

Un exercice de programmation...

Vintage...

...et un outil d'aide à la composition
électroacoustique

Jean-Jacques Girardot & Jean-François Minjard

25 Mai 2022

Le projet « Game Master »

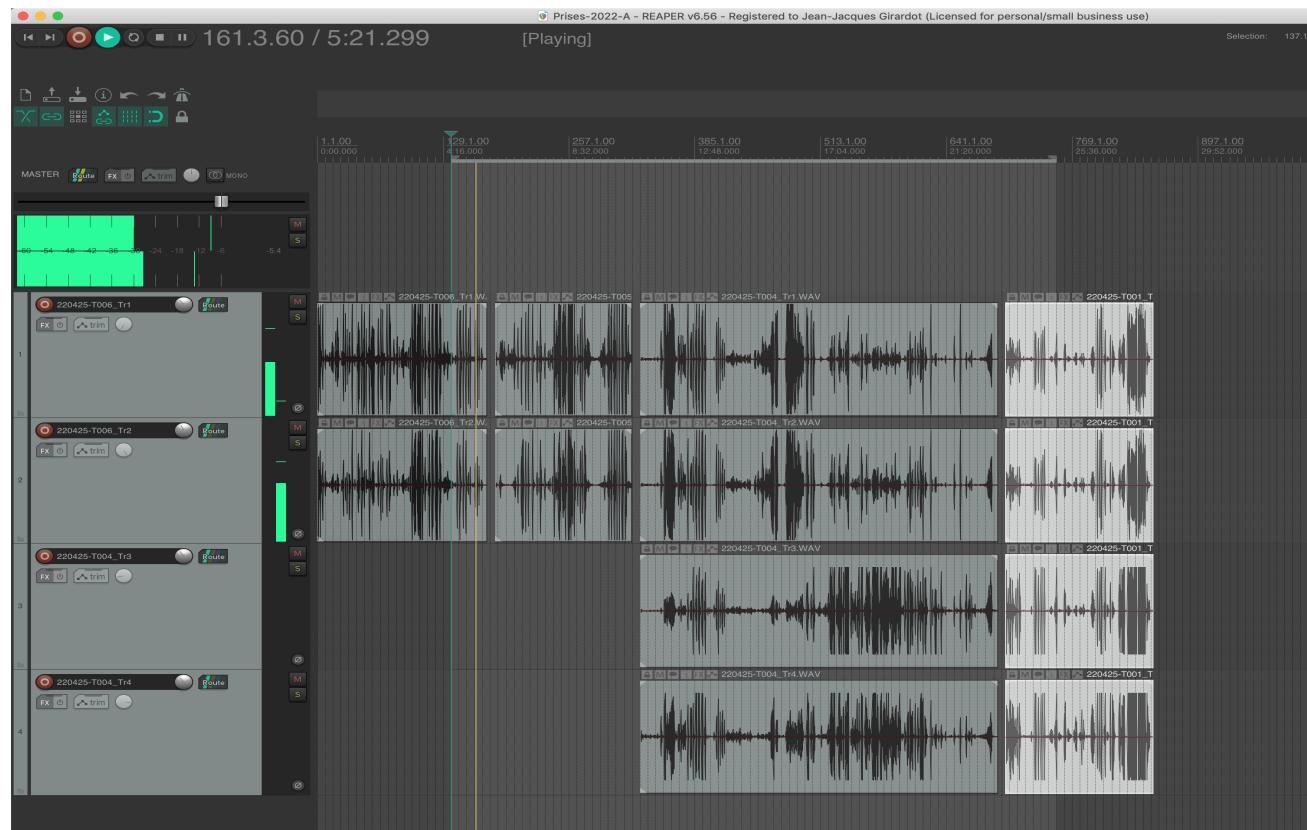
- Animer une installation électroacoustique
 - Jouer des « clips », petites séquences sonores
 - Utilisation d'un « **DAW** »
 - Choix de **REAPER**
 - Pistes son, avec les clips répartis sur la piste
 - Permettre l'interaction avec les « visiteurs »
 - Utilisation de capteurs
 - Piézos branchés sur des entrées de la carte son
 - Réactivité ?
 - Première version : un **plugin** (écrit en **JS**) lit la piste, conserve le son, le restitue lors de l'interaction

Un DAW ?

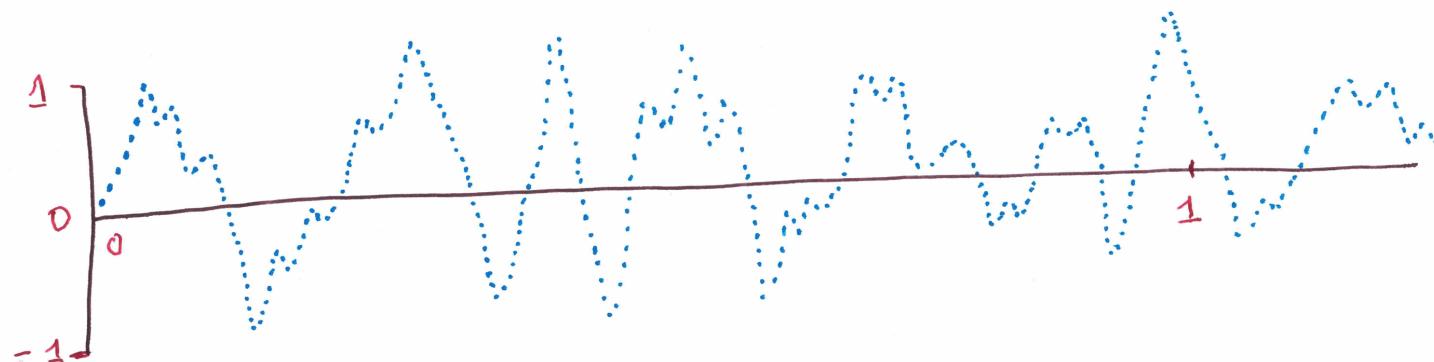
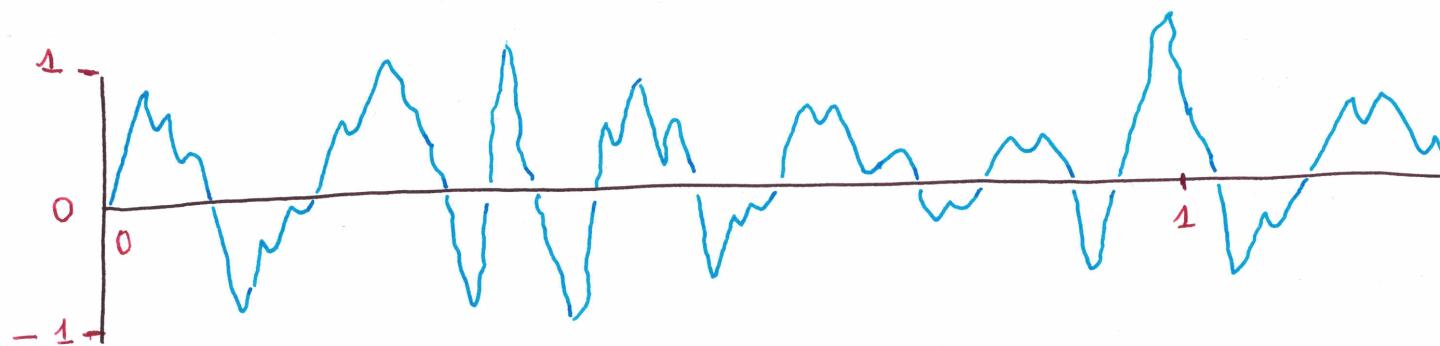
- Des « pistes » au sens d'un magnétophone sur lesquelles on place des fichiers audio
- Les pistes sont mixées vers une piste « master » qui peut être enregistrée ou dirigée vers une carte son
- Le « moteur » du DAW assure l'ensemble des traitements et des synchronisations
- Les pistes et fichiers peuvent recevoir des traitements audio relevant du « DSP », confiés à des « plug-ins » créés par des sociétés spécialisées
- Types de plug-ins standard : VST, AU
- Plug-ins propres à des DAW spécifiques : AAX, JSFX...

L'outil

- REAPER : un «bon daw»
 - Puissant, économique, efficace, outil de création de scripts intégré, « vrai » multicanal



Le son digital



- Une seconde : 44100/48000/96000 samples
- Si l'on a 25 pistes octophoniques, à 48kHz, avec un « calcul » par sample, on a droit à 0.1 microseconde par calcul...

Poursuite du projet...

