DIKOTOMIA

—

Projet original inspiré de *Ionisation* de Edgar Varèse

Dossier d'admission – ENJMIN 2011

Martin BUSSY-PÂRIS

SOMMAIRE

Pièces jointes du dossier:
S'il vous manque une des pièces jointes du dossier, sachez que tout est téléchargeable ici: http://martin.bussy.free.fr/ENJMIN/Concours/

› INTRODUCTION

Ce dossier présente un concept de jeu sonore numérique basé sur une des œuvres musicales d'Edgar Varèse, *Ionisation*.

□ Dikotomia

Type: jeu sonore de réflexion et d'exploration
Public visé: la communauté des jeux sonores + le gamer voyant traditionnel (notamment grâce au mode multijoueur)
Modes de jeu: solo / 2 joueurs en réseau local ou online (coopération)
Plateforme: PC

Synopsis: La ville futuriste de Dikotomia est polluée par des rejets de gaz toxiques dus à une très forte urbanisation. L'accumulation d'un gaz particulier au niveau du sol a peu à peu rendu la totalité des humains aveugles sans que personne ne puisse rien y faire. Vous allez devoir explorer le sous-sol de la ville afin de retrouver des créatures naturellement immunisées au gaz toxique pour essayer de découvrir le secret de leur immunité.

□ Ionisation

• *L'œuvre d'Edgar Varèse*

Ionisation est une composition musicale pour treize percussions écrite entre 1929 et 1931 qui véhicule plusieurs concepts et idées:

-l'expansion et la variation de cellules rythmiques, en référence au phénomène physique de ionisation des molécules (les molécules gagnant ou perdant des électrons),

-l'abandon du système tempéré qui est désuet aux yeux de Varèse et qui ne permet pas, selon lui, d'exprimer musicalement nos émotions et nos idées,
-l'obsolescence des relations intervalliques du tempérament égal,

-l'organisation de masses sonores se mouvant les unes contre les autres,
-la mise en avant du timbre au détriment de la métrique et de la pulsation, au sein du rythme,
-l'agogique et le tempo rubato, en opposition au caractère métronomique de la quasi-totalité de la musique savante et populaire,

-la suggestion de sons caractéristiques de la vie urbaine moderne,
-la référence aux lieux communs partagés par la culture populaire (tambours militaires, sirènes de pompiers, etc.).

- ***Liens avec Dikotomia***

Le projet de jeu présenté dans ce dossier comporte quelques éléments qui font écho à l'œuvre de Varèse, à savoir :

-un environnement urbain futuriste incompatible avec la musicalité traditionnelle du système tonal occidental,
-des éléments de cet environnement urbain qui se rapportent à la vie citadine moderne et aux idées courantes véhiculées autour de la science-fiction,

-la présence de cellules rythmiques basées sur le même schéma, concernant la musique,
-la mise en avant des variations de timbre et de tempo aux dépens de la mélodie et de l'harmonie.

› UNIVERS DU JEU

□ Background

- ***La surface***

Dikotomia est une mégalopole moderne. Pour faire face à un fort accroissement de la population, il a fallu construire en hauteur et de façon très dense. Ainsi, les buildings sont presque collés les uns aux autres et on ne voit quasiment plus la lumière du jour depuis la surface. La base des buildings réunit des systèmes de pompes et de chaudières très bruyants qui alimentent ceux-ci en eau. C'est pourquoi la surface n'est pas très accueillante et que peu de gens y circulent à pied. On y trouve, par contre, un important système de transport automatisé de biens et de personnes. La quasi-totalité de la population habite et travaille plus haut, dans les buildings.

Suite à un accident industriel, un gaz toxique s'est répandu au niveau du sol, rendant aveugle la totalité des gens présents à la surface à ce moment-là. Dans les buildings, par crainte de voir la cécité atteindre toute la population dikotomienne, les autorités ont décidé d'imposer une quarantaine d'une durée indéfinie aux personnes exposées au gaz à la surface, tant que la situation ne serait pas résolue. Ces derniers, livrés à eux-mêmes, vont devoir retrouver la vue par tous les moyens. Ils se surnomment eux-mêmes les « surfaciens ».

- ***Le sous-sol***

Le sous-sol de Dikotomia est particulièrement meuble par endroits. Ainsi, les surfaciens peuvent, à l'aide de pelles, se frayer un chemin vers les profondeurs de la ville pour fuir la pollution et le bruit permanent de la surface. Ils peuvent y trouver de nombreux points d'eau, car le sous-sol est parcouru par de nombreuses canalisations et conduites alimentant en eau et en gaz les pompes et les chaudières des buildings.

□ Entités

• ***Les surfaciens***

Les humains exposés au gaz à la surface se surnomment ainsi. Ils ont tous perdu la vue.

• ***Les haut-perchés***

C'est de cette manière que les surfaciens appellent les habitants de Dikotomia qui n'ont pas été exposés au gaz au moment de l'accident industriel. Ils vivent dans les buildings et ont condamné tous les accès menant à ces derniers pour empêcher aux surfaciens de les rejoindre.

• ***Les véhicules automatisés***

La surpopulation a engendré un développement rapide des transports d'hommes et de marchandises à la surface où les piétons se font rares. Le sol est jonché de rails permettant à différents véhicules à sustentation magnétique de circuler à grande vitesse.

Les surfaciens, soumis à la quarantaine, ne peuvent pas désactiver les véhicules qui continuent à aller et venir, rendant les étroites rues de la ville très dangereuses, surtout pour des non-voyants.

• ***Les créatures***

Dans le passé, lorsque Dikotomia n'était pas aussi bruyante et polluée, les rues et les espaces verts de la ville étaient truffés de petits mammifères atypiques, d'origine inconnue, regroupés au sein de « nids », qui avaient pour particularité d'émettre des signaux électromagnétiques d'une longueur d'onde proche de celle des ondes radio.

Les scientifiques dikotomiens inventèrent un appareil, le Dikophone, permettant de capter ces signaux et de les convertir en signaux sonores de manière à pouvoir les écouter. Les premiers prototypes du Dikophone produisaient des sons purs, sans aucune harmonique. Pour embellir le son émit par les appareils, les scientifiques ajoutèrent des harmoniques musicales aux signaux sonores. La version la plus vendue du Dikophone synthétisait de cette façon le son d'un piano.

Les dikotomiens sortaient ainsi régulièrement dans les jardins de la ville avec leur dikophone personnel pour aller écouter les créatures « chanter ».

Seulement, avec le développement anarchique de la ville qui a suivi, les créatures ont peu à peu disparu et personne ne sait vraiment ce qu'elles sont devenues. Les habitants de Dikotomia ont progressivement délaissé leurs dikophones, abandonnés ou rangés dans un placard. Seuls quelques nostalgiques gardent encore leur dikophone sur eux en espérant le retour des créatures.

Par un heureux hasard, il se trouve justement que quelques surfaciens possédant un dikophone ont retrouvé la trace des créatures chantantes dans le sous-sol de la ville. Or, l'une des rares choses que savent les humains sur ces créatures, c'est qu'elles sont immunisées contre les maladies contractées par ces derniers et ceci pousse certains surfaciens à penser qu'elles seraient également immunisées contre le gaz qui les rend aveugles.

□ Personnages-joueurs

• *En solo*

Le joueur incarne un des rares humains exposés au gaz qui possèdent un dikophone. Les surfaciens cherchant à découvrir la cause de l'immunité des créatures, le personnage-joueur va devoir explorer le sous-sol de Dikotomia pour trouver ces fameuses créatures, regroupées en plusieurs nids.

