIDI.

Chapitre 15

Beta testing

Ce chapitre est destiné plus spécifiquement aux quelques courageux qui acceptent de tester des versions du Game Master (GM) pas nécessairement stabilisées. Je travaille à l'heure actuelle sur un Macbook Pro de 2013 (prêt de JF Minjard, ma propre machine étant en panne), avec une CPU à processeur Intel. Le GM est donc assez sérieusement testé sur cete machine, mais peu/pas sur des machines Apple à processeur M1 ou M2, et encore moins sous windows. C'est dire si votre feed-back peut être utile ! N'oubliez pas, bien sûr, de conserver toutes les versions antérieures...

15.1 Outils de « mise au point » disponibles

Le GM est écrit en « eel2 », langage développé par Cockos, et principalement utilisé sous REAPER. Tous les plug-ins de type JSFX peuvent être « ouverts » durant leur fonctionnement : afficher la fenêtre d'interface graphique du plug-in, et cliquer sur le bouton « Edit... » en haut à droite. La fenêtre d'édition s'ouvre, affichant à gauche le code du plug-in (qui se compose en général d'un grand nombre de référence « include » à la bibliothèque « GM-Libs », et qui en fait ne nous intéresse pas), et à droite la liste des variables utilisées par le plug-in, dont les valeurs sont mises à jour en temps réel. Ces variables sont classées par ordre alphabétique, ce qui permet parfois, simplement en les consultant, de détecter des incohérences des programmes.

Utile également est le moniteur d'activité, obtenu par le menu « View ▸ Performance Meter ». Cocher toutes les options, choisir l'affichage « Display CPU utilization as 100% = all cores fully utilized », et afficher le tableau par « FX CPU » décroissant. Veiller à ce que toutes les pistes soient bien visibles.

Notez dans l'exemple de la figure 15.1 page suivante les informations relatives aux « RT longest-block » (bloc temps réel le plus long rencontré) et « RT xruns » (incapacité à fournir en sortie le signal audio temps réel). On a ici une valeur correcte, la durée du plus long bloc temps réel ayant été de 4.27ms, pour une valeur limite

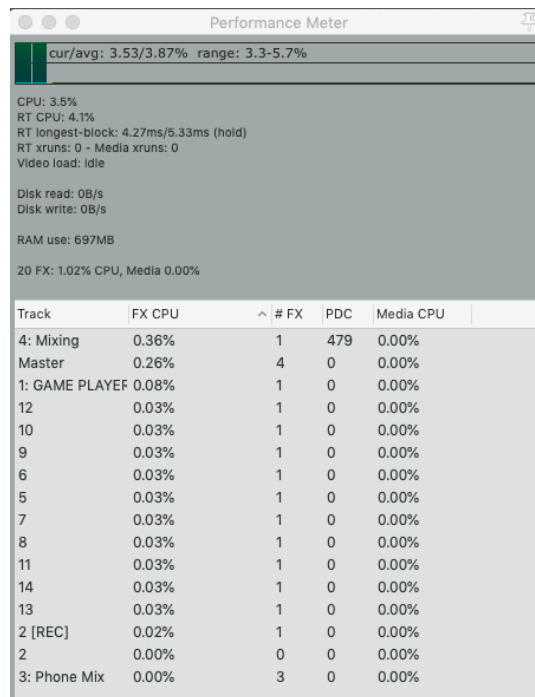


FIGURE 15.1 – Le Performance Meter

de 5.33ms, ce (court) délai correspondant à une taille de bloc de 256 échantillons choisi dans les préférences de REAPER. Au lancement d'une session GM, durant l'initialisation de ce dernier, il est « normal » que cette valeur soit dépassée (par exemple : « 47.48ms/5.33ms », et qu'un petit nombre de « xruns » se produisent. Il faut, juste après le lancement de la session, remettre ces valeurs à 0 en choisissant, dans le menu contextuel du Performance Meter, l'option « Reset graph ».

15.2 Modules spécialisés

Quelques-uns des modules GM sont plus spécifiquement destinés à « explorer » le GM en cours de fonctionnement. Le module « System Log » est souvent utilisé par le GM pour communiquer avec l'utilisateur, et les scripts mSL peuvent y écrire. Le module « Memory » (abordé 6.6 page 135) a pour finalité d'afficher des sections de la mémoire centrale ou de la mémoire partagée, sous différents formats.

Une addition récente (source code « mSL_Vars_Access.jsfx-inc » dans GM-Libs) permet d'accéder à près de 200 variables internes du GM, en lecture et pour certaines en écriture, au moyen de la fonction « get » :

```
v = get(`gm, `PWeight);
```


pour lire le contenu d'une variable JSFX (ici «PWeight»), et de la fonction «set» pour la modifier, lorsque cette modification est permise :

```
set(`gm, `PWeight, 3.75);
```

15.2.1 Players

Les players sont des plug-ins indépendants du GM, mais contrôlés par ce dernier, au travers d'informations déposées dans la mémoire partagée. Chaque player a un *numéro d'unité* (un nombre, compris entre 0 et 127) qu'il s'attribue au hasard au lancement d'une session GM, et qu'il modifie éventuellement jusqu'à ce qu'il n'y ait plus de conflit avec d'autres players. Un player est situé seul sur sa piste, et dispose donc d'un *numéro de piste* qui lui est propre. Enfin, pour l'affichage, le module «Sound Units» attribue à chaque player un *numéro de player* unique, compris entre 1 et *N*. Le numéro d'unité est utilisé pour attribuer au player un bloc de contrôle (270 valeurs environ) permettant la communication entre lui et le GM. L'adresse en mémoire partagée de ce bloc est dite «Unit Address».

Track 9 "Player"		Track 10 "Player"	
Track / Player / U.	9 / 27 / 72	Track / Player / U.	10 / 24 / 68
Status	Ready	Status	Busy
State	1 / 1	State	1 / 2
Gstate	0 577	Gstate	0 307
Err	0000 0000	Err	0000 0000
Unit Addr	20352	Unit Addr	19264
Activity / Check	294 / 0	Activity / Check	294 / 0
KmDs	292 : 16 0	KmDs	292 : 17 17
Gen	29 3 32	Gen	19 4 23
Rbase	2814794325	Rbase	3064722447

FIGURE 15.2 – Deux Players

La figure 15.2 montre les interfaces graphiques de deux players (dans une session qui en comporte 12), affichant des informations qui peuvent être pertinentes pour la mise au point :

Track / Player / U. De gauche à droite, le numéro de la piste REAPER, du player (celui-ci correspondant au numéro de player affiché dans «Sound Units»), et le numéro d'unité du player (0 à 127).

Status Cet indicateur affiche un mot-clef qui est habituellement «Ready» lorsque le plug-in est inactif, et prêt à répondre à une demande du GM, ou «Busy» lorsqu'il est en train de jouer.

State Cet indicateur rassemble deux valeurs, qui sont : l'activité (1 : actif) et l'état (1 : disponible, 2 : occupé).

Gstate Cet indicateur n'a une valeur différente de 0 que durant la lecture physique du fichier audio, ce qui est trop fugace pour être visible, sauf en cas d'erreur de lecture. La seconde valeur est un nombre premier, alloué au lecteur par le GM.

Err Dernière erreur détectée par le player. Une valeur égale à 0 est plutôt bon signe.

Unit Addr Adresse mémoire du bloc de contrôle correspondant au numéro d'unité du player. On peut utiliser cette valeur pour examiner le bloc avec le module « Memory ».

Activity / Check Test régulier (« ticks » à une fréquence de 0.06 Hz environ) des players, et résultat du test. Tous les players opérationnels doivent afficher le même numéro de test, et une réponse de 0 (occasionnellement 1).

Kmds Date (en « ticks ») de la dernière commande reçue, commande reçue (0 en tout début de session, 16 ou 17 pour Play/Loop, 5 pour Stop), numéro de commande (si en cours) ou « 0 » si terminée.

Gen Activité du générateur : nombre de « play », nombre de « loop », total des deux.

Rbase Seed courant du générateur de nombre aléatoires utilisé par ce player spécifique.