- Un premier besoin :
 - Créer des variations dans ces sons
 - Répétition des clips : uniformité, ennui...
 - Introduire du hasard dans le choix des clips
 - ... et dans la manière dont ils sont joués
 - Vitesse, volume, transposition
- Ecriture d'un script (sous forme de plugin) :
 - Lecture des clips depuis le disque
 - Variation de vitesse/hauteur à la manière d'un magnétophone

Le langage de script de REAPER

- Ecriture de plugins traditionnels : C, C++, etc.
- Ou : utilisation de JS, propre à REAPER
 - Utilisé pour la première version de l'installation
 - Des avantages et des inconvénients...
- JS : « Jesus Sonic ».
 - Syntaxe de type « C simplifié »
 - Source, compilateur JIT, code machine non optimisé
 - Fournit « math.h », « string.h », quelques opérations de « DSP » et des fonctions graphiques

Caractéristiques de JS

- Syntaxe de type C, les () remplacent les {}
- Constructions manquantes : switch/case, goto, for, macroprocesseur
- Type de données unique : le flottant double 64 bits
- Pas de déclaration de variables, identificateurs case-insensitive
- Fonctions non récursives, pas de référence avant possible
- Pas de bibliothèques autres que les fonctions de math.h, string.h, fft, convolution, quelques fonctions graphiques
- Accès à un tableau de 32M, propre à chaque plugin JSFX
- Accès à un tableau de 8M, partagé entre tous les plugins
- Accès à un nombre limité de chaînes de caractères
- Accès extérieurs filtrés : lecture dans certains répertoires, lecture et écriture contrôlée de présets, pas d'écriture de fichiers.
- Excellente intégration à REAPER

Caractéristiques de JS

- Un plugin JSFX reçoit des tuples d'échantillons (valeurs 64 bits float) à transformer.
- Les JSFX comportent différentes sections
 - `@init` initialisation, une fois... ou plus.
 - `@slider` en cas d'interaction avec l'interface graphique par défaut
 - `@serialize` sauvegarde et restauration de l'état du plugin
 - `@block` une fois pour chaque bloc de tuples
 - `@sample` une fois pour chaque tuple
 - `@gfx` période de rafraîchissement de l'interface graphique
- REAPER gère les différentes tâches, leur synchronisation, et assure la compatibilité des plugins JSFX au travers des différentes versions du logiciel et des systèmes

Poursuite du projet...

- Lire des fichiers son : opération bien intégrée
 - Comment les désigner ?
 - Associer un numéro (0 à 9999) à chaque clip.
 - Utiliser des fichiers de noms tels « clip1635.wav »
- Créer des « variations » dans le jeu
 - Pas d'utilisation d'effets sonores
 - Modes de jeu : jouer une partie du clip, à des vitesses différentes, voir négatives, jouer des « boucles », courts extraits de 10ms à 30s
 - Modes d'espace, utilisant la multiphonie
- Découverte progressive des capacités du GM

Puis...

- Multiplication des paramètres de jeu
 - Clip, durée, volume, vitesse, canaux, boucles...
- Associer à chaque son les paramètres adéquats
 - Modifications appropriées
- Hiérarchisation, par création de
 - « groupes » ensembles cohérent
 - « banques » sous-classe des groupes, formées de
 - « partiels » type de jeu/espace associé à des ensembles de clips

Descriptions...

- Nécessité d'une syntaxe, par exemple :

Grp , 200

Bnk , 30 , Clps , 2300 , 2304 , SpM , 14 ,
PlM , 24 , PWeight , 0.5 , prVol , -3

Bnk , 30 , Clps , 2305 , 2310 , SpM , 14 ,
PlM , 25 , PWeight , 0.16

- Autres besoins :

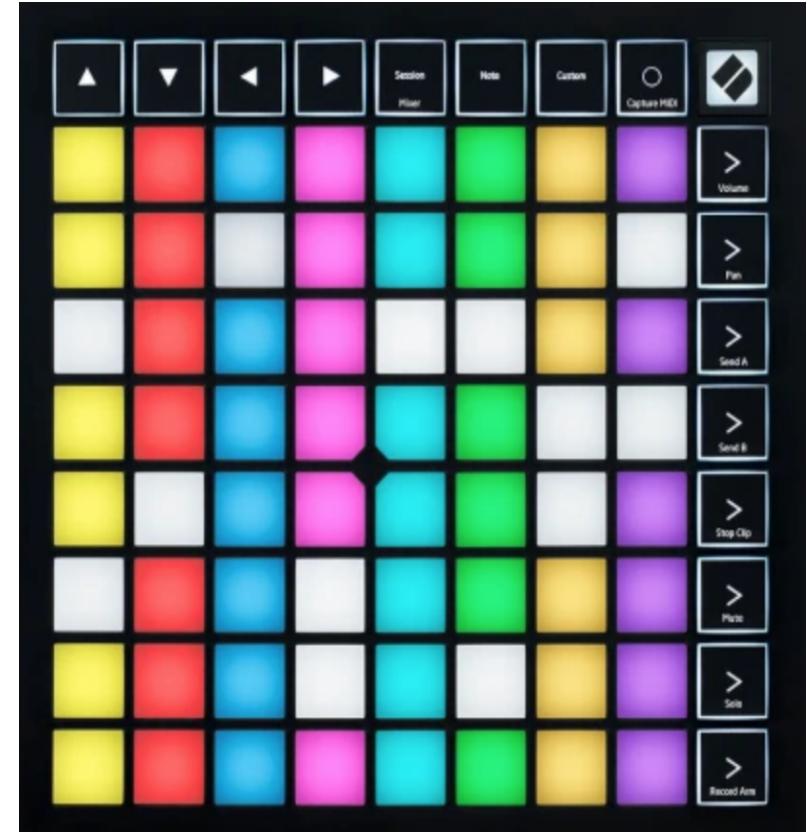
- configurations des HP, modes de jeu, modes d'espace, caractéristiques des clips, etc...
- description des interactions