• *En multijoueur*

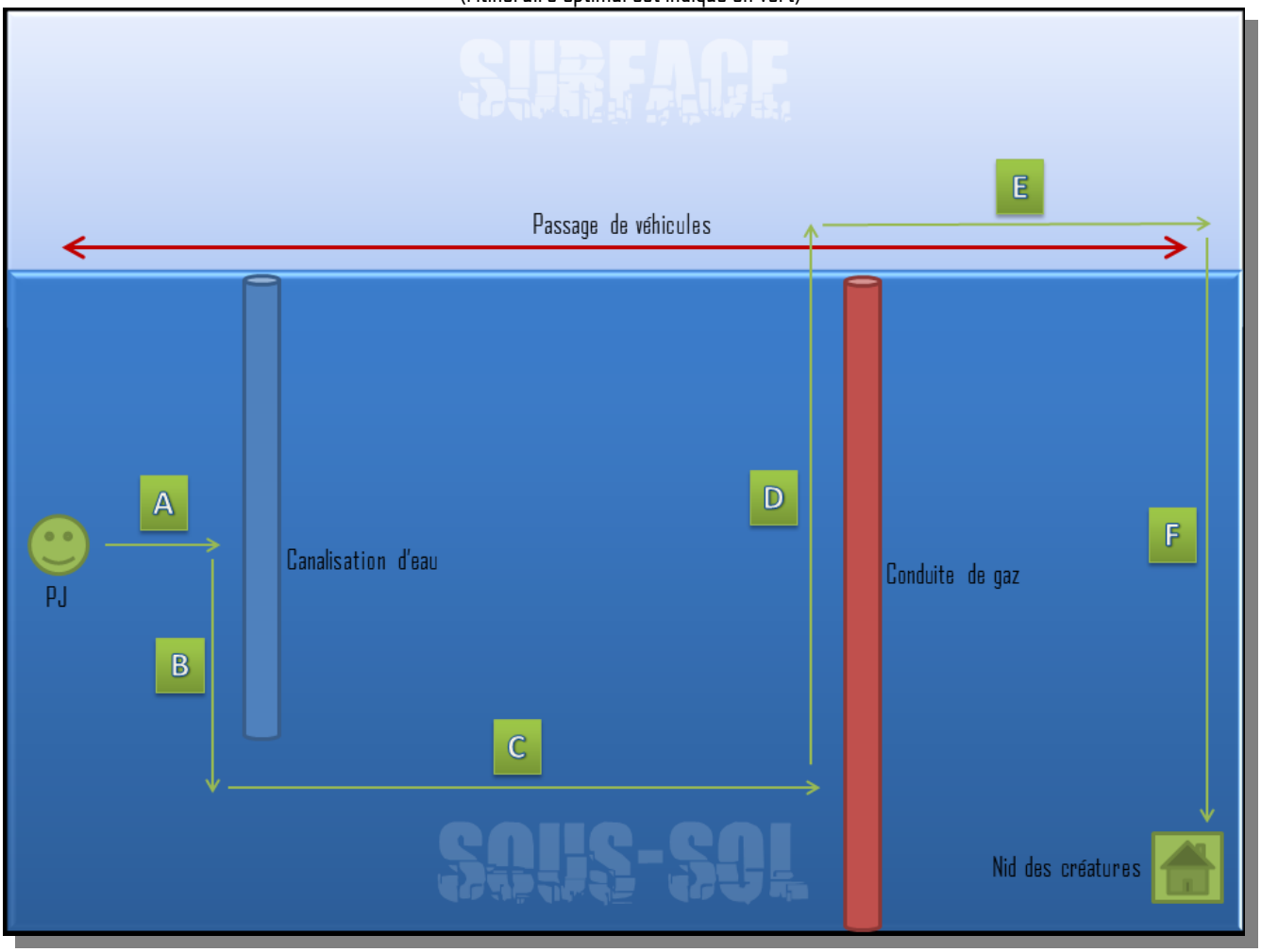
En mode multijoueur, le second joueur se met également dans la peau d'un des surfaciens possédant un dikophone. Ils vont devoir, ensemble, trouver les nids des créatures.

› MECANIQUES DE JEU

□ Principes du gameplay

Exemple de niveau

(l'itinéraire optimal est indiqué en vert)



Le joueur « spawn » toujours en sous-sol, quelque soit le niveau. En effet, il serait trop dangereux de commencer un niveau en surface, là où les véhicules vont et viennent.

Le but du jeu est de trouver le nid des créatures. Il n'y en a qu'un seul par niveau. Pour y parvenir, le joueur doit utiliser le dikophone pour localiser la source de la musique. Il peut se déplacer:
-sous-terre: en creusant des galeries avec sa pelle,
-en surface: en courant.

Plusieurs dangers parsèment les niveaux:
-**Les véhicules automatisés:** ils font des va-et-vient à la surface, de gauche à droite, et sont très rapides. L'impact conduit en général à la mort du personnage-joueur, mais si ce dernier a de la chance, il peut éventuellement s'en sortir (il a une chance sur cinq d'y survivre).
-**Les canalisations d'eau:** le sous-sol de la ville est parsemé de canalisations vétustes. Si, par malheur, le personnage-joueur plante sa pelle dans une canalisation, de l'eau s'en échappe en grande quantité et il se noie en quelques secondes.
-**Les conduites de gaz:** de la même manière que les canalisations d'eau, les conduites de gaz sont à éviter à tout prix car elles sont vieilles et fragiles. Si le joueur plante sa pelle dans l'une d'elles, la cavité où il se trouve est soufflée par les flammes et il meurt très rapidement.

Le joueur doit être attentif aux sons produits par ces 3 types d'obstacles différents.

• *Son et gameplay*

Dans Dikotomia, les sons permettent de détecter, d'identifier et de localiser l'objectif et les dangers:
-*Le nid des créatures:* localisable grâce à la musique
-*Les véhicules automatisés:* localisable grâce au son spatialisé qui leur est associé
-*Les canalisations d'eau:* localisable grâce au son qu'elles émettent
-*Les conduites de gaz:* localisable grâce au son qu'elles émettent

Le joueur peut, à tout moment, éteindre son dikophone de manière à mieux percevoir les dangers qui l'entourent. Il peut, de la même manière, le rallumer.
Tout ceci sera décrit plus en détails dans la partie Son.

• *Exemple de partie*

Le personnage-joueur (PJ) « spawn » sur la gauche du niveau, en sous-sol (le point de spawn est représenté par le smiley vert).

PHASE A
-le joueur met son Dikophone en marche,
-il comprend, en se déplaçant légèrement et en écoutant la musique, que le nid est vers la droite,
-il commence donc à creuser dans cette direction,
-il entend tout à coup la canalisation d'eau.

PHASE B
-il essaye de contourner la canalisation par le bas (notez qu'il aurait aussi pu passer par la surface, mais que cette option est plus périlleuse).

PHASE C
-n'entendant plus la canalisation, il en déduit que la voie est libre sur la droite,
-il continue à creuser vers la droite et la musique lui confirme que le nid est dans cette direction,
-il entend tout à coup la conduite de gaz.

PHASE D
-il essaye de contourner la conduite par le haut,
-la conduite l'empêchant toujours d'aller vers la droite, il est obligé de sortir à la surface pour la contourner.

PHASE E
-à la surface, il éteint son Dikophone car le brouillage l'empêche de se servir de la musique pour se diriger,
-il peut enfin aller vers la droite car il n'y a plus d'obstacle,
-il entend subitement un véhicule arriver par la gauche. Il a tout juste le temps de retourner en sous-sol pour éviter l'impact mortel.
-il peut écouter, juste au-dessus de lui, à la surface, le véhicule repartir vers la droite.

PHASE F
-maintenant qu'il a contourné la conduite de gaz, il remet son Dikophone en marche et descend vers le nid.
-en descendant, il note les changements dans la musique et en déduit qu'il est sur la bonne voie,
-il atteint le nid et remporte donc le niveau.

Deux simulations sonores de ce niveau (« Demo_1.mp3 » et « Demo_2.mp3 ») seront présentées à la fin de la partie Sound Design, ainsi qu'à la fin de la partie Musique. Il est conseillé de relire ce passage écoutant ces extraits.

□ Contrôles

-Déplacements : Touches fléchées
-Dikophone ON/OFF: Espace

□ Interface

• *Interface visuelle*

Dikotomia étant un jeu sonore, l'interface visuelle a peu d'intérêt et permet juste aux joueurs voyants non habitués aux interfaces sonores de naviguer plus facilement dans les menus.

• *Interface sonore*

Les menus du jeu doivent être utilisables par les non-voyants. Pour ce faire, on utilisera un design sonore simple et clair en complément d'informations vocales concises.

Une voix-off présentera l'univers et les enjeux au joueur en début de partie. Elle sera accompagnée par une illustration sonore riche immergeant le joueur dans l'ambiance du jeu.

En mode multijoueur, les 2 joueurs peuvent se parler grâce à un système de communication vocale intégré au programme. Dans le jeu, il s'agit d'un dispositif de communication radio intégré aux dikophones des deux personnages-joueurs. En surface, les signaux radio sont brouillés par les véhicules automatisés et les PJ auront ainsi beaucoup plus de mal à communiquer si l'un d'eux est en surface et que l'autre ne l'est pas. S'ils sont tous les deux en surface, on suppose qu'ils sont assez proches pour se parler et le système fonctionne, mais les voix sont couvertes par les bruits environnants.