Track 9 "Player"				Track 10 "Player"			
Track / Player / U.	9 / -27 / 72	Unit Addr	20352	Track / Player / U.	10 / 24 / 68	Unit Addr	19264
Status	Unresponsive	Activity / Check	310 / 7	Status	Ready	Activity / Check	325 / 0
State	0 / 12	Kmds	308 : 16 0	State	1 / 1	Kmds	323 : 16 0
Gstate	0 577	Gen	33 5 38	Gstate	0 307	Gen	30 6 36
Err	0000 0000	Rbase	1861407209	Err	0000 0000	Rbase	3594355736

FIGURE 15.3 – Deux Players - B

La figure 15.3 montre les mêmes players, le premier ayant été désactivé quelques instants plus tôt (case d'activation décochée en haut à droite). Le plug-in n'étant plus actif, il est réputé « non accessible » (« Unresponsive »). Sa dernière activité (« Activity / Check ») remonte au test n°310, et il a été désactivé au bout de 7 tentatives de communication infructueuses. Son état « State » indique maintenant « 0 / 12 » (pour « désactivé / non accessible »). Dans ce cas précis, il suffit de cocher à nouveau en haut à droite pour qu'il redevienne immédiatement opérationnel.

Notons qu'un plug-in peut être désactivé parce que sa case d'activation a été décochée, parce que la piste sur laquelle il se trouve est en « mute », parce que l'ensemble des plug-ins de la piste ont été « bypassés » par le bouton « FX disable » de la piste... Ou encore parce que le plug-in ne peut pas s'exécuter du fait d'une erreur de programmation... Ce dernier cas est explicitement affiché dans la fenêtre du plug-in.

Remarque : c'est le plug-in qui affiche ces indications. Comment peut-il le faire lorsqu'il est désactivé ? C'est parce que l'on utilise l'option « `gfx_idle` » qui maintient active l'interface graphique du plug-in, même si sa partie audio est désactivée.

15.3 Et aussi

Si vous pensez que le fonctionnement du GM est incorrect, n'hésitez pas à me contacter par mail, après avoir tenté de qualifier cette erreur le mieux possible. Idéalement, vous proposez la démarche détaillée qui permet de la reproduire. Je ne suis intéressé que si cette erreur se produit, ou est reproductible, dans la dernière version distribuée. Signalez également tout problème avec le manuel de référence : description incomplète ou erronée, manque flagrant, etc. N'hésitez pas non plus à me faire part de vos envies d'évolution de la chose... Amicalement.

Chapitre 16

Installation ex-nihilo - v1.6.16

Ce chapitre décrit (au 31 Août 2023) la procédure utilisée pour créer une installation ex-nihilo du Game Master v1.6.1. Elle suppose une (petite) connaissance du logiciel REAPER, et une première lecture de ce manuel pour fixer les idées...

16.1 Projet autonome contenant le Game Master

Voici la liste des opérations à effectuer, à partir du moment où vous avez obtenu le répertoire « Game-Master », pour créer un projet nouveau contenant le logiciel (et donc autonome). Cette approche nécessite la version v6.82 de REAPER.

16.1.1 Création du projet lui-même

Imaginons que vous vouliez créer une session en octophonie, à partir d'un certain nombre de prises de sons de courtes durées (2' à 3' maximum).

1. Lancer REAPER. S'assurer qu'il n'y a pas de projet antérieur automatiquement lancé ; dans ce cas, le fermer, puis faire « File ▸ New project ». Si nécessaire, vérifier que vous avez bien, dans « Options ▸ Préférence... » la bonne carte son sélectionnée.
2. Afficher la piste « Master » par le menu « View ▸ Master track ». Celle-ci apparaît, en haut à gauche de la fenêtre REAPER.
3. Désactiver, toujours dans « View », le « Mixer » et le « Docker » que nous n'utiliserons pas.
4. Dans la piste « MASTER », cliquer sur le bouton « Route » et, dans la boîte de dialogue qui apparaît, choisir le nombre de pistes désirées (« track channels », en haut à droite du dialogue), par exemple 8 si vous allez travailler en octophonie. Dans ce même dialogue, régler « Audio Hardware Outputs »

pour que ces canaux soient envoyés vers les sorties appropriées de votre carte son. Fermer par la case de fermeture.

5. Choisir le menu REAPER « File▷Project settings... », ce qui ouvre une boîte de dialogue..
6. Dans la tabulation « Media », choisir pour « Path to save media files » : « Audio ». Ce nom permet au script auxiliaire de récupérer la localisation du projet. Faire « OK ».
7. Choisissez le menu « File▷Save project as... ». Dans le dialogue, cocher « Create subdirectory for project » (et éventuellement « Copy all media into project directory »). Choisir un nom (par exemple « GM-Test-1 ») et un emplacement pour votre projet, et cliquer « Save ».
8. Quitter REAPER.
9. Allez dans le répertoire de votre projet (disons toujours « GM-Test-1 »). Il doit contenir un répertoire vide, de nom « Audio », et un fichier de nom « GM-Test-1.RPP » qui est la description créée par REAPER de votre projet.
10. Créez dans ce même répertoire un nouveau répertoire de nom « Data », un répertoire de nom « Effects », un répertoire de nom « presets » et un répertoire de nom « Scripts ». Respecter la casse de ces orthographes.
11. Dans le répertoire de nom « Effects », placer une *copie* du répertoire « Game-Master » de la distribution.
12. Dans le répertoire de nom « Scripts », placer une *copie* du fichier « GMAuxiliary.eel » de la distribution.
13. Vous pouvez maintenant relancer REAPER, qui va ouvrir automatiquement votre projet « GM-Test-1 ».
14. Choisir le menu REAPER « Track▷Insert new track ». Cette commande crée la piste « 1 », en-dessous de la piste MASTER.
15. Eventuellement, changer le nom de cette piste (double clic à droite du cercle rouge, indicateur « Record Arm/Disarm ») en « Game Player ».
16. Cliquer sur la case « FX ». Apparaît un dialogue, avec un choix de plug-ins. A gauche, dans « All Plugins », cliquer « JS ». A droite, vous devez trouver, entre autres, « JS:<Project>/Game-Master/GamePlayer.jsfx ». Choisir celui-ci, cliquer « Add » en bas à droite. Si tout se passe bien la (toute petite) fenêtre du GamePlayer s'affiche, vide. A noter que tous les plug-ins référencés par la suite seront ceux qui sont situés dans ce répertoire, et dont le nom débutera par « JS:<Project>/Game-Master/...jsfx ».
17. Dans cette fenêtre maintenant affichée, en dessous de la barre des tabulations («Main», «Studio A», etc.), faire un shift-clic-droit pour afficher le

menu de configuration des panneaux. Choisir « Modules » Sound Units ». Le module s'affiche, sous forme d'une ligne de 8 cases grises : il n'y a encore aucun module son...

18. Cliquer sur le bouton « Route » de la piste « Game Player ». Choisir dans « track channels » 2 canaux, et décocher (haut, gauche), « Master send channels ».
19. Choisir le menu REAPER « Track » Insert new track ». Cette commande crée la piste « 2 », en-dessous de la piste précédente. Donnons lui le nom de « Mixer ». C'est elle qui va recevoir tous les sons générés par les différents players.
20. Cliquer sur « Route » de la piste « Mixer ». Choisir « Track channels » identique à la valeur de la piste MASTER, 8 dans le cas présent.
21. Cliquer sur la case « FX » de la piste « Mixer ». Choisir le plug-in « JS : <Project>/Game-Master/SmoothMultLimiter.jsfx ». Dans celui-ci, régler « channels » à la valeur appropriée, ici 8.
22. Choisir le menu REAPER « Track » Insert new track ». Cette commande crée la piste « 3 », en-dessous de la piste précédente. Donnons lui le nom de « Player ». Ce sera le prototype de tous les players.
23. Cliquer sur « Route » de la piste « Player ». Choisir « Track channels » identique à la valeur de la piste MASTER, 8 dans le cas présent. Dans ce même dialogue, cliquer sur le menu « Add new receive... », et choisir « 1 : Game Player ». Dans le mode de réception associé, choisir « Pre-Fader (Post-FX) ». Grâce à ceci, le player va pouvoir recevoir, déguisées en signaux audio, certaines informations venant du Game Master.
24. Cliquer sur la case « FX » de la piste « Player ». Choisir « JS : <Project>/Game-Master/File ». Dans la fenêtre du Game-Master, on doit voir apparaître un premier player, dans la première case, sous la forme « Player 1 » (c.f. fig. 16.1).