Eléments d'interactions



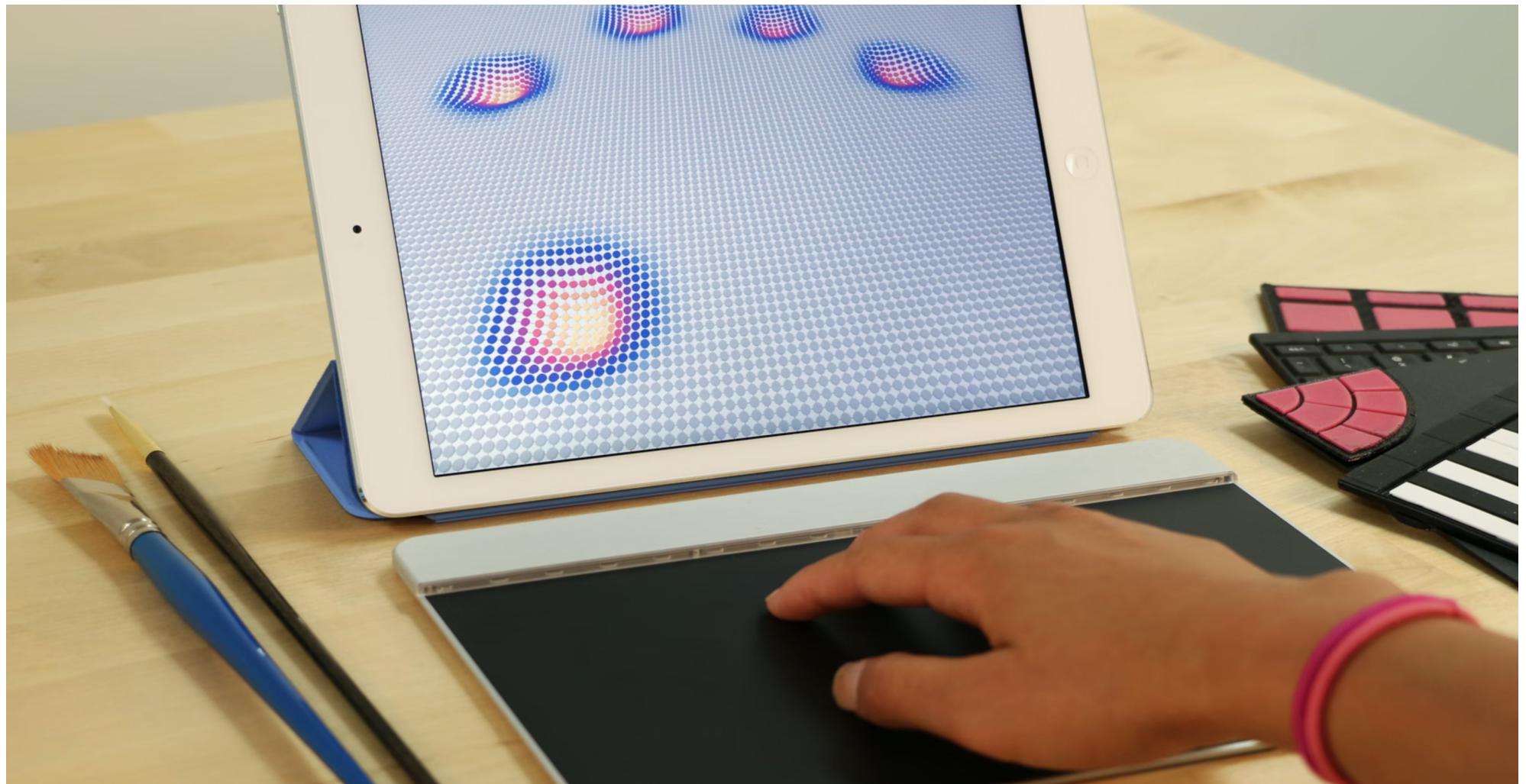
Surfaces MIDI

Novation
LaunchPad X



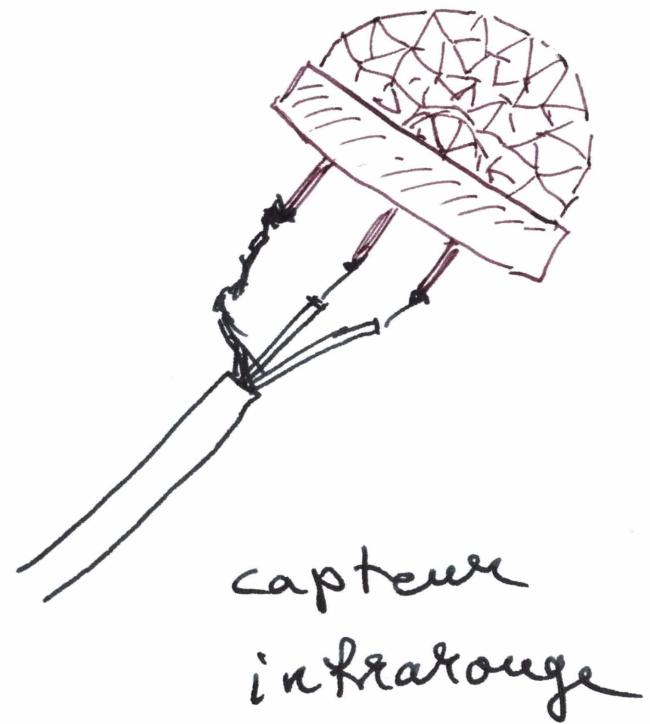
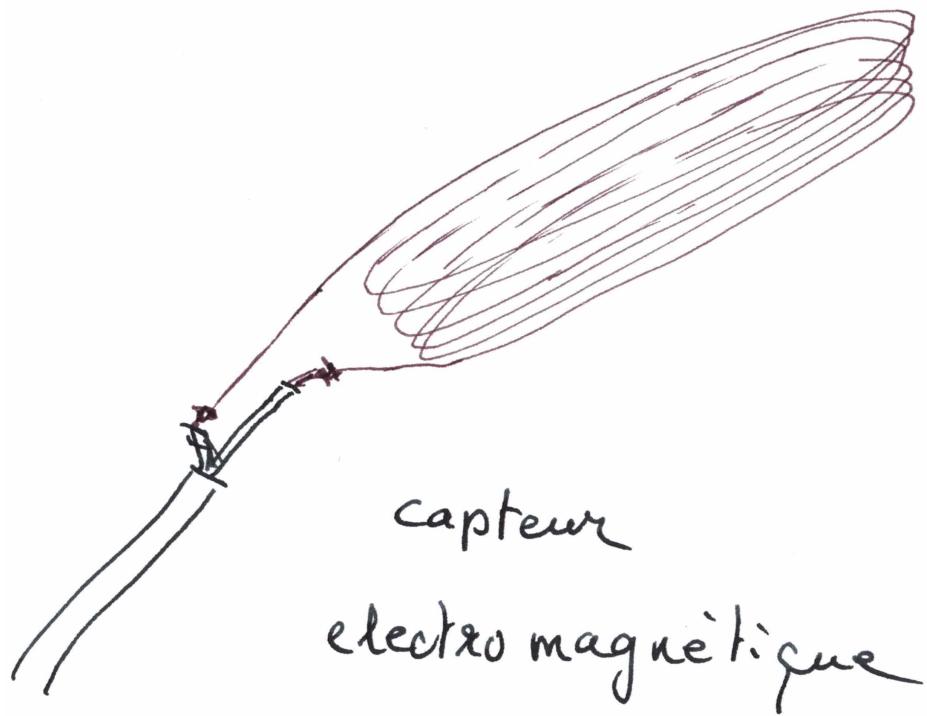
Arturia
Beatstep

Eléments d'interactions

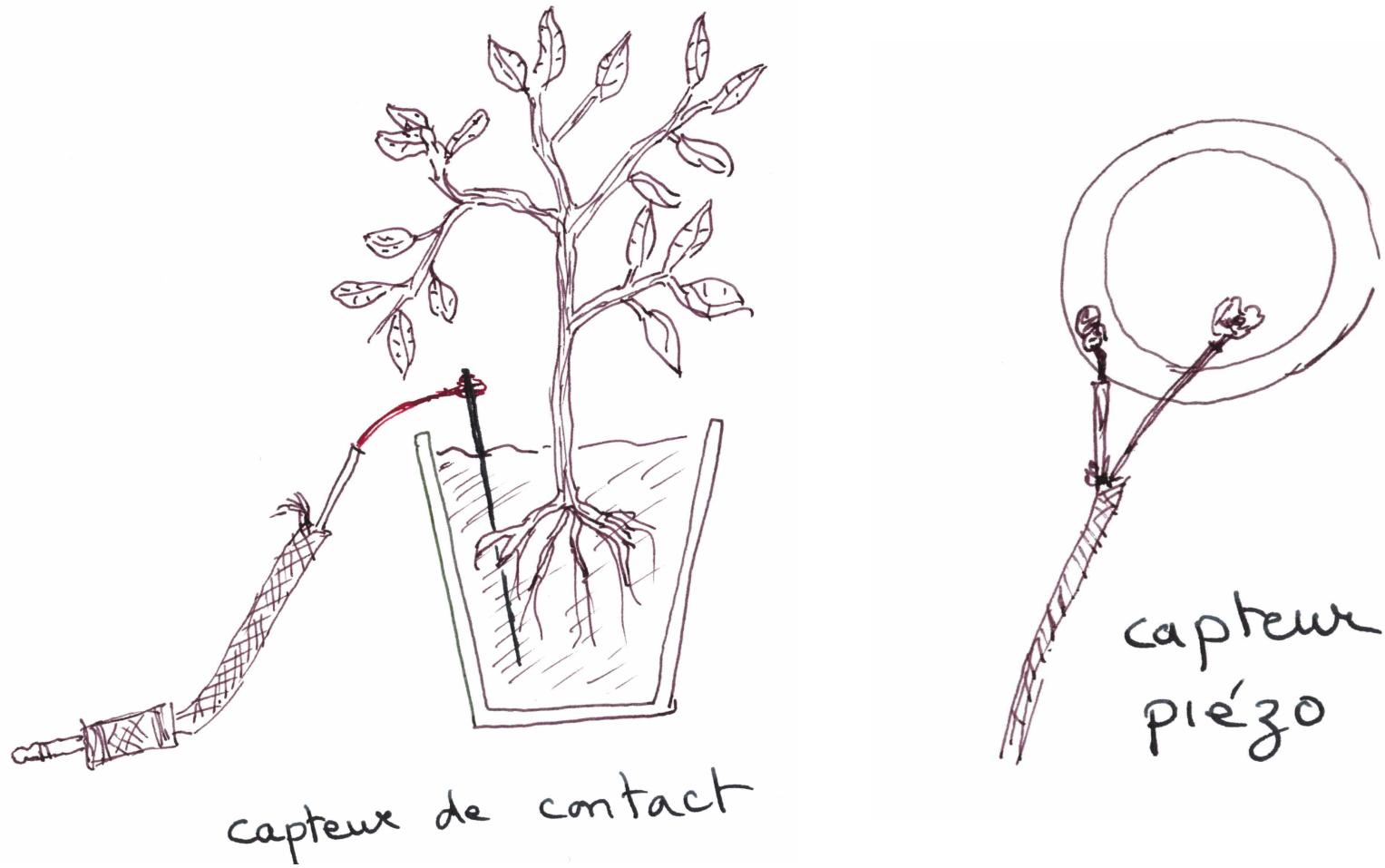


Sensel Morph

Eléments d'interactions



Eléments d'interactions



Nouveaux besoins

- Allocation dynamique de zones de mémoire

```
tab = freemem ; freemem += 1000 ;
```

 - Ne suffit plus !
- Outils de lecture de fichiers
 - Fichier texte, lecture par ligne
 - Fichier de nombres, décodage à la volée
 - Validation de syntaxe et sémantique ?
- Interprétation effective du contenu de ces tables, pour exécuter les actions
- Codification des **Interactions**
- Proposer « mieux » qu'un ensemble de conventions ad hoc

Description des interactions

- Associer une action à une interaction
 - Installation avec capteurs déclenchés par un danseur
- Besoin de séquences d'actions complexes
 - Nécessité d'une historique, représentation de l'état
 - Envie d'utiliser diverses approches algorithmiques
 - Automates, réseaux de pétri, systèmes chaotiques
- Fait apparaître la nécessité de décrire des algorithmes au travers d'un langage

Décision...

- Concevoir un « vrai » langage de script
 - Réutilisable dans d'autres projets
 - Implémenté en JSFX
 - Avec des caractéristiques qui me conviennent
- Environnement :
 - Destiné à être incorporé dans un JSFX
 - Choix logique : syntaxe JSFX
 - Mais quand même...
 - Tant d'erreurs de conception dans JS !