□ Level design

• ***Mode histoire***

Dans un jeu sonore, il faut éviter de perdre l'intérêt du joueur voyant en proposant un gameplay trop complexe auquel il n'est pas habitué (en tant que pratiquant de jeux avant tout visuels). Le level design de Dikotomia se veut ainsi très progressif. En effet, à chaque découverte d'un nouvel élément de gameplay, le joueur sera informé brièvement par la voix-off sur la nature de cet élément juste avant le lancement du niveau.

-**Niveau 1:** présence du nid des créatures et des véhicules automatisés uniquement.
-**Niveau 2:** ajout de l'élément « canalisation d'eau ».
-**Niveau 3-5:** complexification du réseau de canalisations d'eau.
-**Niveau 6:** ajout de l'élément « conduite de gaz ». Vitesse plus importante des véhicules automatisés.
-**Niveau 7-10:** complexification du réseau de canalisations d'eau et du réseau de conduites de gaz.
-**Niveau 11-15:** complexification du réseau de canalisations d'eau et du réseau de conduites de gaz poussant le joueur à remonter souvent en surface. Intégration d'un nouvel élément de gameplay consistant à programmer l'arrêt aléatoire des véhicules automatisés, permettant au joueur de se déplacer sans danger à la surface pendant un court laps de temps (quelques secondes).
-**Niveau 16-20:** les réseaux de canalisations d'eau et de conduites de gaz forment désormais de véritables labyrinthes et il est difficile de ne pas s'y perdre. Le laps de temps pendant lequel les véhicules automatisés sont arrêtés est désormais très court et leur vitesse est très élevée (il faut vraiment tendre l'oreille pour les entendre arriver avant qu'il ne soit trop tard).

• ***Mode libre***

Contrairement au mode histoire, le mode libre ne propose pas de niveaux prédéfinis. Ici, les niveaux sont générés aléatoirement, de façon à proposer au joueur ayant fini le mode histoire de continuer à jouer sans jamais être confronté deux fois au même niveau.
Le joueur choisit son niveau de difficulté et le niveau est généré en fonction de son choix:
-**Très facile:** présence du nid des créatures et de l'élément « canalisation d'eau ».
-**Facile:** ajout de l'élément « conduite de gaz ».
-**Normal:** complexification du level design et augmentation de la vitesse des véhicules.
-**Difficile:** ajout de l'arrêt aléatoire des véhicules automatisés et complexification des niveaux.
-**Cauchemardesque:** complexification labyrinthique des réseaux de canalisations d'eau et de conduites de gaz.

Bien entendu, les niveaux sont générés de façon à être forcément jouables. Il faut que le joueur ait le sentiment de pouvoir finir le niveau.

› SON

□ Une bande-son au service du gameplay

Le rôle du son dans Dikotomia est avant tout une affaire de gameplay. En effet, contrairement à beaucoup de jeux, aucun son n'a ici un rôle uniquement esthétique ou immersif. Ces aspects ne sont pas mis de côté, mais il passe au second plan et le gameplay prime.

L'autre élément particulier de Dikotomia est la présence quasi exclusive de sons non spatialisés. En effet, dans plupart des jeux sonores où le joueur doit se déplacer dans le plan ou dans l'espace, les sons sont spatialisés. Dikotomia, au contraire, ne mise pas sur la spatialisation, mais sur la capacité du joueur à identifier et à analyser certains sons parmi une masse sonore complexe. Ainsi, seul le son des véhicules automatisés est spatialisé dans le jeu.

Note: les fichiers sons joints au dossier sont tous issus directement de FMOD Designer, FMOD Event Player ou Game Maker. Aucun montage n'a été fait par la suite. Tout a été pensé pour pouvoir simuler en temps réel des parties de Dikotomia.

□ Sound Design

• ***Méthode de travail***

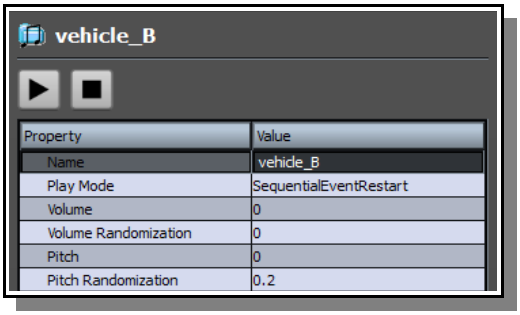
Tous les effets sonores réalisés pour Dikotomia ont été travaillés avec Cubase et Sound Forge, pour ensuite être intégrés à un projet FMOD Designer de manière à pouvoir être tous gérés en temps réel et pour simuler des situations de jeu. Enfin, pour écouter la totalité des sons en simultané et tester la cohérence de la bande sonore, j'ai utilisé FMOD Event Player.

J'ai choisi de réaliser les sons les plus intéressants à gérer en temps réel. Certains sons du jeu n'ont pas été produits.

• ***Véhicules automatisés***

Pour éviter que les véhicules sonnent tous de la même manière, j'ai appliqué un pitch aléatoire aux 2 sons utilisés par ces derniers:

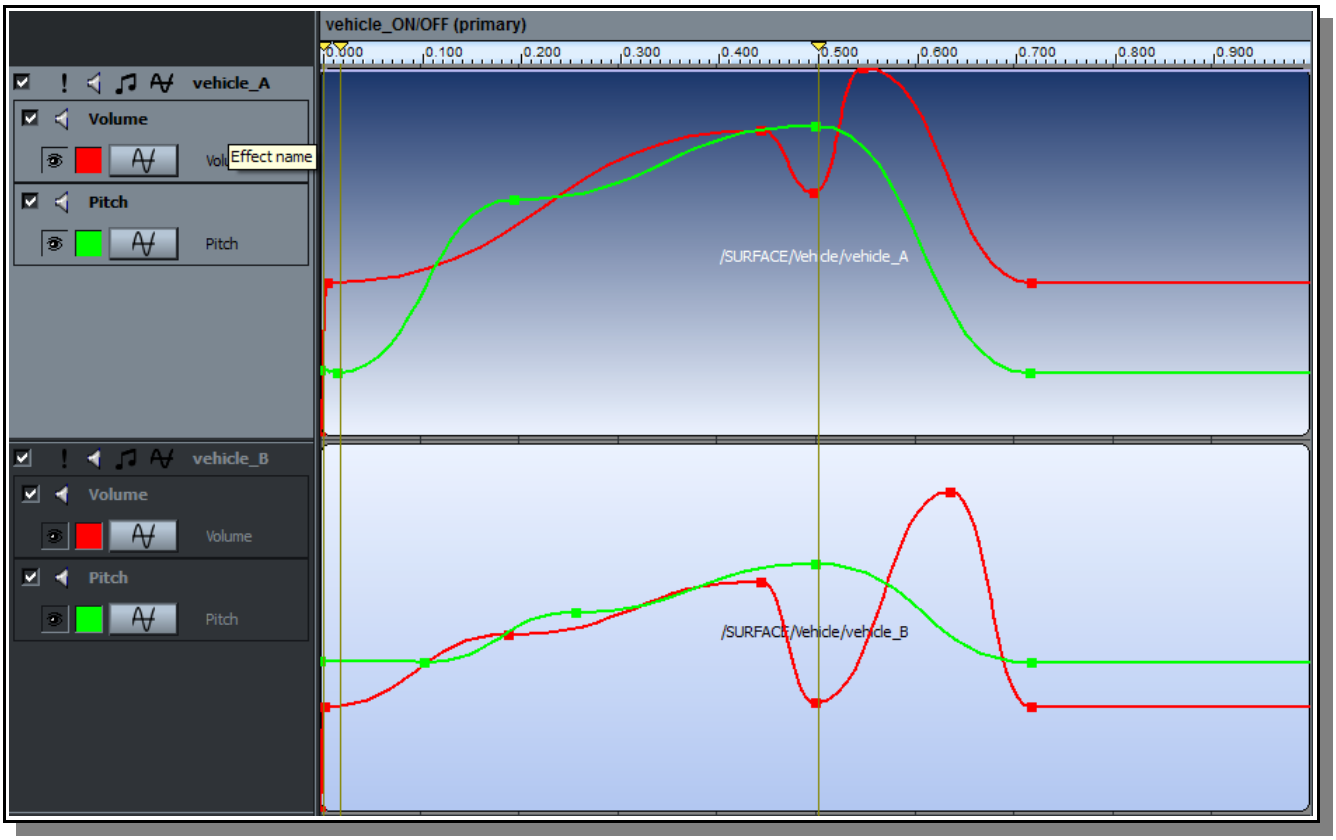
Pitch Randomization



L'unité du pitch est l'octave. Ici, le son sera donc pitché aléatoirement d'une valeur inférieure ou égale à 0,2 octave dans le grave ou l'aigu.