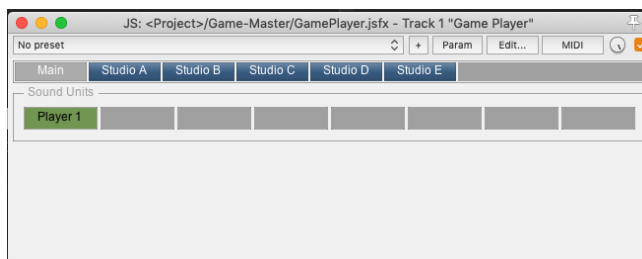
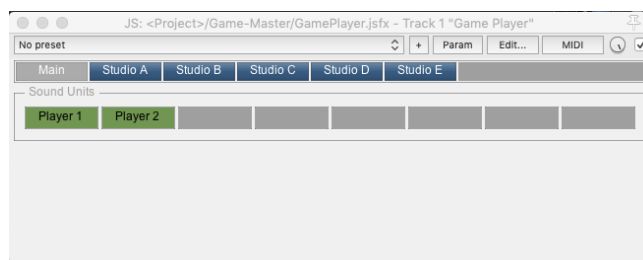


FIGURE 16.1 – Premier player *Game Master*

25. Dans la piste 2, tout à gauche, sous le chiffre « 2 », cliquer sur la petite icône à peine visible. Au premier clic, on constate que la piste 3 se place en retrait : elle est désormais « fille » de la piste 2, laquelle va recevoir tous les sons émis par la piste 3.
26. Faire un clic-droit sur la piste 3, et choisir « Duplicate tracks ». REAPER crée une copie de cette piste 3, de même nom, et de numéro 4. On voit alors apparaître un second player dans la fenêtre du Game Master (fig. 16.2).

FIGURE 16.2 – Second player *Game Master*

27. Répéter cette opération autant de fois que nécessaire, jusqu'à obtenir le nombre voulu de players. Cette valeur correspondra à la polyphonie désirée. On peut aller jusqu'à 40 ou 50 players sans gros problèmes de CPU. Si vous voulez réduire ce nombre, il suffit de supprimer des pistes. Notons que dans ce cas les players sont bien sûr immédiatement supprimés, mais que le Game Master tente de les reconnecter durant deux minutes environ, avant de les retirer de sa liste dans « *Sound Units* ».
28. Sauvegarder le projet, qui est désormais prêt à accueillir vos sons.
29. Il reste à installer le script auxiliaire qui permet d'offrir au GM certaines possibilités supplémentaires. Il est nécessaire pour ceci d'avoir installé les extensions « SWS/S&M ». Chercher ce sigle sur internet, qui vous conduit à « www.sws-extensions.org ». Ce site est considéré comme absolument sûr. Télécharger et installer la version appropriée à votre machine. Relancer REAPER après cette installation.
30. Vérifiez que votre projet contenant le Game Master est bien actif. Choisir le menu « Actions » Show actions list », ce qui ouvre le dialogue des actions.
31. En bas, à droite, choisir le pop-up menu « New action » Load ReaScript... »
32. Dans la fenêtre de dialogue qui s'ouvre, se déplacer jusqu'au répertoire « Scripts » de votre projet, et choisir « GMAuxiliary.eel ».

33. Le script apparaît dès lors dans la liste des action (en général, tout en haut de celle-ci), sous la forme « Script : GMAuxiliary.eel », suivi d'un code du style « _RS92e8a687... ».
34. Faire un clic-droit sur cette ligne, et choisir dans le menu « Copy selected action command ID ».
35. Dans « filter », tout en haut, taper « startup », ce qui n'affiche plus que les actions contenant ce mot.
36. Choisir par un clic-gauche « SWS/S&M : Set project startup action », puis cliquer « Run » tout en bas de la fenêtre.
37. Dans le dialogue « Set project startup action », coller l'identification du script « GMAuxiliary.eel » obtenue lors d'une opération antérieure. Faire « OK ».
38. Fermer la fenêtre des actions : « Close », en bas à droite.
39. *Sauver le projet !*

Il vous est alors loisible de quitter REAPER, puis de le relancer. Votre projet se recharge, et vous pouvez vérifier que le menu « Actions » comporte désormais la ligne supplémentaire « Running script : GMAuxiliary.eel ».

16.1.2 Décisions, décisions...

Il faut maintenant décrire la configuration de votre installation, choisir les sons que vous désirez utiliser, et de quelles manières ils vont être mis en oeuvre. Je suggère de vous intéresser en premier à votre installation sonore, que vous ne modifiez probablement pas tous les jours, et dont la description sera réutilisable d'une installation à l'autre.

16.1.3 Description de l'installation

Ce qui est dit des canaux et des haut-parleurs, dans la section 1.6.2 page 28 est bien sûr toujours d'actualité. On considère toujours que chaque HP est relié à un canal différent de la carte sons (donc 8 HP pour les 8 canaux).

Votre configuration une fois décrite dans le fichier « config-HP.mSL » et chargée dans le GM pourra être vérifiée grâce au module « HP Configuration » (section 2.11 page 64). Il n'y a guère plus à en dire que ce qui est expliqué à la section 13.3 page 200.

Il est difficile de donner de bons conseils, tant il peut y avoir de variantes pour une installation, même de 8 haut-parleurs : des différentes formes d'octos circulaires, de la ligne droite au cube, les possibilités sont multiples. Il est bon de prévoir des sorties mono et stéréo (quand l'auditeur est au centre du cercle), des stéréo proches ou lointaines (lorsque les HP sont sur scène), et des tutti (tous les HP...).

Considérons d'abord une octo circulaire, avec des HP de n° 1 à 8 dans le sens des aiguilles d'une montre. On peut imaginer la configuration suivante¹ :

```
table(data(
  Table HPConf Clear
    101 1 1 0   102 1 2 0   103 1 3 0   104 1 4 0
    105 1 5 0   106 1 6 0   107 1 7 0   108 1 8 0
    201 2 1 2 0   202 2 2 3 0   203 2 3 4 0   204 2 4 5 0
    205 2 5 6 0   206 2 6 7 0   207 2 7 8 0   208 2 8 1 0
    401 4 1 2 3 4 0   402 4 3 4 5 6 0
    403 4 5 6 7 8 0   404 4 7 8 1 2 0
    801 8 1 2 3 4 5 6 7 8 0
  End ));
```

Ici le groupe 100 (numéros de 101 à 108) désigne chaque HP individuellement, le groupe 200 les paires de HP consécutifs, 400 des ensembles de 4 HP (nord, est, sud, ouest, par exemple), et 800 la configuration pour les tutti.

Si au contraire nous nous intéressons à un cube (4 HP au sol, circulairement 1 à 4, et 4 HP en hauteur, de 5 à 8, le 5 étant placé au-dessus du 1, le 6 au-dessus du 2, etc.), on peut choisir :

```
table(data(
  Table HPConf Clear
    101 1 1 0   102 1 2 0   103 1 3 0   104 1 4 0
    105 1 5 0   106 1 6 0   107 1 7 0   108 1 8 0
    201 2 1 7 0   202 2 2 8 0   203 2 3 5 0   204 2 4 6 0
    401 4 1 2 3 4 0   402 4 5 6 7 8 0
    801 8 1 2 3 4 5 6 7 8 0
  End ));
```

Le groupe 100 (de 101 à 108) désigne comme précédemment chaque HP individuellement, 800 le tutti ; nous avons choisi les paires de HP les plus éloignées pour les groupes de 2 HP de 200, et 400 représente les deux configurations de 4 HP au sol pour la première, 4 HP en hauteur pour la seconde.

Souvenons nous qu'il est possible d'utiliser, par exemple 203 pour cette paire spécifique, et 200 pour l'une quelconque des paires de HP.

16.1.4 Création des clips...

L'aspect suivant du travail consiste à effectuer le choix, au milieu de l'immense bibliothèque de sons accumulés depuis des années, voire des décennies, des clips que

1. On se souvient qu'une configuration est définie par un numéro de votre choix, suivi d'un nombre de HP, suivi de la liste de ces HP, et se termine par un 0.

vous avez l'intention d'utiliser. Là, je ne peux rien pour vous.

Une fois les clips choisis, il faut les déclarer au Game Master.

Une première approche consiste à créer explicitement un répertoire, contenant l'ensemble des clips que vous désirez utiliser. Cette approche est plus complexe, mais offre le grand intérêt de vous permettre de créer un projet autonome, qu'il est possible de transférer sur une autre machine, lors de la mise en place d'une installation, par exemple.