Différences entre JSFX et mSL

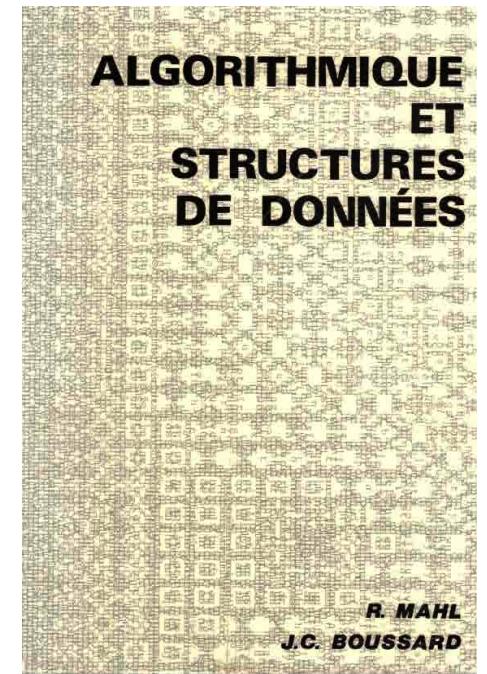
- Aspects de « mSL », *micro Script Language*
 - Déclaration des variables obligatoire, case sensitive
 - Globales, semi-locales, locales
 - Fonctionnel, fonctions « citoyens de première classe »
 - Récursion, cross récursion, références avant, détection des récursions terminales
 - Gestion de mémoire dynamique
 - malloc, free + garbage collector
 - Gestion de processus légers
 - Pas d'interférences avec la production de son

Les challenges ?

- Ecrire un langage de script en 2022 est trivial...
 - Outils, bibliothèques, connaissances, expériences...
- Mais... en JSFX ?
 - Pas d'outils d'analyse syntaxique
 - ~~Lex~~, ~~yacc~~, ~~flex~~, ~~bison~~, etc
 - Pas d'outils de développement
 - Limitations du langage
 - Type de données unique : le flottant double
 - Pas d'interface graphique « conviviale »
 - Pas d'outils de mise au point
 - Il faut travailler en temps réel

Les challenges ?

- Alors ?
 - Récupération d'une bibliothèque graphique
 - « ui-lib » open source, due à Geraint Luff (Merci!)
 - Nouvelles additions (pads, menus, etc.)
 - Incorporation de tests/traces/debug dans le logiciel
 - Utilisation de VS code
- Heureusement...
 - On a eu en 1970 des cours d'informatique...
 - Et on les a compris en 50 ans de carrière...

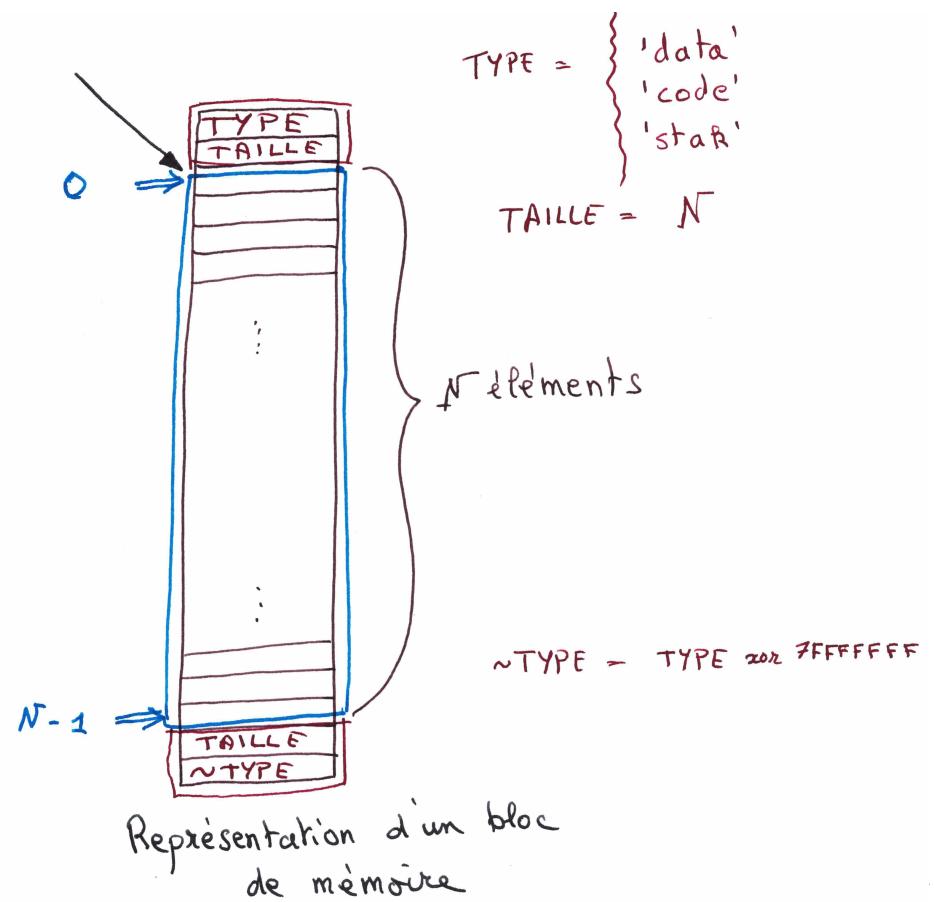


La gestion de la mémoire...

- Un besoin universel
 - Représenter (et allouer) les tables « fixes »
 - Répondre aux besoins du compilateur
 - Répondre aux besoins des programmes chargés dynamiquement
- Quel usage ?
 - Des tableaux classiques [0 ... n-1] : `tab[10]`
 - Des tables hashcodées, de l'adressage symbolique
 - `tab['code']` `tab[`symbole`]`

Les tables/tableaux mSL

- Structure unique :
 - Contenu encadré par en-tête et en-queue
 - Validation structure, basée sur le type
 - Constante multicaractères
 - 'code'
 - Tailles indiquées
 - Cohérences
- Allocation statique ou dynamique



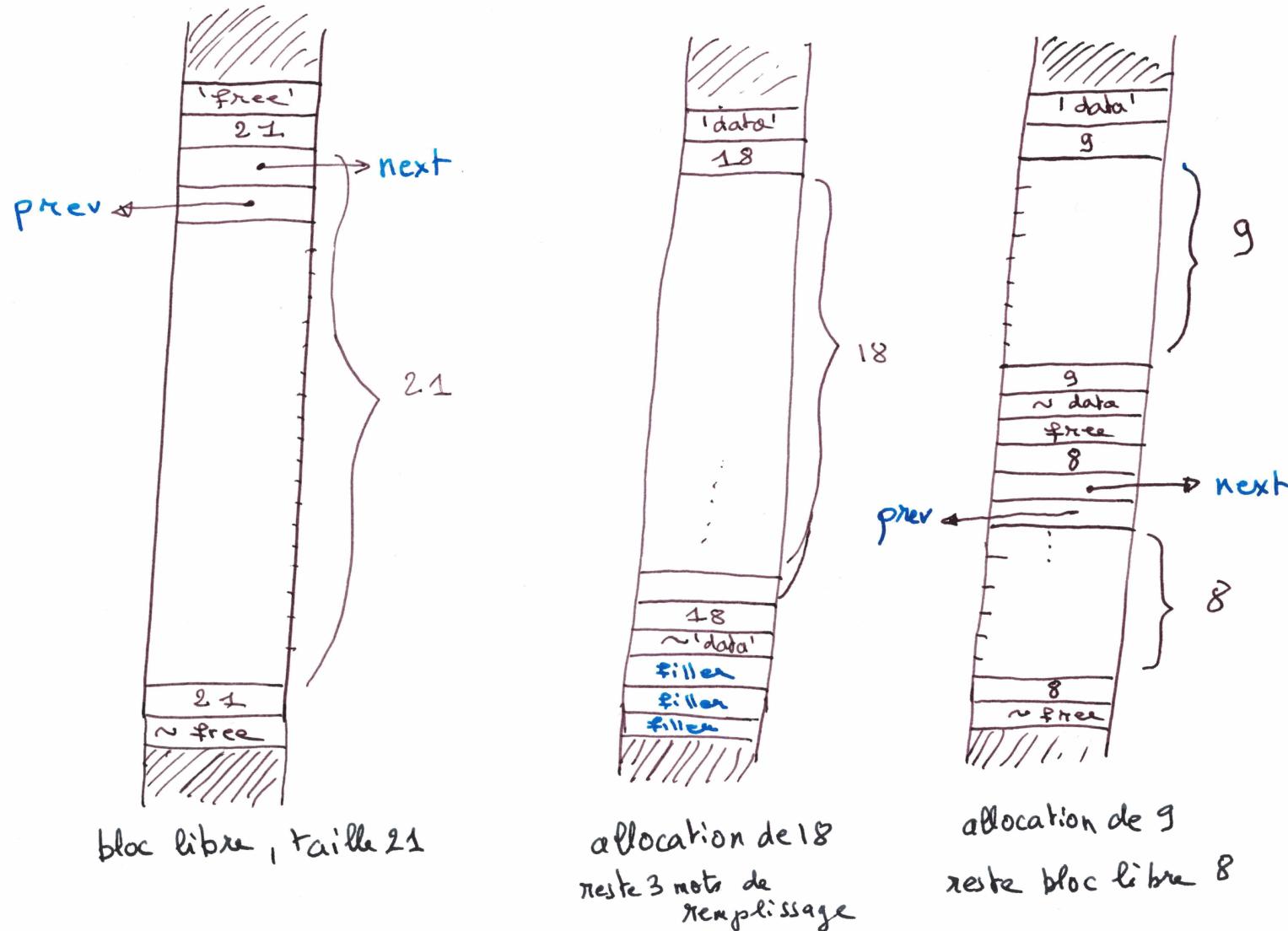
Comment rendre efficace une allocation dynamique ?