Les véhicules automatisés circulent en surface en passant horizontalement dans l'écran de jeu. Parfois, ils s'arrêtent momentanément au milieu du niveau pour une durée indéterminée (relativement courte).
Il a donc fallu définir plusieurs états pour gérer le son des véhicules:
-STOP: moteur en marche, mais véhicule immobile.
-STARTING: accélération.
-IDLE: moteur en marche, pleine vitesse.
-STOPPING: décélération.

Moteur des véhicules



FMOD applique en temps réel une valeur de volume et une valeur de pitch aux 2 sons (ce pitch est indépendant du Pitch Randomization vu précédemment). Ces valeurs sont représentées par les courbes. Les marqueurs jaunes (Sustain Points) permettent au curseur de la timeline de se figer à des endroits précis. Ici, le 2nd Sustain Point correspond à l'état **STOP**, et le 3ème à l'état **IDLE**. On peut également paramétrer la vitesse de défilement du curseur sur la timeline (pour faire accélérer et décélérer plus ou moins vite le véhicule).

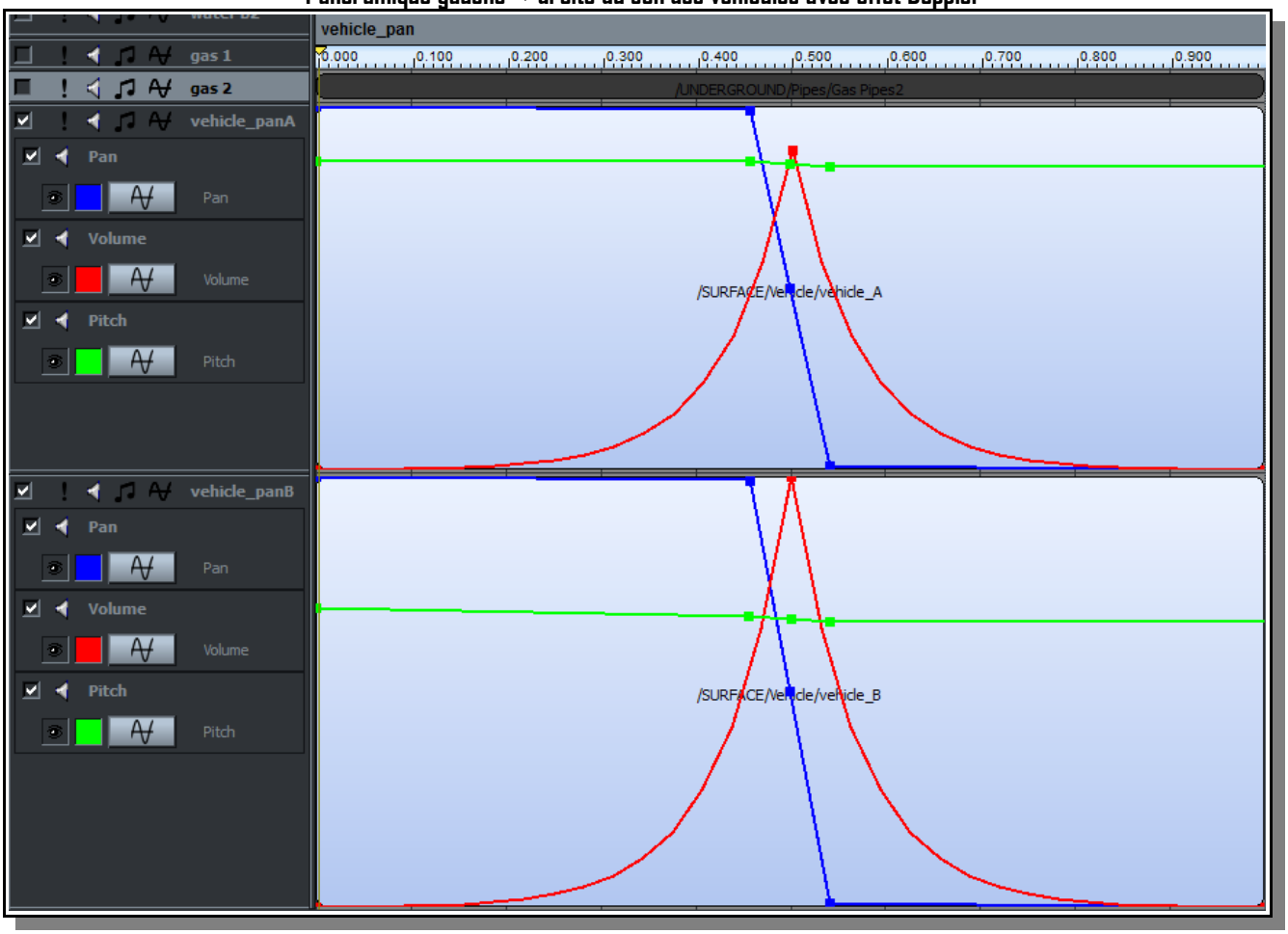
Voici quelques exemples:

♫ **Vehicle_Engine_1.mp3:** ici, on passe de l'état **STOP** à l'état **IDLE** lentement. Autrement dit, le véhicule accélère doucement. On revient ensuite de nouveau à l'état **STOP**, c'est la décélération.

♫ **Vehicle_Engine_2.mp3:** idem, mais avec une vitesse de défilement du curseur de la timeline plus importante.

♫ **Vehicle_Engine_3.mp3:** ici, la vitesse de défilement est encore différente et on reste à l'état **IDLE**.

Panoramique gauche → droite du son des véhicules avec effet Doppler



Ici, on peut entendre le passage en surface d'un véhicule, de gauche à droite. En plus de la courbe de panoramique et de la courbe de volume, il y a également une courbe de pitch qui permet de simuler un effet Doppler (lorsque le véhicule s'approche de l'auditeur, le son qu'il perçoit est plus aigu que le son émis par le moteur et lorsque le véhicule s'éloigne de lui, le son perçu est plus grave). Il s'agit du seul son spatialisé du jeu.

Voici quelques exemples avec l'ambiance de surface dont on parlera plus tard:

♫ **Vehicle_Pan_1.mp3:** le véhicule passe de gauche à droite. Pour les besoins de la démonstration, le véhicule passe à travers le PJ, ce qui n'arrive jamais dans le jeu.

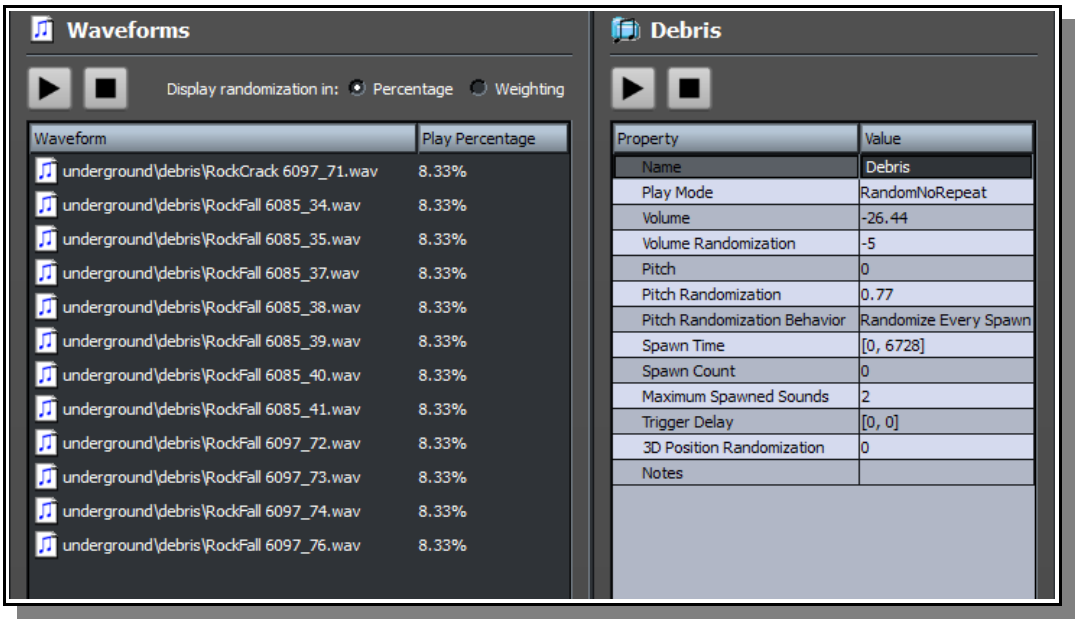
♫ **Vehicle_Pan_2.mp3:** idem, en plus rapide.