Une seconde approche permet de faire du « cherry picking » de vos clips, et de les rendre accessibles au Game Master par un simple glisser-déposer. Le projet obtenu sera opérationnel sur votre installation, mais ne sera pas facilement exportable sur une autre machine.

16.1.4.1 Création d'un répertoire spécial pour le projet

La contrainte supplémentaire que vous impose le Game Master est qu'il ne sait utiliser, jusqu'à présent, que des clips dont le nom est de la forme « clip1XXXX.SSS », où « XXXX » est un nombre de 4 chiffres exactement, compris entre « 0000 » et « 9999 », et « SSS » est le suffixe qui indique le format du clip, par exemple « wav », « aiff », « mp3 », « flac », etc. Une fois que vous avez décidé d'utiliser un fichier représentant un son particulier, par exemple « cri du cormoran au-dessus des jonques.wav », il ne vous reste plus qu'à le recopier dans le répertoire « WAVES », lui donner le nom sous lequel vous allez l'utiliser, par exemple « clip12720.wav », et vous souvenir de la correspondance.

Je vais vous décrire une approche qui va permettre de se simplifier la vie, mais...

Mais sur Mac... La longue digression qui suit s'adresse aux utilisateurs de Macintosh ou de Linux, dans la mesure où vous disposez en natif d'une application de type *Terminal*, et d'un *shell*² (*sh*, *bash*, etc.) relativement classique³. Vous aurez besoin de :

- savoir un peu utiliser *TextEdit*, le simplissime éditeur de texte du Mac, ou équivalent.
- savoir recopier sans erreur quelques lignes de code assez absconses
- comprendre mes explications, même lorsqu'il y a beaucoup d'implicite. Par exemple, je n'écirai nulle part « *l'ordinateur doit être en route pour que ce*

2. Un *shell* – il en existe en quantité – est un interprète des commandes introduites au terminal. Il permet aussi d'écrire, à l'instar de tout langage de programmation, des programmes complets, dits *scripts*, destinés à exécuter une fonction particulière.

3. Mais avec un peu de travail supplémentaire, vous pouvez, même sous Windows, installer un *shell* opérationnel. Cherchez sur Internet une information telle que « Utiliser le terminal *bash* sous Windows 11 » ou équivalent pour découvrir la marche à suivre.

que vous tapez au clavier soit pris en compte », ni plein d'autres choses du même style.

Sur Mac, ma première suggestion sera de placer les icônes des utilitaires « *TextEdit* » et « *Terminal* » dans le « *Dock* » afin qu'ils soient aisément accessibles. Dans la suite, nous interviendrons en interaction tantôt avec l'application *Terminal*, tantôt avec *TextEdit*. Je tenterai de mon mieux d'indiquer quelles commandes doivent être exécutées immédiatement au terminal, et quelles autres doivent être introduites dans le fichier en cours d'édition dans l'éditeur de texte.

Pour lancer l'application « *Terminal* », il suffit donc de cliquer sur l'icône de l'application. Une fenêtre apparaît, avec un « *prompt*⁴ » tel que « `~ Jonz$` » qui indique que vous pouvez introduire une commande. Si vous faites « *return* », le même prompt réapparaît à la ligne suivante. Dans les paragraphes qui suivent, on suppose que les commandes sont tapées en réponse à un tel prompt.

Modifications de votre « profil » Il n'est pas inutile d'opérer quelques modifications dans votre profil utilisateur de terminal pour vous simplifier la vie. Lancer « *Terminal* » comme expliqué ci-dessus si ce n'est pas déjà fait. Assurez-vous d'être dans votre répertoire d'accueil par « `pwd` » ou « `cd` ».

A savoir La commande « `pwd` » (« *print working directory* ») liste dans votre terminal le *nom* du répertoire courant. Sur un Mac, le répertoire d'accueil de l'utilisateur « *Jonz* » est « `/Users/Jonz` ».

A savoir La commande « `cd` » (« *change directory* ») utilisée *sans paramètre* vous ramène dans votre répertoire d'accueil. Avec un paramètre, elle permet des déplacements relatifs dans l'organisation arborescente des répertoires d'un disque : « `cd toto` » descend dans le sous-répertoire « `toto` » du répertoire courant. « `cd ..` » remonte au répertoire parent. Elle permet aussi des déplacements absolus : « `cd /Users/Jonz/Library/Support/REAPER` » va dans le répertoire personnel des ressources de REAPER de l'utilisateur *Jonz*. Remarquons qu'Apple a eu la bonne idée de placer un espace dans le nom du répertoire « `Application Support` » ce qui fait que dans le shell, il faut placer un « *backslash* » devant cet espace pour indiquer qu'il fait partie du nom du fichier, et n'est pas un séparateur de commandes...

Dans votre répertoire d'accueil, faire « `open .profile` » pour ouvrir le fichier décrivant votre profil utilisateur. Si vous avez droit à la réponse « `The file /Users/Jonz/.profile does not exist.` », faites juste « `touch .profile` » pour créer ce fichier, que vous pouvez ensuite ouvrir par « `open .profile` ». Si un autre message d'erreur apparaît (tel que « aucune application n'est définie pour

4. Ici, « *Jonz* » est l'indentification de l'utilisateur ; le prompt sera probablement différent sur votre machine.

ouvrir ce fichier »), essayez « `open -e .profile` », qui suggère explicitement d'utiliser `TextEdit` pour ouvrir le fichier.

Pour ma part, je trouve pratique :

- de disposer de quelques commandes personnelles pour aller directement dans un répertoire fréquemment utilisé. Ainsi, pour se rendre dans le répertoire des ressources de REAPER, je tape directement « `RR` ».
- de disposer de variables du shell qui désignent directement un répertoire spécifique.
- de disposer d'un emplacement personnel pour placer les nouvelles commandes que l'on écrit sous forme de shell script, afin qu'elles soient accessibles en tout lieu et à tout instant.

Créons donc une commande pour aller dans le répertoire des ressources de REAPER. Dans un terminal (on peut en avoir autant que l'on veut, par le menu « `Shell` » Nouvelle fenêtre... »), Ouvrons le profil (par « `cd` » puis « `open .profile` »), plaçons nous dans la fenêtre ouverte par *TextEdit*, et rajoutons la ligne suivante⁵ :

```
alias RR="cd '/Users/Jonz/Library/
Application Support/REAPER' "
```

Dès lors (très précisément, à partir du moment où l'on a sauvegardé le fichier, et dans tout nouveau terminal ouvert après cet instant), la commande « `RR` » vous permet de vous positionner immédiatement dans ce répertoire. Notons que toute la partie droite de l'affectation est pacée entre double quotes, et que le nom du répertoire lui-même est entre simples quotes : cette précaution permet de s'assurer que tout ce qui est entre simple quotes fait partie intégrante du nom du fichier, y compris le fameux espace séparant « `Application` » de « `Support` ».

Il peut être pratique de confirmer visuellement que l'on est bien dans ce répertoire, par exemple en imprimant son nom. On définira alors « `RR` » sous la forme de la ligne unique (même remarque que ci-dessus) suivante :

```
alias RR="cd '/Users/Jonz/Library/
Application Support/REAPER' ; pwd"
```

L'alias que nous créons se compose ici de deux commandes séparées par un point-virgule. Elle seront exécutées en séquence, de gauche à droite.

Il peut être également pratique d'imprimer immédiatement le contenu du répertoire, ne serait-ce que pour se souvenir de l'endroit où l'on désirait aller après nous être rendus dans ce répertoire. On peut créer « `RR` » ainsi :

```
alias RR="cd '/Users/Jonz/Library/
Application Support/REAPER' ; ls"
```

5. Ligne unique, malheureusement présentée ici sous la forme de deux lignes consécutives pour que le manuel soit lisible – le caractère « `A` » du début de seconde ligne doit suivre immédiatement, sans espace, le caractère « `/` » de fin de ligne précédente.

A savoir La commande « `ls` » (« *list* ») affiche sur le terminal la liste, ordonnée alphabétiquement en une ou plusieurs colonnes selon la largeur disponible, des fichiers contenus dans le répertoire courant. Cette commande peut aussi accepter nombre d'options dont certaines des plus utiles sont :

`ls -l` affiche au format long, une entrée par ligne, les noms, tailles, dates de modification, droit d'accès, etc. de chaque fichier.