- Algorithmes nombreux, anciens, bien connus...
- Besoins « temps réel », durée des opérations strictement bornée : algorithmes en $O(1)$
- Liste des blocs libres doublement chaînée
- Une liste de blocs libres par classe de taille
 - Classes basées sur la série 2/3
 - 8 (..11) 12 (..15) 16 (..23) 24 (..31), etc.
 - Soit 40 à 50 classes
 - « good fit », perte inférieure à 50% de la taille demandée

Est-ce vraiment en O(1) ?

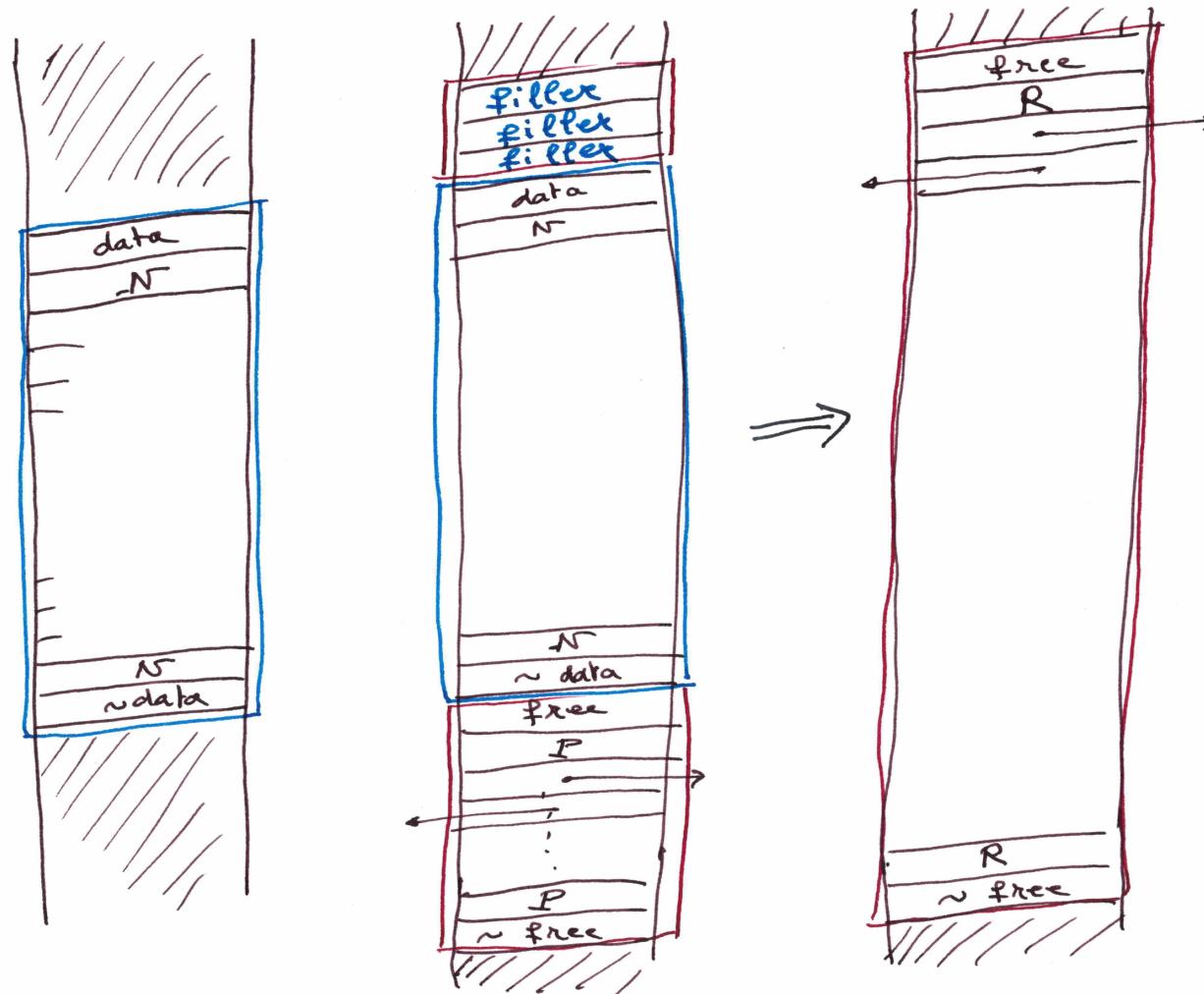
- Pour une demande de « N » mots
 - Recherche dichotomique dans la table des listes
 - $O(\log k)$: $k == 50$. Durée strictement limitée
 - Retrait du premier bloc de la liste
 - $O(1)$: 2 manipulations de pointeurs
- Pour une libération d'un bloc de « N » mots
 - Recherche dichotomique dans la table des listes
 - Insertion du bloc libéré en tête (ou en queue)
 - $O(1)$: 2 manipulations de pointeurs

Analyse d'une allocation



Note : il n'y a jamais 2 blocs libres contigus

Analyse d'une libération

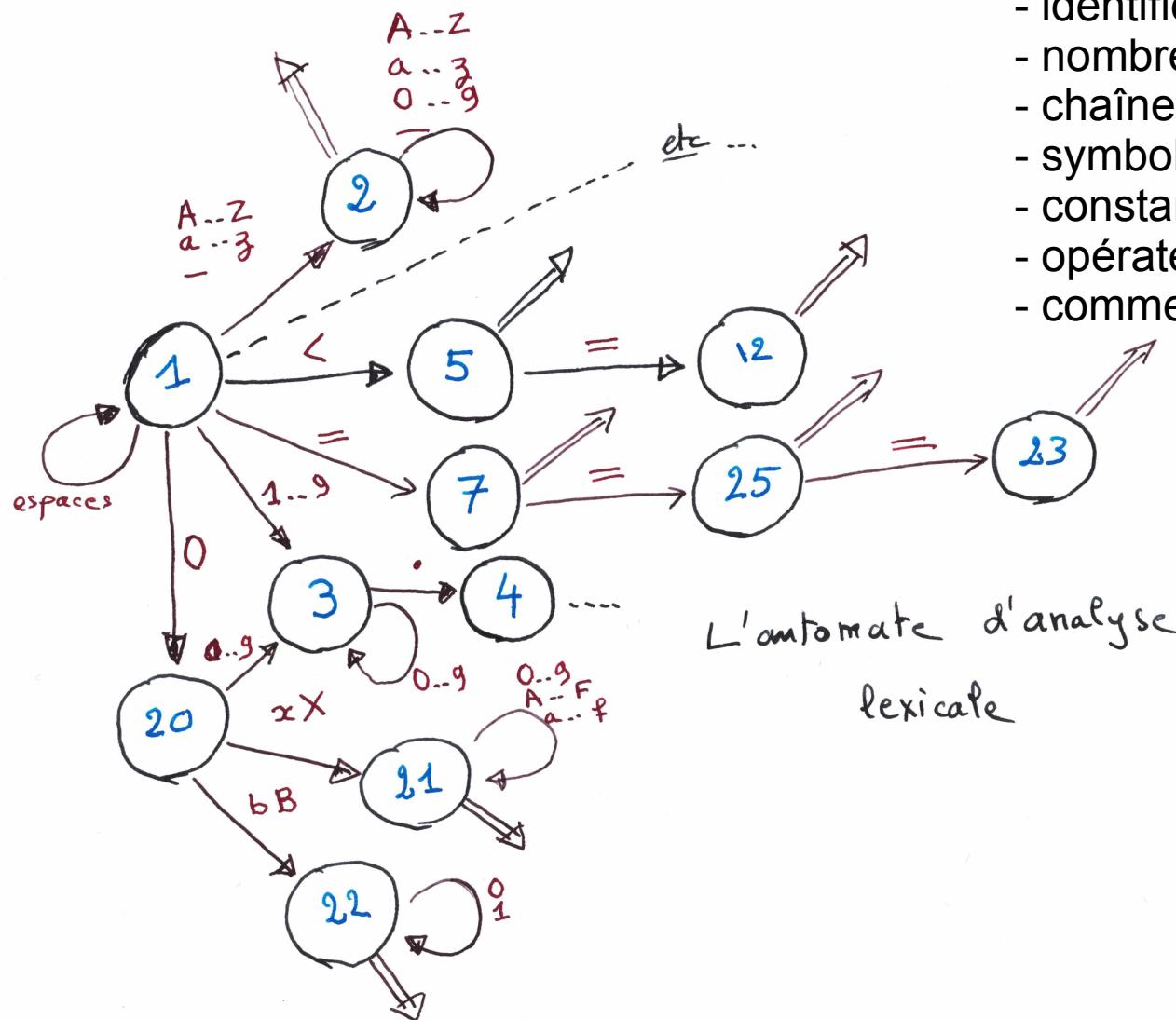


Libération d'un bloc de mémoire

Le compilateur mSL

- Générateur de code pour une machine virtuelle
 - « byte » code + opérande
 - Machine à pile
 - Tables symboles : globale/semi-locale
 - Fonctions : paramètres et locales allouées dans la pile
 - Références mémoire : accès direct à la mémoire JS
- Evaluateur
 - Boucle sur une série de tests destinés à reconnaître le code...