- ***Ambiance du sous-sol***

Pour faire une ambiance à base d'une boucle sonore, j'ai créé deux Sound Definitions dans FMOD Designer contenant le même son, sauf que le second son n'est pas lancé dès le début. En effet, il est lancé aléatoirement, entre 1000 et 5000 ms plus tard (Trigger Delay). De cette façon, on n'entendra pas le premier son boucler car il sera couvert par le second et inversement.

A ces deux sons, j'ai rajouté un pitch aléatoire (Pitch Randomization) d'1 octave chacun. J'ai également appliqué un pitch fixe de 0,7 octave dans le grave au premier son, de manière à ce qu'il sonne plus grave que le second son.

Sound Definition contenant les sons de débris



Aux deux boucles sonores, j'ai ajouté des sons plus brefs de poussières, de gravats et de légers éboulements, soit 12 sons supplémentaires. A ces sons, j'ai appliqué plusieurs choses:
-une lecture aléatoire d'un des 12 sons sans que le même son puisse être joué deux fois à la suite (Play Mode: RandomNoRepeat)
-un volume aléatoire (Volume Randomization),
-un pitch aléatoire (Pitch Randomization),
-le fait que FMOD attende aléatoirement entre 0 et 6728 ms après le lancement d'un son pour en déclencher un autre (Spawn Time),
-le fait que 2 sons puissent être joués simultanément et ainsi se superposer (Maximum Spawned Sounds),
-le fait que tous les sons soient équiprobables (Play Percentage: valeurs identiques).

L'ambiance du sous-sol est donc fortement stochastique (soumise à l'aléatoire) et le joueur, sans vraiment être conscient des différences à chaque écoute, n'éprouvera en tout cas aucune sensation de répétition.

Voici quelques exemples:

♫ **Amb_Underground_1.mp3**

♫ **Amb_Underground_2.mp3**

♫ **Amb_Underground_3.mp3**

- ***Ambiance de la surface***

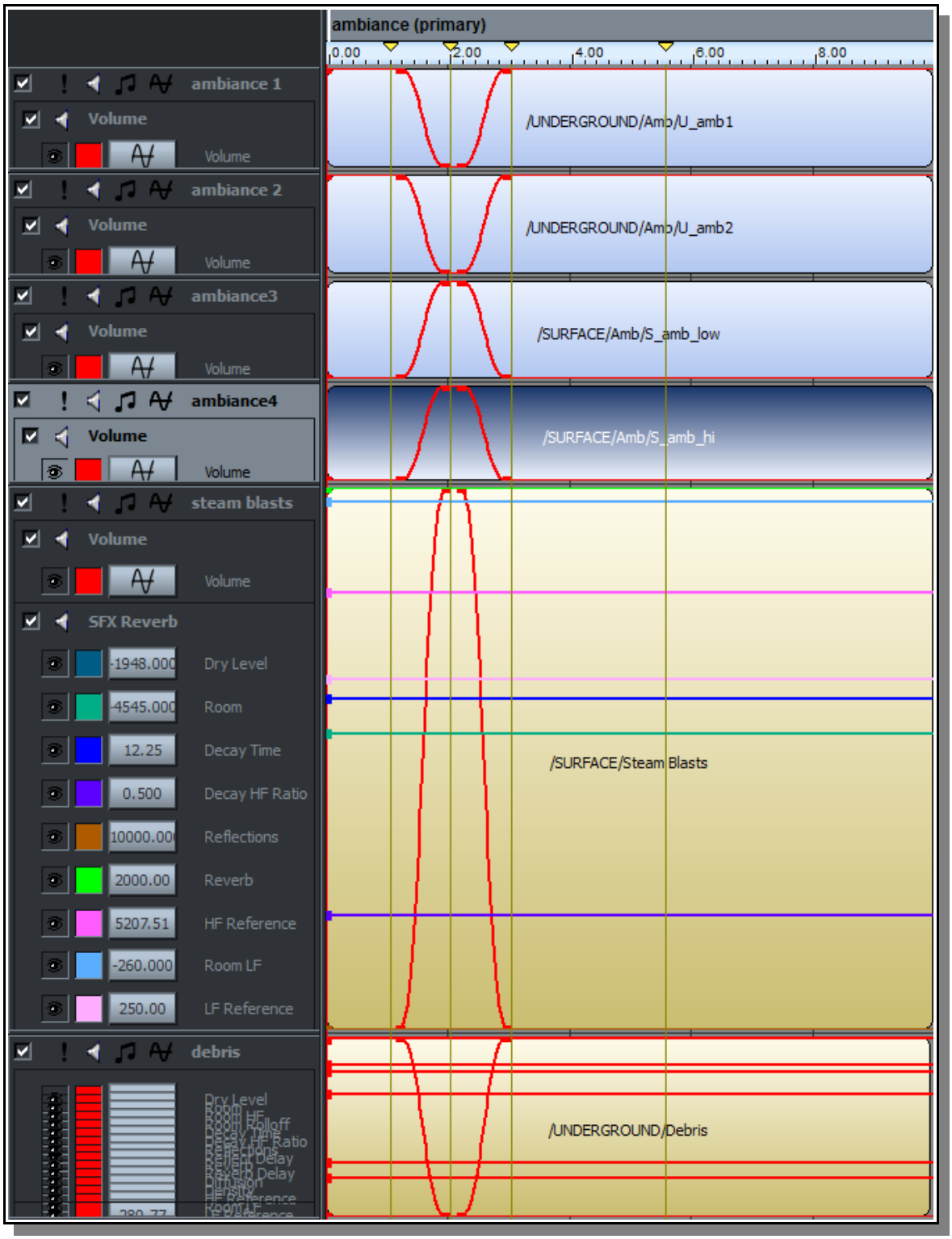
Pour l'ambiance de la surface, la méthode a été sensiblement la même, sauf que les deux Sound Definitions ne contiennent pas le même son. De plus, les 12 sons de débris sont remplacés ici par 7 sons de relâchements de vapeur.

Voici un exemple:

♫ **Amb_Surface.mp3**

- ***Transitions entre les deux ambiances***

Gestion des transitions entre les deux ambiances



Il a fallu définir plusieurs états pour gérer les transitions entre l'ambiance souterraine et l'ambiance en surface:
-SURFACE: ambiance sonore de surface.
-SOUS-SOL: ambiance sonore souterraine.
-SOUS-SOL → SURFACE: transition de l'ambiance souterraine à l'ambiance de surface.
-SURFACE → SOUS-SOL: transition de l'ambiance de surface à l'ambiance souterraine.

Sur l'image, les 3 premiers Sustain Points délimitent les 4 états possibles. Grâce à une courbe de volume identique à celle de l'ambiance de la surface, la piste des sons de relâchements de vapeur accompagne cette dernière. De la même manière, les sons de débris sont liés à l'ambiance souterraine.

Note: les paramètres de la reverb utilisée (SFX Reverb) sur les sons de relâchements de vapeur et de débris ne changent pas de valeurs au cours du jeu et j'aurais donc pu faire différemment, et appliquer, en amont, une reverb à ces sons, via le séquenceur ou l'éditeur audio. J'ai fait ainsi car j'utilise la même reverb sur plusieurs sons du jeu, dont la musique, et dans ce cas-là, les paramètres de la reverb varient en temps réel selon les actions du joueur. J'ai donc voulu utiliser la même reverb (avec des paramètres assez proches) pour donner une cohérence à l'ensemble de la bande son, au détriment de la charge CPU (la SFX Reverb fait partie des effets les plus gourmands de FMOD).

Les transitions entre les deux ambiances se font donc par un crossfade assez rapide.