`ls -t` affiche les noms des fichiers, du plus récemment modifié au plus ancien. En combinant « `l` », « `t` », et « `r` » (qui veut dire en ordre inverse), « `ls -ltr` » affiche les fichiers au format long, le plus récent en dernier.

A savoir De manière plus générale, il peut être utile de consulter le manuel de la commande, que l'on obtient par « `man ls` ». Naturellement, il existe des descriptions de toutes les commandes du shell : « `man cd` », « `man pwd` », etc.

Enfin et pourquoi pas, vous pouvez imprimer votre message personnel au moyen de la commande « `echo` » par exemple :

```
echo Dans le répertoire REAPER
```

Nous avons évoqué les « variables du shell ». Ce terme concerne un petit nombre de variables, propres à chaque incarnation d'un shell (donc liées à une fenêtre terminal particulière). La liste de ces variables, avec leur valeur, s'obtient par « `env` ».

```
$ env
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash
TERM=xterm-256color
TERM_PROGRAM_VERSION=421.2
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
PWD=/Users/Jonz/Library/Application Support/REAPER
HOME=/Users/Jonz
LOGNAME=Jonz
```

(Quelques variables non intéressantes ici ont été supprimées de cette liste).

Ces variables peuvent s'utiliser dans un script sous la forme d'un « `$` » suivi du nom de la variable :

```
echo Le répertoire principal de $LOGNAME est $HOME
```

S'il peut exister une ambiguïté dans la suite de caractères placée après un « `$` », on peut utiliser des accolades autour du nom de la variable :

```
echo Le répertoire principal de ${LOGNAME} est ${HOME}
```

Il est possible de créer des variables propres à un processus (ce que l'on va faire dans de petits scripts), ou des variables d'environnement, qui seront connues de tous les processus héritant de cet environnement. Ainsi, dans les écritures suivantes :

```
toto=hello
export titi=world
```

La variable « `toto` » est propre au processus courant, mais « `titi` », dont la déclaration est précédée de « `export` », sera utilisable par des descendants du processus, et apparaîtra dans le résultat de « `env` ». Décidons que nous avons besoin d'une variable qui désigne le répertoire d'accès aux ressources de REAPER. Nous pouvons écrire dans notre « `.profile` » :

```
export RR='/Users/Jonz/Library/Application Support/REAPER'
```

puis, ultérieurement, utiliser :

```
ls "$RR/Effects"
```

pour lister les effets de type JSFX définis dans notre version de REAPER ⁶.

Enfin, pour conclure sur l'utilisation du profil, remarquons la variable « `PATH` » qui décrit une liste de répertoires, séparés par des caractères « `:` », où le *shell* peut trouver des programmes exécutables. Il est utile, plutôt que de modifier les répertoires du système, de rajouter un répertoire personnel dans lequel nous rangerons nos propres programmes. Il est pratique d'appeler « `bin` » ce répertoire, et de le créer dans notre répertoire d'accueil. Dans une fenêtre terminal, écrire :

```
cd ; mkdir bin
```

On va maintenant indiquer qu'il contient des exécutables, en ajoutant dans la fenêtre d'édition du « `profil` » :

```
export PATH=.:$HOME/bin:$PATH
```

A partir de maintenant, le shell acceptera des scripts ou des exécutables situés dans le répertoire courant « `.` », dans votre répertoire « `bin` » (par « `$HOME/bin` »), ou encore dans tous les répertoires antérieurement définis par défaut (« `/usr/bin` », « `/bin` », etc.).

6. Remarquons que les double-quotes autour du paramètre de « `ls` » sont nécessaires pour compenser l'effet nocif de l'espace inclus dans « `Application Support` ». Mais on peut aussi écrire : « `ls "$RR"/Effects` ».

Le concept Continuons nos explications, et abordons la notion de « lien ». Un système de fichiers décrit l'organisation logique d'un disque dur (ou d'une partie de celui-ci). L'espace adressable d'un disque dur est partagé en un certain nombre de « secteurs » de taille fixe (par exemple 256 octets, 1024 octets, etc.). Tout fichier est représenté, selon sa taille, par un ou plusieurs secteurs. Les fichiers sont, quant à eux, organisés sous forme arborescente : la « racine » est un répertoire, qui référence des fichiers ou d'autres répertoires, chacun d'eux pouvant référencer des fichiers ou des répertoires, etc. Un répertoire n'est lui-même rien d'autre qu'un fichier particulier, qui définit des *liens* entre un « nom de fichier » et « l'adresse disque » de ce fichier. Un tel lien n'a de sens qu'au sein d'un certain système de fichier. Sur un autre disque, un même numéro de secteur va désigner un autre fichier.

Un nom de fichier peut constituer une référence « *relative* », à l'intérieur du répertoire courant, tel « mouche-noire.aiff », ou un nom « *absolu* », relatif au système de fichier, « /Users/Jonz/Sons/Insectes/mouche-noire.aiff ».

Une commande du shell, « *ln* » (pour « *link* »), permet de créer un nouveau lien (ici « clip17021.wav ») sur un fichier existant :

```
ln criquet-sahara-7.wav clip17021.wav
```

La commande ne crée pas une copie : simplement, on dispose dorénavant de *deux noms différents* pour référencer le *même* fichier, et l'accès à ce fichier est tout aussi performant, quel que soit le nom utilisé.

L'intérêt, bien sûr, est que ce nouveau nom peut être placé (ou déplacé) dans un répertoire différent du répertoire original. Ceci va nous permettre, sans utiliser d'espace supplémentaire sur le disque, de laisser nos fichiers sons originaux dans leur hiérarchie propre, mais aussi d'en fournir l'accès au Game Master selon son propre protocole, à l'intérieur du répertoire WAVES.

Notons encore que, tant que l'on reste dans le même système de fichiers, « déplacer » un fichier d'un répertoire à un autre, que ce soit par une commande shell ou une action à la souris, ne fait que déplacer ce lien, sans bouger le fichier qui reste au même emplacement du disque. C'est donc une action extrêmement rapide.

A savoir Comparons les résultats de trois commandes assez semblables :

```
ln file1 file2
cp file1 file2
mv file1 file2
```

Après la commande « *ln* », que l'on vient de voir, il existe un fichier unique sur disque, l'original « *file1* », maintenant accessible par les deux noms « *file1* » et « *file2* ». Comme il s'agit du *même* fichier, toute modification de « *file1* » sera visible dans « *file2* » et réciproquement.

Après la commande « cp » (« *copy* »), il existe un nouveau fichier, « file2 », qui est en tous points identique à « file1 ». Cependant, de l'espace a été utilisé sur le disque pour représenter ce nouveau fichier, et ils sont physiquement différents. Une modification de l'un n'a pas d'impact sur l'autre.

Enfin, « mv » (« *move* ») va simplement changer le nom du fichier original, de « file1 » en « file2 ».

Dans toutes ces commandes, « file1 » ou « file2 » peuvent être des références relatives ou absolues. On peut ainsi déplacer ou copier un fichier d'un répertoire à un autre, éventuellement en changeant son nom, etc.

Lorsque l'on s'intéresse à des fichiers situés dans des systèmes de fichiers différents (par exemple, le disque système monté sur la racine « / », et un disque contenant essentiellement des fichiers audios, monté par exemple sur « /Volumes/Sons »), il n'est pas possible de créer sur l'un des disques un lien direct sur un fichier situé sur l'autre disque. On va alors créer un *lien symbolique*. Imaginons que notre terminal soit positionné sur un répertoire situé sur le disque système. On pourrait alors écrire :

```
ln -s /Volumes/Sons/Baleine-33.wav clip12003.wav
```

afin de créer, dans ce répertoire, un lien (symbolique, par l'option « -s ») vers un certain fichier se trouvant sur l'autre disque. Cette approche permet donc d'utiliser, sans les recopier, des fichiers situés sur d'autres volumes que le disque système. Elle est un peu plus lente, puisqu'il faut d'abord décoder le lien symbolique, puis ouvrir effectivement le fichier référencé. Elle est également plus fragile, en ce sens qu'un fichier ainsi référencé peut ne pas être en ligne (disque non monté, par exemple).

Après cette brève introduction à certains aspects du shell, revenons à notre préoccupation initiale, et voyons en quoi ces nouvelles connaissances vont nous aider à mieux gérer nos clips.

Imaginons un scénario raisonnable : vous allez choisir des clips dans différents répertoires, les classer selon certains critères, et créer une dizaine de groupes de clips, dont vous avez déjà déterminé la finalité.