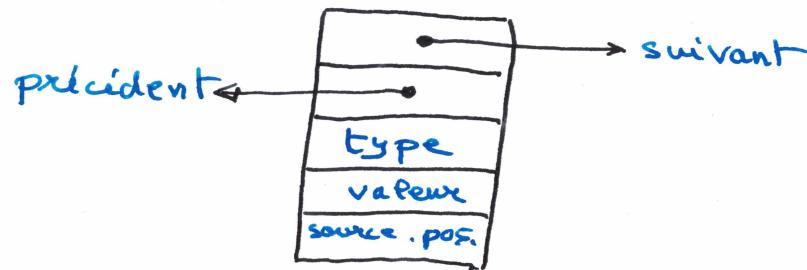
L'analyse Lexicale



Décode :

- identificateurs
- nombres entiers, flottants, 0xvvvvv, etc.
- chaînes de caractères
- symboles ('identificateur')
- constantes symboliques (\$pi, \$e)
- opérateurs < <= === && etc.
- commentaires // et /* */

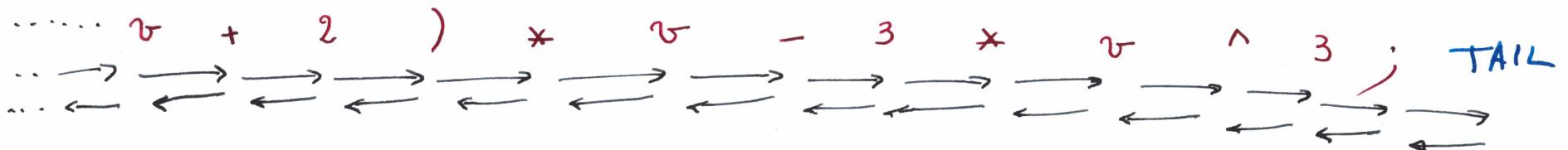
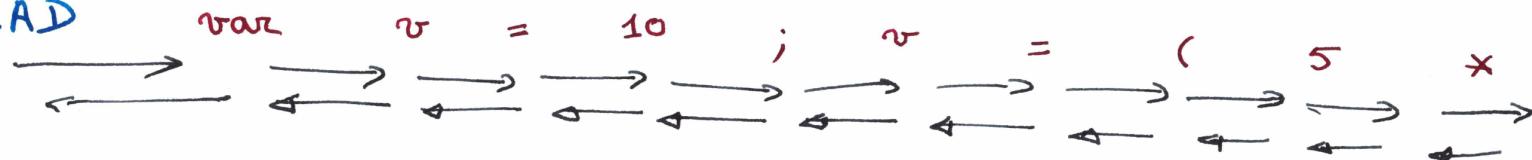
Codification d'une entrée



var v = 10;

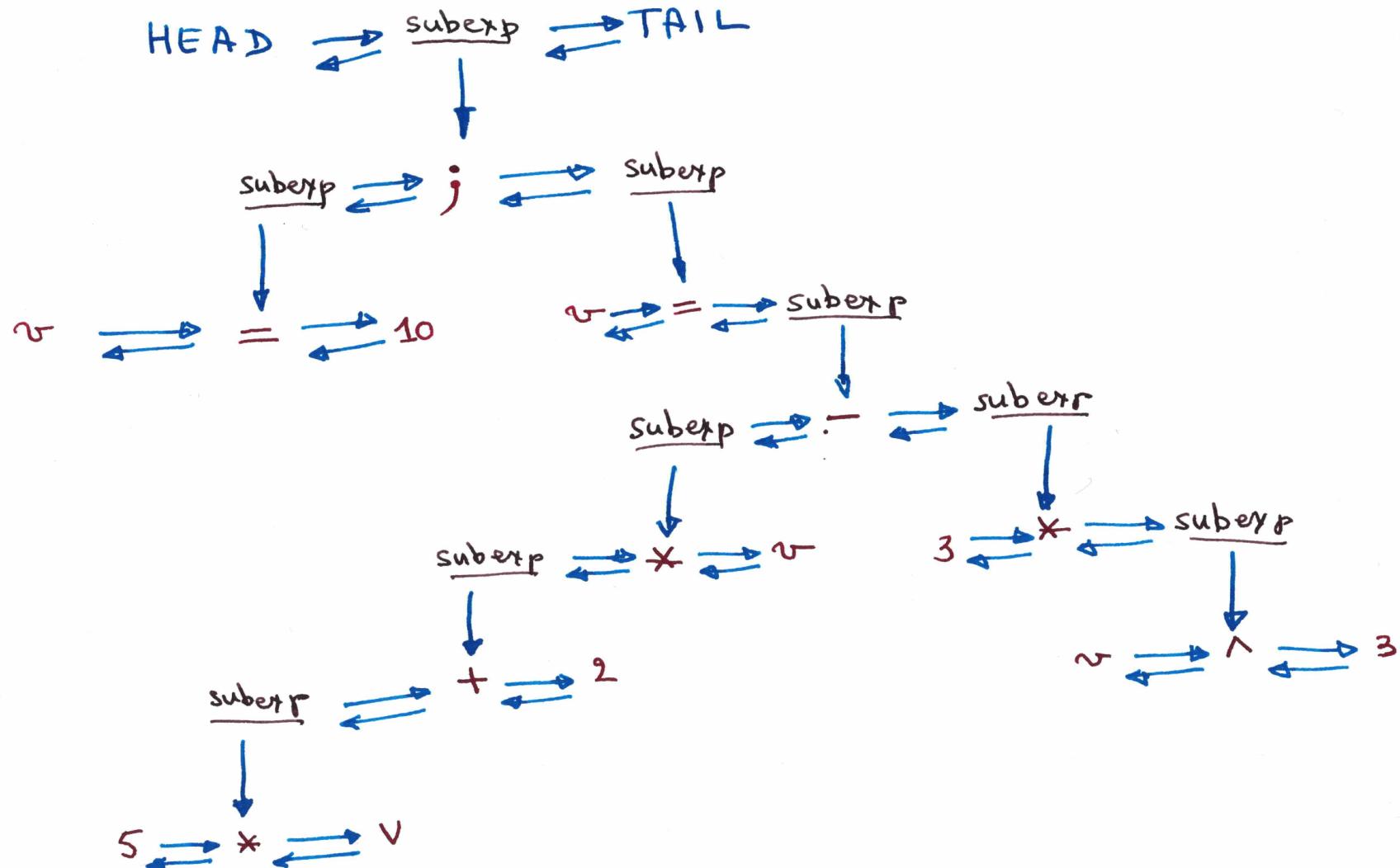
v = (5 * v + 2) * v - 3 * v ^ 3;

HEAD



Après l'analyse lexicale

L'analyse syntaxique



Après l'application des règles de réécriture

La génération de code

- Part de l'expression encadrée par HEAD – TAIL
 - Placée au sommet de la pile du compilateur (CS)
- Approche classique :
 - Si le TOCS est une sous-expression
 - Empiler l'opérateur, l'opérande droit, l'opérande gauche
 - TOCS = opérande :
 - générer le code qui place sa valeur dans le TOES
 - TOCS = opérateur
 - Générer le code qui applique l'opération sur le TOES

Le « byte code »