Voici un exemple:

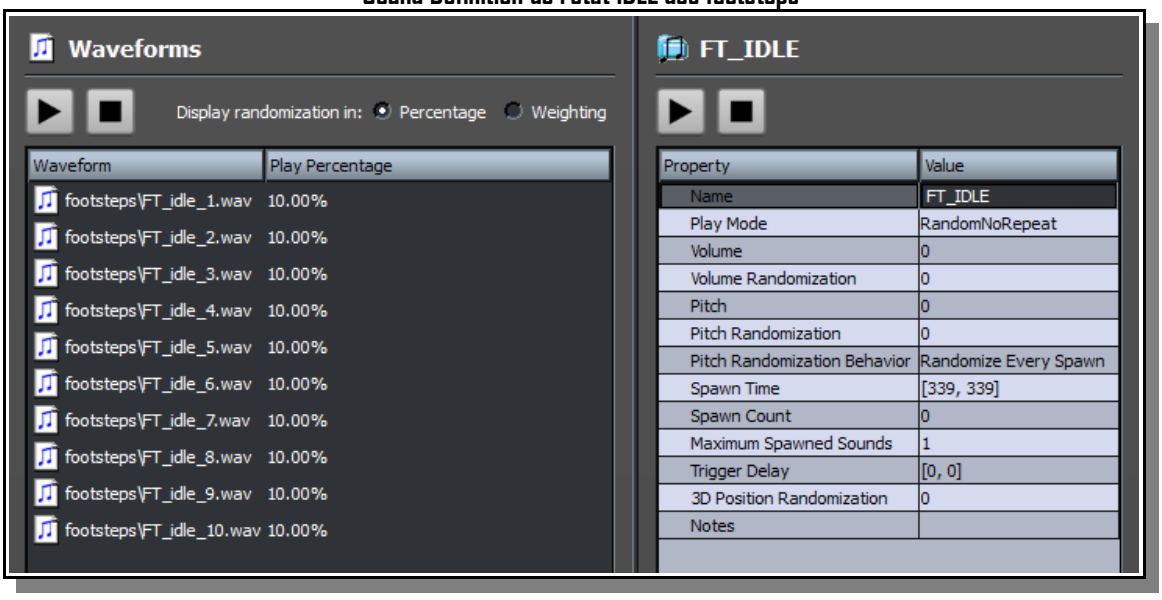
♫ **Amb_Transitions.mp3:** enchaînement des états (dans l'ordre) **SOUS-SOL, SOUS-SOL → SURFACE, SURFACE, SURFACE → SOUS-SOL, SOUS-SOL.**

- ***Footsteps***

A la surface, le joueur se déplace en courant. Il a donc fallu faire un système de bruits de pas, que l'on puisse déclencher et arrêter à volonté, de façon cohérente. Pour ce faire, j'ai défini 3 états:
-ON: début de la course (accélération).
-IDLE: course à vitesse constante.
-OFF: fin de la course et arrêt du personnage-joueur (décélération).

État ON:
Cette Sound Definition est constituée de 3 sons de début de course, chacun formé par 2 paires droite/gauche (pied droit, puis pied gauche). Un des sons est tiré aléatoirement, sans répétition (RandomNoRepeat), à chaque fois que le joueur commence à courir.

Sound Definition de l'état IDLE des footsteps



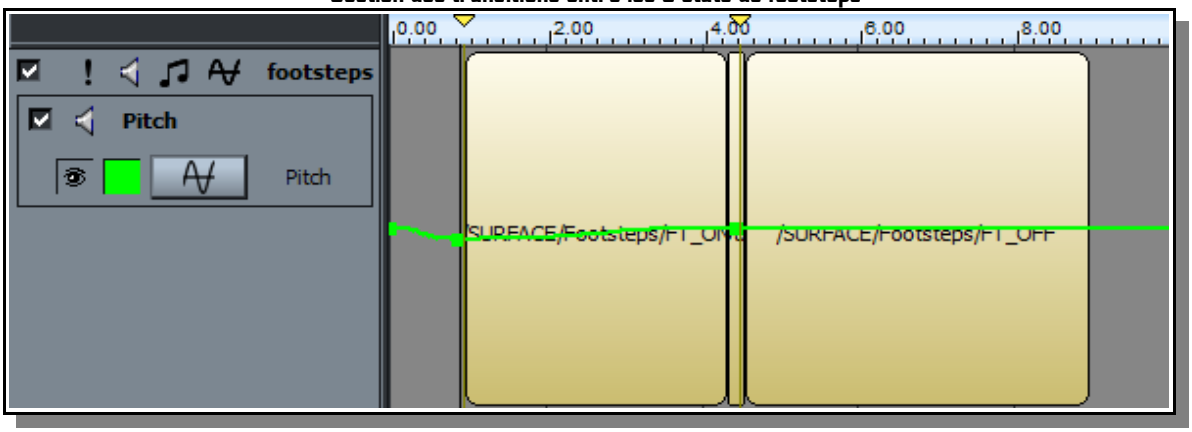
État IDLE:

Ici, la Sound Definition comprend 10 sons différents de course à vitesse constante, chacun étant formé par 1 paire gauche/droite. Comme précédemment, ils sont tirés aléatoirement, sans répétition. Là, par contre, la Sound Definition est jouée en boucle jusqu'à ce que le joueur s'arrête de courir et qu'on passe à l'état suivant. Pour caler rythmiquement les couples droite/gauche, j'ai ajouté 339 ms de battements entre 2 sons (Spawn Time).

État OFF:

Cette dernière Sound Definition est formée par 3 sons de fin de course d'un nombre de pas distinct. Un des sons est tiré aléatoirement à chaque fois que le joueur arrête de courir (autrement dit, d'appuyer sur la touche en question). Pour une question de calage rythmique avec les sons de l'état précédent (**IDLE**), j'ai dû ajouter un léger retard de 59 ms (Trigger Delay).

Gestion des transitions entre les 3 états de footsteps



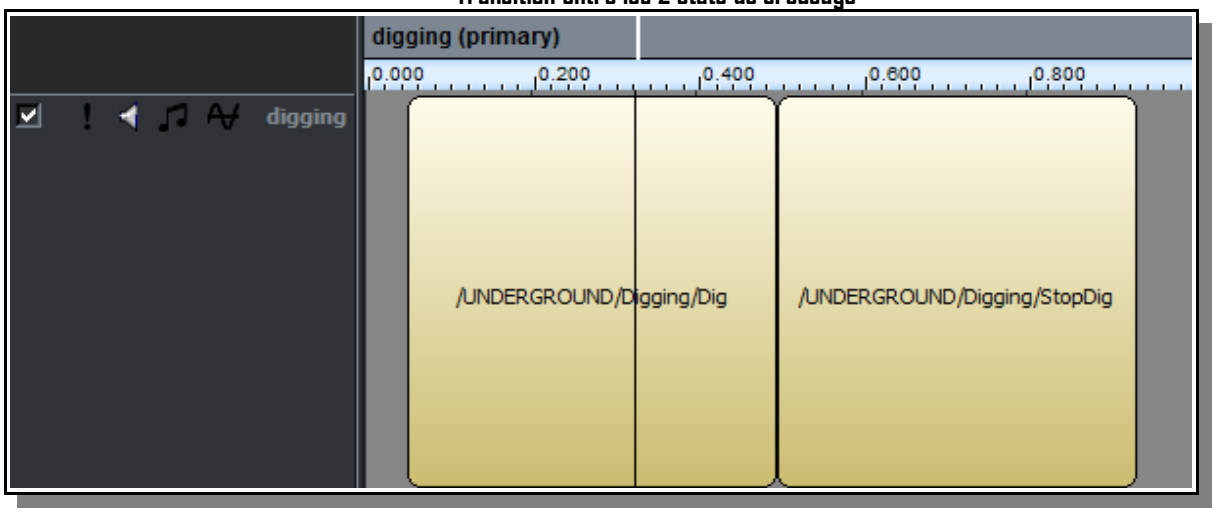
Pour définir et gérer les 3 états, j'ai ajouté 2 Sustain Points à la timeline. Ainsi, lorsque le joueur appuie sur une des touches qui lui permettent de courir (flèches directionnelles gauche et droite), l'état **ON** est déclenché, ce qui a pour effet de lire un des 3 sons de début de course. Immédiatement après, on passe à l'état **IDLE**. Ici, le programme attend la fin du son précédent pour lancer la boucle aléatoire des sons de course à vitesse constante. Enfin, lorsque le joueur arrête de courir (qu'il relâche la touche pressée), le curseur de la timeline enchaîne sur l'état **OFF**, qui a pour effet de déclencher un des 3 sons de fin de course (une fois que le son joué précédemment est terminé). J'ai ajouté une courbe de pitch pour amplifier légèrement le phénomène d'accélération en début de course (état **ON**).

Voici quelques exemples:

- **Footsteps_1.mp3**
- **Footsteps_2.mp3**
- **Footsteps_3.mp3**
- **Footsteps_4.mp3:** ici, j'ai rajouté, d'un simple clic, l'ambiance de surface.

- ***Creusage dans le sous-sol***

Transition entre les 2 états de creusage



Pour sonoriser l'action de creuser le sous-sol avec la pelle, il y a deux types de sons différents, représentés par deux états:
-DIGGING: creusage en cours.
-STOP: arrêt du creusage.