La méthode proposée consiste à constituer des listes de clips représentant ces groupes, puis à utiliser un utilitaire qui générera les commandes de création des noms de clips attendus par le *Game Master*. Il est facile de construire une telle liste : placez-vous dans le répertoire « X » contenant vos sons, et générez la liste de tous les fichiers de type « wave » dans le fichier « listeA.txt » par :

```
cd X ; ls *.wav > liste.txt
```

L'expression « *.wav » va filtrer tous les noms de fichiers qui se terminent par « .wav », suffixe caractéristique des fichiers au format « wave ». Le caractère « > » qui suit indique une redirection : le résultat ne va pas sortir sur le terminal, mais dans le fichier de nom « liste.txt » – le suffixe « .txt » indique un format textuel.

Vérifiez le contenu de ce fichier, en le listant sur le terminal par :

```
cat list.txt
```

Il vous est maintenant loisible d'éditer ce fichier, pour éliminer les sons qui ne vous intéressent pas, éventuellement d'en changer l'ordre, pour grouper des sons aux caractéristiques similaires, etc. Vous allez effectuer ce travail dans *TextEdit*, en ouvrant le fichier par :

```
open -e liste.txt
```

A savoir L'expression « *.wav » va sélectionner tous les fichiers « wave » du répertoire courant. Vous pouvez sélectionner en une seule commande plusieurs types de fichiers :

```
ls *.wav *.flac *.aiff
```

Vous pouvez aussi sélectionner les fichiers dans d'autres répertoires, au moyen de chemins relatifs ou absolus :

```
ls */*.wav  
ls /Volumes/Sons/Bruits/*.wav /Volumes/Sons/Modulations/*.wav
```

La première expression va filtrer tous les « waves » situés dans des sous-répertoires du répertoire courant, la seconde va chercher les « waves » dans deux répertoires spécifiques du disque « Sons ». Dans le premier cas, vous obtiendrez des noms de fichiers relatifs au répertoire courant, dans le second des noms absolus.

Nous allons maintenant utiliser certains scripts *shell* distribués avec le Game-Master, et placés dans le répertoire *Scripts*. Allez dans ce répertoire, et copier dans votre répertoire « bin » créé ci-dessus (c.f. § 16.1.4.1 page 239) tous les fichiers de type « .sh », qui sont des commandes destinées au shell.

sndfiles.sh Ce premier script « emballe » une séquence de commandes, qui liste les fichiers du répertoire courant et des sous-répertoires, filtre ceux dont le suffixe correspond à un type de fichier son reconnu (« wav », « aif », « flac », etc.), et ajoute en tête leur position dans la hiérarchie du système de fichier courant. Le script est directement utilisable par son nom dans la mesure où il est placé dans votre répertoire « bin ». Deux utilisations typiques :

```
sndfiles.sh | more
```

va nous permettre de lister page par page cette liste de noms de fichier, en utilisant l'opération « | » qui transmet directement la sortie du script à l'entrée de l'utilitaire « more » : on passe à la page suivante par un espace, on en sort par « q ».

```
sndfiles.sh > liste.txt
```

qui cette fois écrit, en utilisant la redirection « > » la liste des sons dans le fichier « `liste.txt` ».

Il est dès lors possible d'éditer cette liste, (par « `open -e liste.txt` ») pour ne conserver que les sons qui vous intéressent, éventuellement en les déplaçant dans le fichier, avant de créer une série de clips destinés à être inclus dans le répertoire « WAVES ».

A savoir Sans devoir passer par un éditeur de texte, vous pouvez visualiser le contenu d'un fichier texte par la commande « `more` » ci-dessus, ou encore « `cat` » qui imprime tout le contenu du fichier sur votre terminal :

```
more liste.txt
cat liste.txt
```

creatclips.sh Cet autre script a pour tâche de créer une série de noms adaptés au Game Master à partir de cette liste de clips. Il attend trois paramètres : le premier est le nom d'un fichier, contenant la liste originale des clips, à raison d'un nom par ligne; le second paramètre est le numéro du premier clip cible (par exemple 2340, pour créer les clips 2340, 2341, 2342, etc.); le troisième paramètre est un mot-clef qui indique de quelle manière se fait cette génération :

copy signifie que les clips originaux vont être recopiés, c'est-à-dire dupliqués, ce qui occupera autant d'espace disque supplémentaire que les clips initiaux.

move indique que les clips originaux vont être déplacés dans le répertoire courant, sous ces nouveaux noms. Il n'y a pas d'espace disque utilisé en plus, mais les fichiers originaux auront disparu de leur répertoire.

link demande qu'un nouveau lien physique soit créé sur le fichier d'origine. Il n'y a pas d'espace disque utilisé en plus, le nouveau nom et l'ancien faisant référence au même fichier physique. Ce lien physique ne peut être créé que si le fichier source et le fichier cible sont sur le même disque physique.

syml indique qu'un lien symbolique va être créé sur le fichier d'origine. Il n'y a que quelques octets de plus utilisés pour chaque fichier, et les clips peuvent faire référence à des fichiers sur n'importe quel disque monté. En contrepartie, tous les fichiers devront être sur des systèmes de fichiers montés pour que le GM puisse les utiliser.

list se contente de vous indiquer quelle sera la correspondance entre les noms originaux et les noms transformés, sans rien modifier. Cette option peut être utile pour créer après coup la liste des correspondances, si vous disposez toujours du fichier texte utilisé comme premier paramètre.

A titre d'exemple, après avoir créé dans un répertoire le fichier « `liste.txt` », nous pouvons créer les fichiers « clips » attendus par le GM :

```

Jonz$ creatclips.sh liste.txt 4500 symb
clip14500.wav /Users/Jonz/SONS/DIEGO-2022/Audio 1-SndShGS_09.wa
clip14501.wav /Users/Jonz/SONS/Serie-1/JJG-STE-002.wav
clip14502.wav /Users/Jonz/SONS/DIEGO-2022/Audio 1-XNsStr_21.wav
clip14503.wav /Users/Jonz/SONS/Serie-1/Click-03.wav
clip14504.wav /Users/Jonz/SONS/DIEGO-2022/Audio 1_03.wav
clip14505.aif /Users/Jonz/SONS/MarcAntoineMillon/tôle C-Ducer#0
clip14506.wav /Users/Jonz/SONS/Serie-1/Chocs-aigus-forts.wav
clip14507.aif /Users/Jonz/SONS/MarcAntoineMillon/tôle pavillon#
clip14508.wav /Users/Jonz/SONS/Serie-1/JJG-STE-006B.wav
clip14509.wav /Users/Jonz/SONS/REMY/Bim_13.wav
clip14510.aif /Users/Jonz/SONS/Serie-1/Arrival-01-S.aif
clip14511.wav /Users/Jonz/SONS/Serie-1/Bip-repetitif.wav
clip14512.wav /Users/Jonz/SONS/Serie-1/Bips-repetitif-4.wav
clip14513.wav /Users/Jonz/SONS/Serie-1/Carillon-bruits.wav
clip14514.wav /Users/Jonz/SONS/Serie-1/Chocs-verre-metal.wav
clip14515.wav /Users/Jonz/SONS/Serie-1/Click-01.wav
clip14516.wav /Users/Jonz/SONS/DIEGO-2022/Audio 1_02.wav
clip14517.wav /Users/Jonz/SONS/Serie-1/JJG-STE-005B.wav
clip14518.wav /Users/Jonz/SONS/REMY/Bim_26.wav
clip14519.wav /Users/Jonz/SONS/Serie-1/Machine-en-fonctionnemen
Jonz$

```

Le script nous imprime d'une part la correspondance entre les fichiers créés et les noms originaux – et, bien sûr, nous crée tous ces nouveaux fichiers. Les voici :

```

Jonz$ ls
clip14500.wav clip14507.wav clip14514.wav
clip14501.wav clip14508.aif clip14515.wav
clip14502.wav clip14509.wav clip14516.wav
clip14503.wav clip14510.wav clip14517.wav
clip14504.aif clip14511.wav clip14518.wav
clip14505.aif clip14512.wav clip14519.wav
clip14506.wav clip14513.wav liste.txt
Jonz$

```

Notons qu'au lieu de laisser la liste des correspondances s'imprimer sur le terminal, on peut l'écrire dans un fichier texte - par exemple « set-4500.txt » :

```
creatclips.sh liste.txt 4500 symb > set-4500.txt
```

Conservons précieusement tous ces fichiers indiquant des correspondances. On peut les consulter (par « cat », « more »), les imprimer pour en garder la trace, ou même

les trier :

```
sort set-4500.txt
```

va trier le tableau sur les numéros de clips.

```
sort -k 2 set-4500.txt
```

va trier le tableau selon la seconde colonne, c'est-à-dire les noms des fichiers.