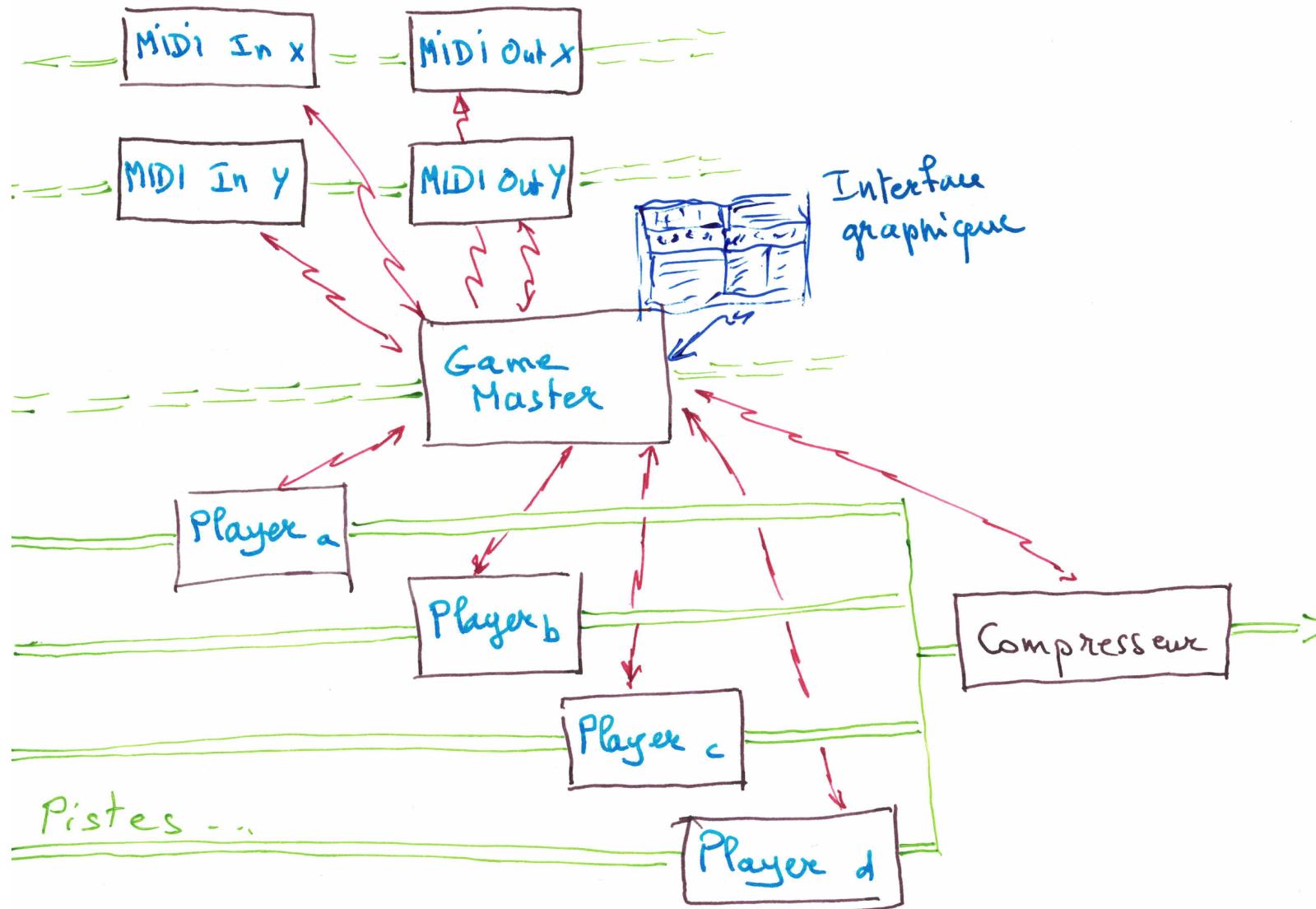
- Environ 140 instructions
 - Accès (lecture/écriture) aux objets du langage
 - Appel de fonctions primitives (sin, cos, +, etc.)
 - Gestion des appels/retours de sous-programmes
 - Opérations de branchements...
- Codification
 - Code sur 8 bits
 - 1 bit : op code long + code sur 8 bits
 - Opérande sur 32 bits + code sur 8 bits

```
// NOTE : Les tests sont générés par un programme « C »
(Xop&0x80)?((Xop&0x40)?((Xop&0x20)?((Xop&0x10)?((Xop&0x8)?((Xop&0x4)?((Xop&0x2)?((Xop&0x1)?(
    //====255==//
    // Multi-adic "mSL_255" — to be redefined later
    m = mSL_TOS[0];
    mSL_TOS -= m;
    mSL_TOS[0] = mSL_255(m, mSL_TOS)
):(
    //====254==//
    // Multi-adic "mSL_254" — to be redefined later
    m = mSL_TOS[0];
    mSL_TOS -= m;
    mSL_TOS[0] = mSL_254(m, mSL_TOS);
));)((Xop&0x1)?(
....;
);)((Xop&0x1)?(
    //====1==//
    // exit —
    mSL_ExitValue = mSL_TOS[0]; // exit value
    mSL_ICount -= rpeat; // deduce rest of count
    rpeat = 0;
    mSL_CodePtr = mSL_PCode; // lock on current address
    mSL_NxtOp = mSL_baseCode[mSL_CodePtr];
    mSL_CodeStop = mSL_RC_Stop; // set "stop" flag
):((
    //====0==//
    // illegal instruction "0"
    mSL_errX=1901;
    mSL_CodePtr = mSL_PCode; // lock on current address
    mSL_NxtOp = mSL_baseCode[mSL_CodePtr];
    mSL_CodeStop = mSL_RC_ILLInst; // illegal inst "stop" flag
);););););););;
```

Aspects temps réel

- Processus légers, scheduler préemptif
 - Gestion transparente au programmeur
 - ... ou explicite thread(), yield(), wait(), etc.
- Intégration de l'interprète dans la partie « temps réel haute priorité »
- Intégration du compilateur et du garbage collector dans la partie « basse priorité »
- Accès par mSL à la plupart des paramètres et opérations du système

Organisation du système



Quelques aspects originaux

Identificateurs

- Syntaxe : [A-Za-z_] [A-Za-z0-9_] *
- Représentation traditionnelle
 - Table des symboles, chaînes de caractères ou dictionary-tree
 - Codification : par numéro d'entrée
- Inconvénients
 - Peu de chaînes disponibles
 - Codification dépendante de l'historique
 - On veut compiler à la volée, charger des scripts déjà compilés.

Note : réflexions liées aux choix d'implantation, à la nature de JS, etc.

Identificateurs

- Besoins
 - associer à un identificateur une codification (nombre) unique, toujours la même, quelle que soit l'historique des opérations.
- Solution
 - Limiter le nombre de caractères d'un identificateur.
 - L'ensemble [A-Za-z0-9_] comporte 64 caractères, qui peuvent être codifiés sur 6 bits
 - Un float 64 bits permet de représenter tous les entiers de 0 à 2^{53} . Soit 8 caractères.
- Inconvénients
 - Quid de l'envie d'un programmeur d'utiliser plus de 8 caractères ?

Identificateurs

- Une solution relativement satisfaisante
 - Compactage des identificateurs
 - Préfixe de 1 caractère [0-9], « checksum » des transformations
 - Application déterministe d'une suite de règles
 - $\{A\}xx\{B\}$ devient $\{A\}x\{B\}$
 - $\{A\}xy\{B\}$ devient $\{A\}z\{B\}$: ch → H, qu → Q, es → 5...
 - $\{A\}x\{B\}$ devient $\{A\}\{B\}$ si $f(\{A\}, \{B\})$
 - Exemples
 - PresetCheckLoaded → 7p63bLdd
 - A_First_Long_Identifier → 2AfI2fr
 - A_Second_Long_Identifier → 5AzclGfr
 - mSL_SysLog041 → 1J1ylg4s
 - mSL_SysLog088 → 4m8Syq0q

Identificateurs

- Résultats après ajustement des règles
 - Pas de collision dans les identificateurs « compacts » générés
 - Fonction de hashcode :
 - $f(0xuuuuuuuvvvvvv) \rightarrow 0xuuuuuu \text{ xor } 0xvvvvvv$
 - Pas de collision dans la fonction de hashcode
 - Peu de collisions dans la mise en table de hashcode
 - 5% pour une table de taille 3^N
 - Deux identificateurs qui auraient le même « compact » seraient détectés du fait de l'obligation de déclaration.
 - Note
 - Les identifiEURS de moins de 8 caractères sont cadrÉS à gauche
 - Tous les identifiEURS sont dans :
[0x4000000000 . . . 0xFFFFFFFFFFFF]

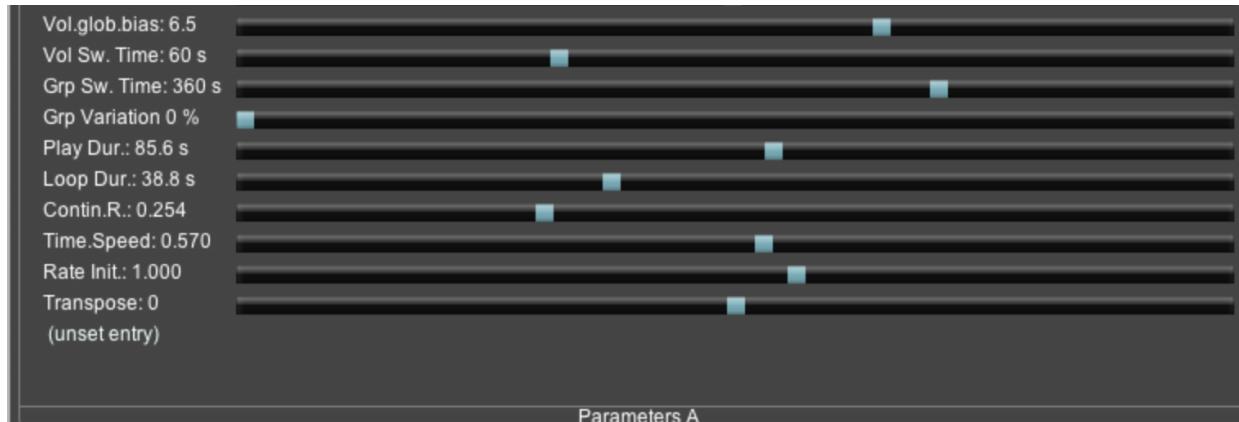
L'interface utilisateur

- Un WIP, donc une évolution incessante...
- Diverses finalités, aux nécessités différentes
 - Mode « concert »
 - Régler des paramètres, jouer des séquences
 - Modes « interactif » : live, installation avec capteurs
 - Mode « algorithmique » : programmes-compositions
 - Mode « studio »
 - Écoute de clips, modification des paramètres de jeu, analyse de l'historique

L'interface utilisateur

- Une fenêtre graphique unique
 - Tabulations 1 à 4 panneaux dans chaque tabulation, 1 ou + modules dans chaque panneau
- Chaque module a une finalité spécifique
 - Eléments graphiques : afficheurs, pads, potentiomètres, texte
 - En partie configurables
 - Choix libre de l'affichage des modules dans les panneaux
 - * création de nouveaux panneaux sous mSL