État DIGGING:
Composée de 6 sons représentant chacun un unique coup de pelle, la Sound Definition associée à cet état « pitche » également aléatoirement chaque son (Pitch Randomization). Pour le reste, elle fonctionne comme la Sound Definition de l'état **IDLE** des footsteps.

État STOP:
Ici, 3 sons différents pour illustrer l'action d'arrêter de creuser. On entend le personnage-joueur retirer sa pelle de la terre et cette dernière tomber à ses pieds. Pitch aléatoire également.

La transition entre les 2 états fonctionne de la même manière que pour les footsteps, sauf qu'ici, il n'y a pas de Sustain Points et c'est au programmeur d'appliquer la bonne valeur au paramètre lié à cette transition (« digging » sur l'image). Sur la capture d'écran, ce serait par exemple:
digging = 0,2 → État **DIGGING**.
digging = 0,6 → État **STOP**.
Pour ce genre de sons, on préfèrera quand même la technique des Sustain Points, qui est plus pratique tant pour le programmeur que pour le sound designer.

Voici quelques exemples:

♫ **Digging_1.mp3**

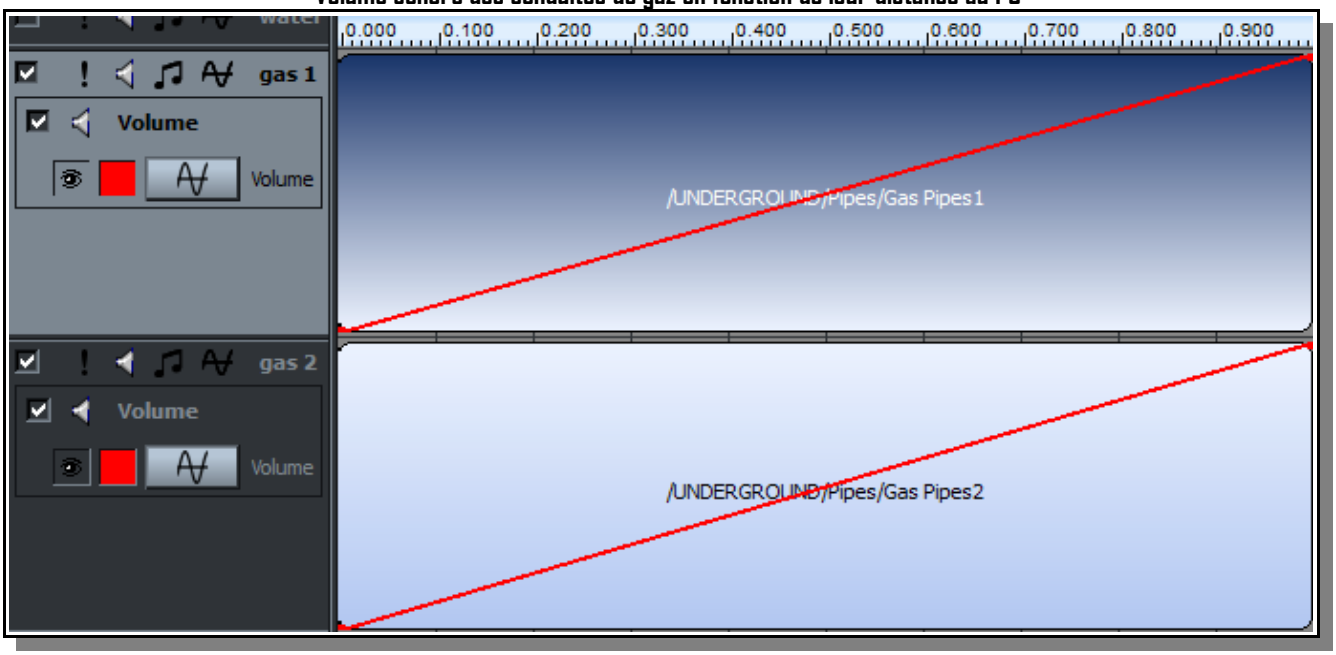
♫ **Digging_2.mp3**

♫ **Digging_3.mp3**

♫ **Digging_4.mp3:** ici, j'ai rajouté, d'un simple clic, l'ambiance souterraine.

- ***Conduites de gaz***

Volume sonore des conduites de gaz en fonction de leur distance au PJ



Le son des conduites de gaz étant bouclé, il a fallu utiliser le même procédé que pour l'ambiance souterraine, c'est à dire lire simultanément les deux mêmes sons, avec un décalage temporel, pour éviter de les entendre boucler. Pour ce faire, j'ai appliqué un retard variable (aléatoire) au second son de 1000 à 6000 ms.
Ensuite, comme les courbes sur la capture d'écran l'indiquent, il s'agit juste d'établir une gestion du volume en fonction de la distance entre les conduites de gaz et le PJ (personnage-joueur). Le programmeur a juste à lier la variable de distance au paramètre de volume grâce à la fonction *FMOD_EventParameter_SetValue()*, issue de la librairie FMOD. Par contre la variable doit être modifiée puisque lorsque la distance augmente, le volume diminue, et inversement.
Ainsi, schématiquement: volume = -distance

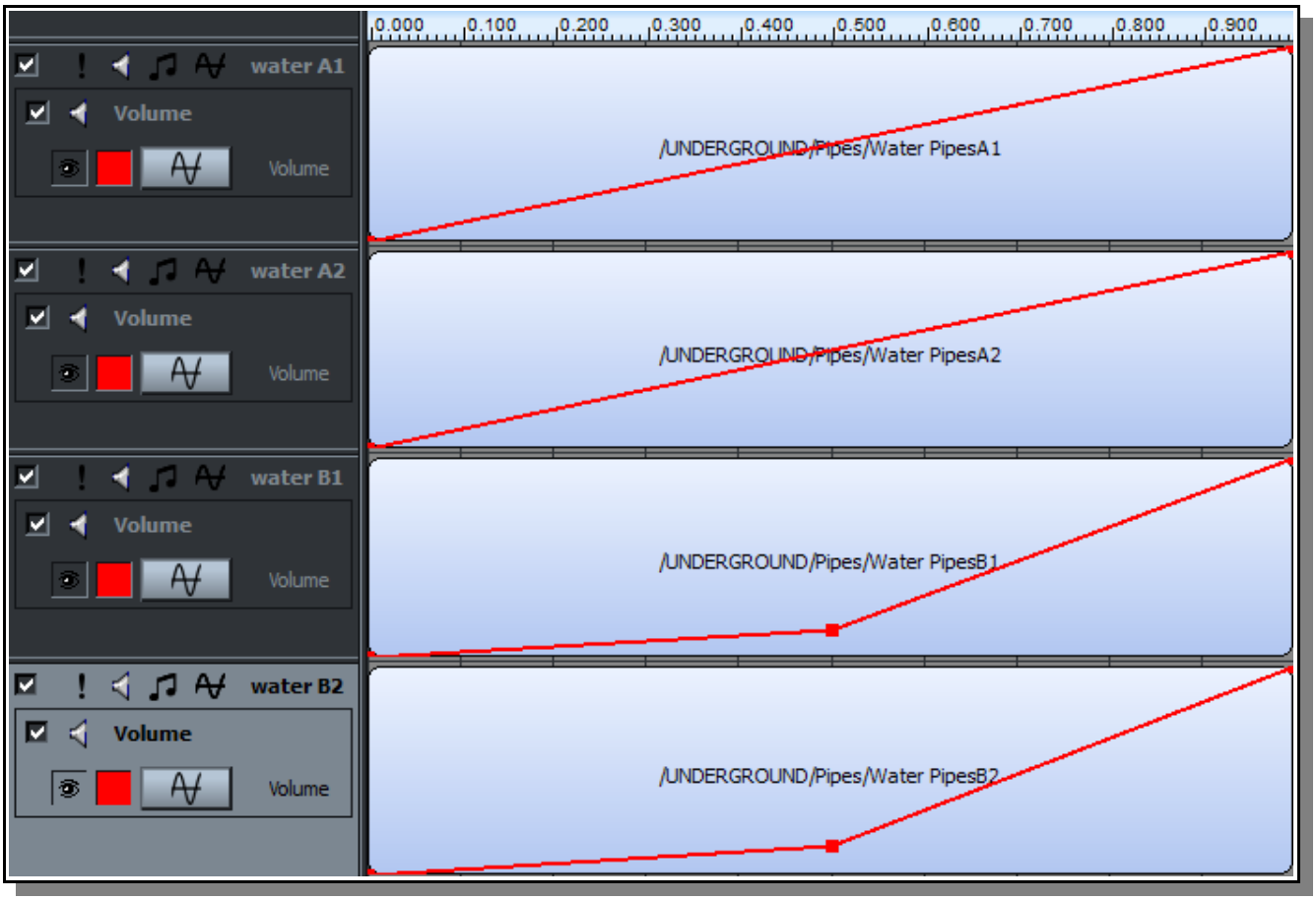
Voici un exemple:

♫ **Gas.mp3:** ici, j'ai simulé une phase de jeu où le joueur s'approche, puis s'éloigne, d'une conduite de gaz (ambiance souterraine (avec debris) + creusage + conduite de gaz). Le son de la conduite de gaz est ici facilement perceptible, mais il faut imaginer qu'il y a, en temps normal, la musique en plus, qui a tendance à couvrir le bruit des dangers. Il faut que le joueur soit concentré pour pouvoir distinguer tous les éléments sonores.