En répétant ces opérations sur d'autres fichiers, il est possible de créer d'autres familles de clips : 4600, 4210, 4300, etc. Il est pratique de conserver l'ensemble de ces fichiers de correspondances dans un fichier unique, par exemple :

```
cat set-4500.txt set-4600.txt set-4210.txt set-4300.txt > all-sets.txt
```

Puis d'imprimer les résultats de «`sort all-sets.txt`» et «`sort -k 2 all-sets.txt`» pour disposer des correspondances dans les deux sens pendant vos travaux de composition.

Dans tous les cas, on déplacera les noms ainsi créés dans l'emplacement où le GM les attend, par exemple :

```
mv clip1* "MyProject/Data/WAVES"
```

En cas d'erreur de manipulation, si vous avez utilisé l'option «`copy`», «`link`» ou «`symb`», les originaux ont été conservés, et vous pouvez tout simplement supprimer les clips que vous venez de créer, par :

```
rm clip1*
```

Si par contre vous avez utilisé l'option «`move`», c'est plus ennuyeux, car les clips originaux ont disparu...

En résumé, ce processus n'est pas excessivement complexe, et simplifie grandement la tâche de manipulation des clips à destination du Game Master.

16.1.4.2 Seconde approche : le glisser-déposer

Cette opération, disponible depuis la version 1.7, est beaucoup plus simple à mettre en œuvre,

16.1.5 Autres fichiers de configuration

On imagine maintenant que vous avez créé vos différents groupes de clips, avec une première idée de l'utilisation que vous allez en faire. Supposons que vos choix soient les suivants :

- clips en 6700 : ils sont courts, vont bien se prêter à des transpositions, des passages à l'envers. Ils vont être joués dans leur intégralité. Ils seront plutôt ponctuels (en mono/stéréo).
- clips en 6800 : on peut s'en servir pour des boucles de moyennes durées : quelques secondes, boucles qui vont se répéter en s'étirant jusqu'à une minute. On peut envisager une diffusion plus large, par exemple 4 à 8 HP.
- clips en 6900 : ils ont des micro textures intéressantes, et vont être utilisés pour faire de la granulation, avec des grains de 50ms à 500ms. Là également, on choisira une diffusion mono ou stéréo.

Il est dès lors possible d'écrire une première version de certaines configurations de jeu et d'espace.

```
table(data(
  Table PlModes Clear
    PMd 1, Kmd KmdPlay, PVol -9 0
    PMd 2, Kmd KmdPlay, PVol -9 0, Speed 0.5 1.5 0.5
    PMd 3, Kmd KmdLoop, PVol -2 -4, Speed 0.5 0.8,
      Gsize 5 50, GDepth 3
    PMd 4, Kmd KmdLoop, PVol -4 -8, Speed 0.5 1.8 0.5,
      Gsize 0.05 0.50, GDepth 6
  End
));
```

Les modes de jeu 1 et 2 proposent des variations de volume, et pour le second, des variations de vitesses. Les boucles du mode de jeu 3 auront des durées de 5 à 50 secondes, celles du groupe 4 des durées de 50 à 500 ms.

```
table(data(
  Table SpModes Clear
    SMd 1, HPSet 100
    SMd 2, HPSet 200
    SMd 4, HPSet 400
    SMd 7, HPSet 800, HPCnt 5 7
    SMd 8, HPSet 800, HPCnt 8
  End
));
```

Les modes d'espace reprennent ici les configurations définies pour les HP. On a choisi les configurations « génériques », 100, 200, etc, si bien qu'à chaque utilisation une configuration spécifique tirée au hasard sera utilisée. On a ajouté un mode 7, qui utilisera entre 5 et 7 HP parmi ceux de la configuration 800.

Il reste à faire des choix pour les banques, dont les partiels vont associer des numéros de clips à des modes d'espace et de jeu.

```

table(data(
  Table Banks Clear
  Grp 1
    Bnk, 1, Clps 6700 6799, SpM 1, PlM 1
    Bnk, 1, Clps 6700 6799, SpM 1, PlM 2
    Bnk, 1, Clps 6700 6799, SpM 2, PlM 1
    Bnk, 1, Clps 6700 6799, SpM 2, PlM 2
    Bnk, 2, Clps 6800 6899, SpM 4, PlM 3
    Bnk, 2, Clps 6800 6899, SpM 7, PlM 3
    Bnk, 2, Clps 6800 6899, SpM 8, PlM 3
    Bnk, 3, Clps 6900 6999, SpM 1, PlM 4
    Bnk, 3, Clps 6900 6999, SpM 2, PlM 4
  End
));

```

On a donc défini trois banques (au moyen de 9 partiels) qui recouvrent les trois classes de clips définies ci-dessus.

Nous disposons donc a priori de tout le nécessaire pour effectuer nos premières expérimentations – qui nous conduiront probablement à modifier certains de ces fichiers selon ce que l’expérience nous apprendra...

Chapitre 17

Erreurs détectées par le Game Master

Le plug-in propose de nombreuses fonctions, et tente d'effectuer un maximum de vérifications avant d'opérer ses actions. Pour cette raison, on peut se trouver en présence de nombre d'erreurs différentes dans les fichiers de configuration, ou dans les scripts. Voici une liste non nécessairement exhaustive des erreurs qui risquent d'être signalées. Notons qu'une grande partie de ces erreurs ne s'est jamais présentée lors des tests du logiciel. Dans certains cas, l'erreur est rendue sous forme d'un nombre négatif : -404 correspond alors à l'erreur 404.

0001 à 0255 Erreur de compilation dans un script. [exec].

0257, 0258 Erreur dans une liste d'actions

0260 Table des macro-définitions saturée

0261 Action inconnue

0262 Macro-definition, symbole non trouvé

0263 Action inconnue

0310 Le fichier de configuration des HP ne peut être ouvert

0311 Numéro de configuration trop élevé

0312 Nombre de HP incorrect

0313 Numéro de configuration de HP incorrect

0314 Numéro de configuration de HP incorrect

0315 Numéro de HP incorrect

0316 Zéro final absent

0317 Configuration déjà définie

0319 Table des configurations de HP pleine

0320 Le fichier de configuration des modes de Jeux ne peut être ouvert

250 *CHAPITRE 17. ERREURS DÉTECTÉES PAR LE GAME MASTER*

0321	Erreur de syntaxe dans le fichier des modes de Jeux
0322	Numéro de version du fichier des modes de Jeux incorrect
0326	Mot-clef attendu
0327	Nombre attendu
0328	Mot-clef attendu
0329	Paramètre invalide
0330	Le fichier de configuration des partiels ne peut être ouvert
0331	Erreur de syntaxe dans le fichiers des modes d'espace
0333	Numéro de version du fichier des modes d'espace incorrect
0334	Le fichier de configuration des modes d'espace ne peut être ouvert
0336	Un mot clef est attendu
0337	Mode d'espace invalide
0338	Un mot clef est attendu
0339	Trop de paramètres après un mot-clef
0342	Numéro de clip incorrect
0343	Numéro de version incorrect
0344	Numéro de groupe invalide
0345	Numéro de banque invalide
0346	Numéro de clip invalide.
0348	Numéro de mode de jeu invalide
0349	Numéro de mode d'espace invalide
0350	Valeur de volume minimum invalide
0351	Valeur de volume maximum invalide
0352	« Poids » du partiel invalide
0353	Valeur des flags invalide
0354	Table des partiels pleine
0356	Valeur inattendue
0357	Paramètres trop nombreux
0404	Fichier audio non trouvé
0421	Fichier audio probablement endommagé
0448	Fichier audio probablement endommagé
0468	Fichier audio probablement endommagé