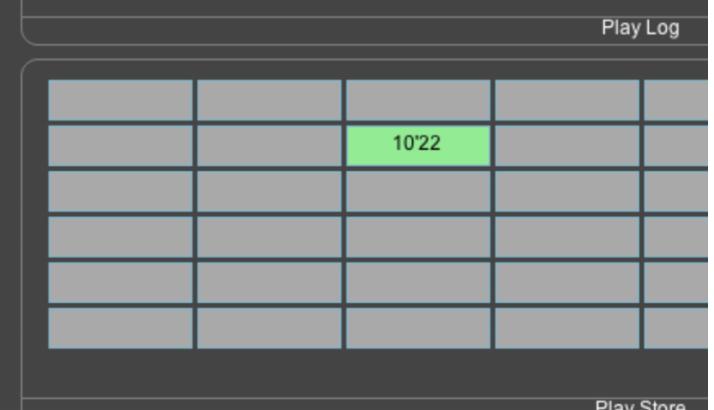
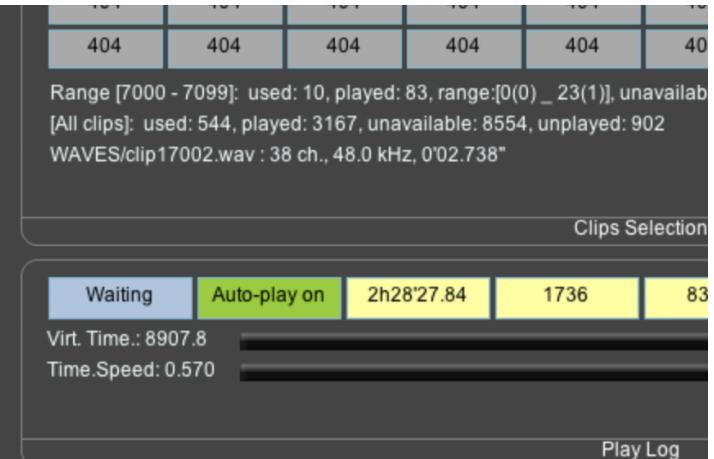
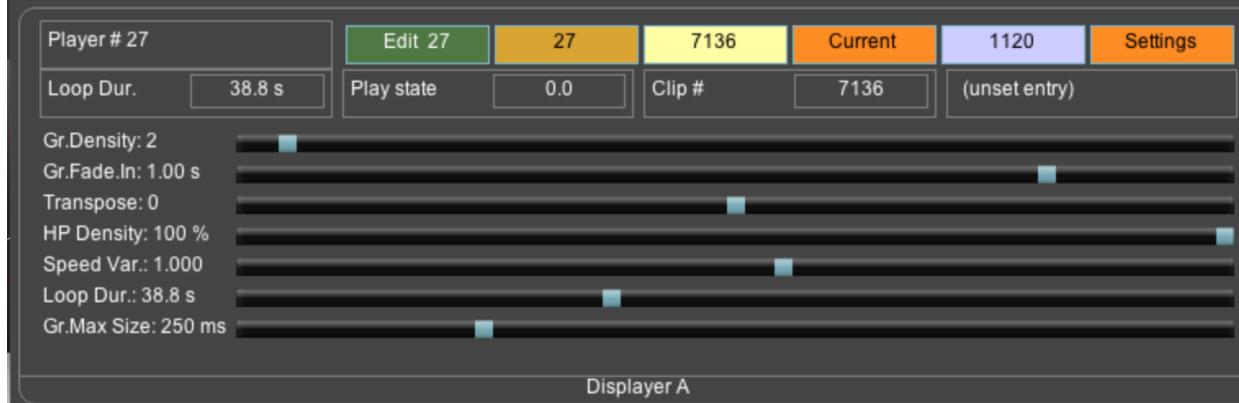
L'interface utilisateur



Parameters A

Player 0	Player 1	Player 2	Player 3	Player 4	Player 5	Player 6	Player 7
Player 8	Player 9	Player 10	Player 11	Player 12	Player 13	Player 14	Player 15
Player 16	Player 17	Player 18	Player 19	Player 20	Player 21	Player 22	Player 23
Player 24	Player 25	Player 26	Loop:7136	Loop:6209	Loop:7131	Loop:2609	Loop:3203
Loop:6651	Play:0135	Play:2310	Play:0536	Play:2400	Play:1114	Play:0516	Player 39
Player 40	Player 41	Player 42	Player 43	Player 44	Player 45	Player 46	Player 47

Sound Units



Play Store

T88-268614-Phase2 start	T91-It
T89-268638-End	T91-D
T89-Result: 3804758.854101	T88-2
T89-Iterations: 6000000	T90-2
T89-Duration: 261.22 s.	T90-R
T92-268662-End	T90-D
T92-Result: 4000000	S05-2
T92-Duration: 239.09 s.	S05-2
T91-290898-End.	T92-2
T91-Result: 3226120.958185	T91-2

System Log

Le « play log »

- Enregistrement de « sons figés »
 - Ensemble des paramètres utilisés lors du jeu d'un clip
 - Jusqu'à 100000 « sons figés », sous forme de deux listes doublement chaînées [débuts et fins]
 - Opérations proposées :
 - Arrêt sur un son, noter un instant
 - Jouer les clips entendus à un instant donné
 - Rejouer depuis un instant donné
 - Modifier la vitesse de jeu, ou n'importe quel paramètre
- Enregistrer l'audio [REAPER]
 - Temps réel ou temps différé

Est-ce vraiment une « aide » ?

- L'outil permet de découvrir rapidement les « heureuses rencontres », qui font que deux ou trois sons « fonctionnent bien » ensemble
- Le « play log » permet de revenir, écouter en boucle, modifier chaque paramètre
- Tous les sons produits peuvent être enregistrés
- Le langage mSL devrait permettre de nouvelles approches algorithmiques de la composition

Merci...

- De votre attention...
- Pour leur confiance, aux compositrices et compositeurs qui nous ont donné leurs sons
 - François Bayle, Jean-François Bau, Laurence Bourdin, André Dion, Jean-Marc Duchenne, Marc Favre, Béatrice Ferreyra, Alain Girardot, Diego Losa, Victor Minjard-Fel, Philippe Moënne-Loccoz, Isabelle Nesme, Gilles Roussi, Francis Valéry
- Aux danseuses et danseurs, pour leur prise de risque
 - Sarah Avelisjan Segealon, Virginie Barjonet, Julie de Bellis, Daisy Fel, Béatrice Garnier, Lætitia Naud, Kevin Naran, Cedric Sabatier
- A ceux dont l'aide nous a été précieuse
 - Annie Corbel, Olivier Dutel, Gaëlle Joly, Stephan Morschhauser
- Aux organismes et associations qui ont soutenu nos projets
 - La Ville de Saint-Etienne, le Gran Lux, CréoCorsica, Le Labo, Le Son des Choses, le Festival des Arts Numériques Pléïades, MiXiT

A cartoon illustration of a young boy with brown hair, wearing a blue cap and a purple jacket over a red shirt. He is holding a brown pipe in his right hand and a small white mug in his left hand. He is looking towards the right with a slight smile. The background is a light grey.

Pub !

Stage Reaper et GameMaster

11 au 15 Juillet
Ardèche

contact@lesondeschooses.audio

Reaper et GameMaster

Stage/atelier du 11 au 15 juillet 2022
à Jouangrand, Ardèche

Le son des choses
Jean-Jacques Girardot, Jean-François Minjard

Cinq jours dans une vallée de la Haute-Ardèche pour découvrir Reaper, de fond en comble ou vous mettre à niveau, ainsi que notre plug-in de génération de séquences sonores, le GameMaster.

Les journées se diviseront en deux parties, les matins sous la forme d'un stage et les après-midis sous la forme d'ateliers.

Le stage proposera, durant les cinq matinées et en un même lieu pour tous, une mise à niveau des connaissances et des usages de Reaper et du **GameMaster**.

Les ateliers permettront une mise en pratique des éléments envisagés lors de la partie stage et pourront être, à la demande, personnalisables. Ils se dérouleront dans deux lieux distincts permettant un prolongement plus théorique d'un côté et une expérience de l'espace multiphonique de l'autre sur l'**acousmonium 36.1** installé dans la grange.

Ces cinq jours s'adressent à toute personne désirant approfondir les questions du rapport entre les sons, **vos sons**, d'un côté et leur mise en relation dans une visée compositionnelle de l'autre par le truchement du GameMaster.

Venez avec vos sons ou de quoi en faire : microphones, enregistreurs, plug-ins et ordinateurs.

La semaine se clôturera par une écoute/concert le soir du vendredi 15.

L'accueil des stagiaires se déroulera dimanche 10 juillet dans l'après-midi et le départ se fera samedi 16 durant la matinée.

Le coût par personne est de 300 € — ou 220 € suivant les réductions habituelles : étudiant, chômeur... — et comprend l'hébergement pour les 6 nuits et les repas.

À noter: le stage ne peut accueillir plus de **10 personnes**
la date limite des inscriptions est le **1^{er} juillet**

contact@lesondeschooses.audio

Aller à Jouangrand : <http://kiwi.emse.fr/JJGCX/JaG/>
et son nouveau numéro : 09 88 66 21 19