0500	Erreur dans une commande de lecture de clip
0504	Partiel incorrect
0506, 0507	Mode de jeu incorrect
0508, 0509	Mode d'espace incorrect
0511	Aucun player audio disponible
0513	Commande « loop » incorrecte
0515	Commande « play » incorrecte
0516	Commande inconnue
0518	Clip non trouvé
0524	Partiel incorrect
0526, 0527	Mode de jeu incorrect
0528, 0529	Mode d'espace incorrect
0604	Fichier clip incorrect
0601	Erreur de syntaxe dans le fichier clip
0606	« CIVol » attendu
0607, 0608	Numéro de clip invalide
0609, 0610	Intervalle de clips incorrect
0613	Paramètre de « Fident » invalide
0614	Répertoire de clips invalide
0615	Format de clip invalide
0600-0999	Erreur dans « table ».
1001	Compiler structure error
1002	Compiler table overflow
1003	Compiler table overflow
1010	Erreur dans « mSL_make_constants_array »
1016	Erreur dans « mSL_make_thread »
1100	Appel de « malloc » incorrect
1104	Trop de paramètres dans un « sprintf ».
1105	Appel de « set » incorrect.
1271 à 1272	Unable to allocate a memory bloc for internal compiler needs
1601	Pas de bloc de contrôle de thread disponible
1602	Trop de tâches actives

1619	Débordement de la pile
1620 à 1622	Une fonction est attendue [exec].
1901	Instruction illégale : 0 [exec].
1980	invalid evaluator
1983	Erreur dans un appel de « mSL_set_compiler »
1999	Erreur dans un paramètre de « compile »
2011	Chain structure broken
2201	Erreur de syntaxe
2202	Erreur de syntaxe
2206	Syntax error in operator
2207	Operand missing
2220	Erreur de syntaxe
2221 à 2223	Chaining error
2224	Bracket missing
2225	Invalid brackets nesting
2226 à 2234	Erreur de syntaxe dans les instructions « var », « loc » ou « function ».
2235	Error in « import » construct.
2251 à 2255	« function » construct error
2301	Identifier too long
2302	Lexical error
2331	Syntax error in parenthesized expression
2332	Bracket syntax error
2333	Missing identifier in indexing
2334	« gmem » construct error
2335	Syntax error in function call
2336, 2337	Syntax error in « while » construct
2338	Syntax error in parenthesized expression
2341	Unknown unit in expression
2404	Le fichier sélectionné n'a pas pu être ouvert.
2405	Le script sélectionné n'a pas pu être ouvert.
2407	Le paramètre de « mkthread » n'est pas une fonction.
2408	Impossible de créer un nouveau thread dans « mkthread ».

2409	Une erreur dans un script a empêché sa compilation.
2412	incorrect scheduler entry
2413	incorrect code
2414	incorrect thread
2416	incorrect code
2417	no defined stack
2418	no defined global table
2419	no defined pointer table
2420	no defined own table
3001	marker lost
3301	Error in « import » construct.
3333 à 3335	Undefined operation
3401 à 3403	Invalid expression
3404	Invalid indexing construct
3411	Invalid « loop » construct
3412 à 3414	Invalid expression in « data » construct
3416 à 3419	Undeclared identifier
3555 à 3556	Invalid function call
4001	Internal table underflow
4002	Internal table overflow
4401 à 4404	Error in function definition
7273	Play log saturé
9602	Erreur dans « mSL_load_thread »
9603	Erreur dans « mSL_save_thread »
9606	Pas de thread disponible
9900 et suivantes	Erreurs internes. Ne devraient pas survenir dans les versions distribuées.

Quatrième partie

Annexes

Chapitre 18

Glossaire

Cette partie tente d'expliquer certains concepts implicitement utilisés dans ce document, mais non formellement décrits. Elle est organisée sous forme alphabétique.

Flag

Un *flag* est un indicateur dit « booléen » ou « logique », ne prenant que deux valeurs différentes, « 0 » ou « 1 », ou encore « faux » ou « vrai », « off » ou « on », « non positionné » ou « positionné », etc. Un *flag* peut, par exemple, représenter l'état d'un interrupteur commandant une ampoule électrique : « on » et « off » correspondent alors à « allumé » et « éteint ».

Un *flag* est parfois représenté seul dans une variable (celle-ci va alors avoir la valeur « 0 » ou « 1 »), mais ils sont en fait souvent groupés pour constituer un nombre entier. Celui-ci est alors calculé comme la somme (en réalité, mais c'est dans ce cas équivalent, le « ou » logique) des valeurs suivantes :

- premier flag (0 ou 1) multiplié par 1 (2 puissance 0), soit 0 ou 1 (décimal), ou encore 0x0 ou 0x1 (hexadécimal)
- deuxième flag (0 ou 1) multiplié par 2 (2 puissance 1), soit 0 ou 2 (décimal), ou encore 0x0 ou 0x2 (hexadécimal)
- troisième flag (0 ou 1) multiplié par 4 (2 puissance 2), soit 0 ou 4 (décimal), ou encore 0x0 ou 0x4 (hexadécimal)
- quatrième flag (...) multiplié par 8 (2 puissance 3), soit 0 ou 8 (décimal), ou encore 0x0 ou 0x8 (hexadécimal)
- cinquième flag (...) multiplié par 16 (2 puissance 4), soit 0 ou 16 (décimal), ou encore 0x00 ou 0x10 (hexadécimal)
- sixième flag (...) multiplié par 32 (2 puissance 5), soit 0 ou 32 (décimal), ou encore 0x00 ou 0x20 (hexadécimal)
- ...

- vingt-huitième flag (...) multiplié par 134217728 (2 puissance 27), soit 0 ou 134217728 (décimal), ou encore 0x00000000 ou 0x08000000 (hexadécimal)
- vingt-neuvième flag (...) multiplié par 268435456 (2 puissance 28), soit 0 ou 268435456 (décimal), ou encore 0x00000000 ou 0x10000000 (hexadécimal)
- trentième flag (...) multiplié par 536870912 (2 puissance 29), soit 0 ou 536870912 (décimal), ou encore 0x00000000 ou 0x20000000 (hexadécimal)
- trente et unième flag (...) multiplié par 1073741824 (2 puissance 30), soit 0 ou 1073741824 (décimal), ou encore 0x00000000 ou 0x40000000 (hexadécimal)
- trente-deuxième flag (...) multiplié par 2147483648 (2 puissance 31), soit 0 ou 2147483648 (décimal), ou encore 0x00000000 ou 0x80000000 (hexadécimal)

En pratique, les entiers étant souvent représentés sur 32 bits, la dernière des valeurs ci-dessus correspond alors à un nombre négatif, ce qui complique la manipulation des flags dans cette représentation. On se limite toujours à 31 flags par nombre.

Comme on le voit, un nombre représentant une suite de flags, dont seuls les flags 1, 2 et 5 sont positionnés, peut s'écrire $1+2+16$ soit 19 en décimal, 0x13 en hexadécimal. Pour des valeurs élevées du numéro de flag, on préfère la notation hexadécimale, beaucoup plus facile à interpréter visuellement. Ainsi, lorsque les flags 1, 2, 5, 28 et 31 sont positionnés, le nombre s'écrit 0x48000013, mais 1207959571 en décimal, ce qui est vraiment peu parlant.

Dans une opération « data », les valeurs exprimées en décimal ou hexadécimal sont acceptées, ce qui fait que ces deux notations (0x48000013 ou 1207959571) fournissent naturellement la même valeur. L'opération « data » accepte aussi une opération « | » de calcul du « ou » logique de deux constantes. On peut alors trouver dans un script une notation telle que « HPFlgs 1|2|16 » ou encore « HPFlgs 0x1|0x2|0x10 » qui dispense de devoir effectuer manuellement le calcul de ces valeurs. Notons encore qu'il existe une notation « binaire » qui permet de représenter les bits individuels d'un nombre (en l'occurrence chaque flag), et qui pour les flags 1, 2 et 5 se noterait « 0b10011 » (les flags se positionnant donc de droite à gauche).

Bibliographie

- [1] Audio production without limits. <http://www.reaper.fm/>, 2023.
- [2] REAPER User Guide. <http://www.reaper.fm/userguide.php>, 2023.
- [3] Cockos. Site JS. <http://reaper.fm/sdk/js/js.php>, 2017.
- [4] Cockos. Wiki JS. http://wiki.cockos.com/wiki/index.php/Jesusonic_Effects_Documentation, 2017.
- [5] Cockos. Reaper strings functions. https://reaper.fm/sdk/js/strings.php#js_string_funcs/, 2023.
- [6] Pierre Couprie. Le vocabulaire de l'objet sonore. In *Du sonore au musical*, pages 24–32. L'Harmattan, 2001.