- ***Canalisations d'eau***

Pour les canalisations d'eau, c'est exactement le même principe que pour les conduites de gaz, sauf qu'ici il y a 2 sons bouclés différents, soit 4 Sound Definitions. La courbe de volume diffère légèrement pour les 2 sons.

Volume sonore des canalisations d'eau en fonction de leur distance au PJ



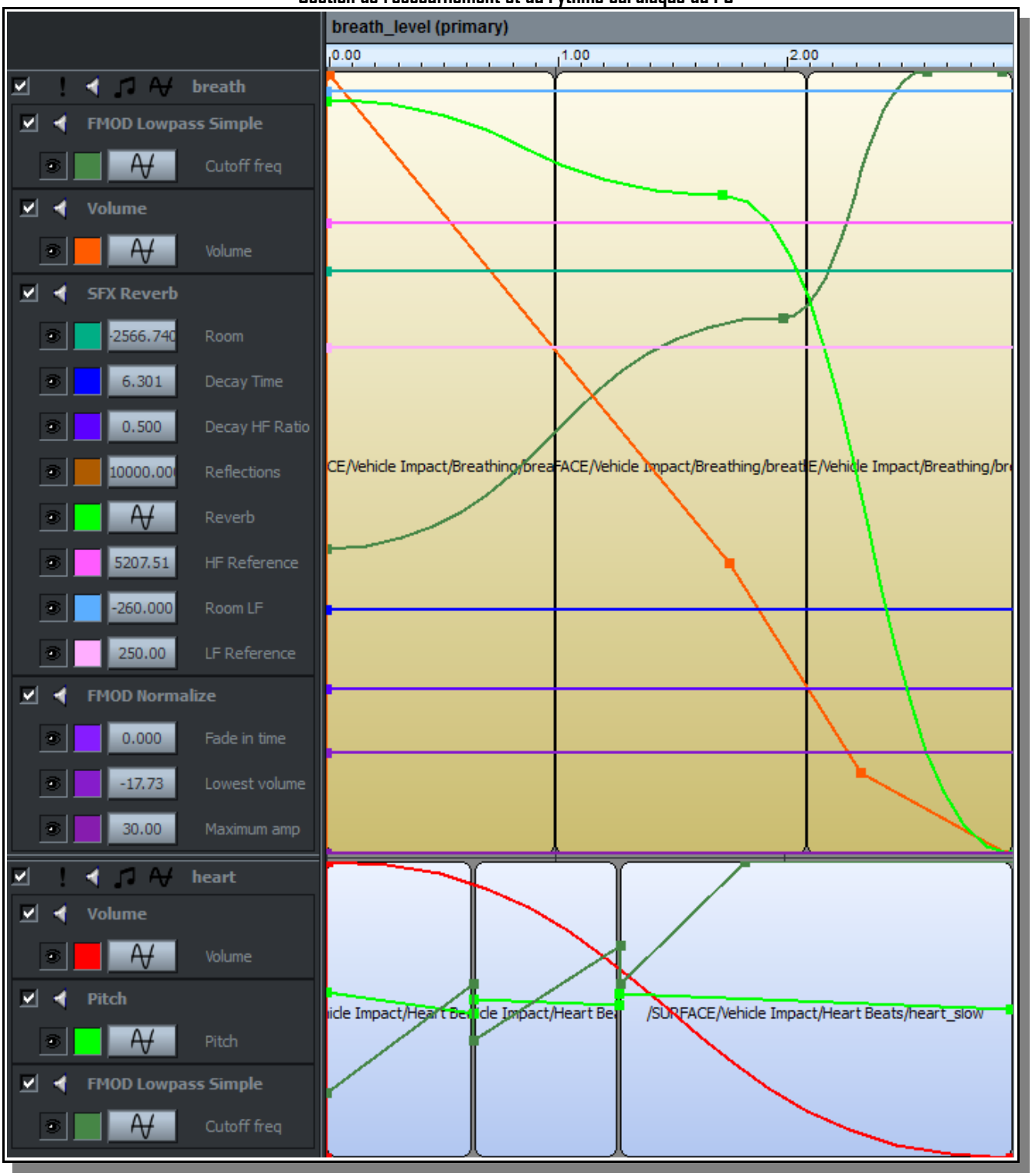
Voici un exemple:

♫ **Water.mp3:** ici, j'ai simulé une phase de jeu dans laquelle le joueur s'approche puis s'éloigne d'une canalisation d'eau, de la même manière que pour l'exemple sonore précédent avec la conduite de gaz.

- ***Essoufflement, battements cardiaques et effet d'occlusion***

Comme indiqué dans la partie Mécaniques de jeu, l'impact entre un véhicule et le PJ n'implique pas forcément la mort du joueur. Il y a effectivement une probabilité d'une chance sur cinq pour que le véhicule ne le frappe pas de plein fouet et qu'il soit juste sonné quelques secondes. Concrètement, cela se traduit au niveau du son par la présence de la respiration et de la pulsation cardiaque du PJ et par un fort effet d'occlusion sur tous les sons environnants. Autrement dit, c'est proche de l'acouphène, avec le sifflement en moins.

Gestion de l'essoufflement et du rythme cardiaque du PJ



Respiration:
L'essoufflement est découpé en 3 Sound Definitions, avec, dans l'ordre, une respiration rapide, moyenne et lente. Chaque Sound Definition contient entre 3 et 6 sons (un son comportant une inspiration et une expiration) qui sont tirés aléatoirement en boucle. Le calage rythmique des sons se fait via des Spawn Times dans chaque Sound Definition et un très léger pitch aléatoire est ajouté à chaque son (0,03 octave).
Ensuite, on fait en sorte que les 3 Sound Definitions se suivent et que l'arrivée du curseur de la timeline sur une nouvelle fréquence respiratoire n'ait pas pour conséquence de couper le son en cours d'exécution (Start Mode: Wait for previous).
Comme indiqué sur la capture d'écran, j'ai appliqué plusieurs effets à ces sons de respirations:
-FMOD Lowpass Simple: un passe-bas qui va s'atténuer progressivement au fur et à mesure que le PJ récupèrera.
-une courbe de volume.
-SFX Reverb: de la même manière que le passe-bas, la reverb s'atténue avec la récupération du PJ. Le passe-bas et la reverb sont responsables de l'effet d'occlusion.
-FMOD Normalize: normaliser le son de respiration permet de compenser la perte en volume sonore due au passe-bas.

Battements cardiaques:
Le rythme cardiaque est globalement géré de la même façon que la respiration avec 3 vitesses de base, sauf qu'ici, chaque Sound Definition ne contient qu'un unique son bouclé. Naturellement, contrairement à la respiration, les battements cardiaques ne sont pas pitchés aléatoirement (ce serait incohérent). Du côté des effets, si on retrouve une courbe de volume et un passe-bas assez similaires à ce qu'on a vu avec la respiration, le pitch, quant à lui, est spécifique aux pulsations cardiaques. Il permet de lisser les transitions entre les 3 fréquences cardiaques de base. La courbe du passe-bas est en dents de scie car il faut compenser l'effet du pitch.

La vitesse du curseur de la timeline est variable, car le PJ ne récupère pas aussi vite à chaque fois. Le problème c'est qu'on ne peut pas faire varier aléatoirement la vitesse (Velocity) du curseur de la timeline (Parameter) dans FMOD Designer. Pour cela, il faut encore encore une fois passer par le code source du jeu avec la fonction correspondante de la librairie FMOD:
FMOD_EventParameter_SetVelocity(0, random_range(0.1, 0.5); // (syntaxe Game Maker)
→ Affecte aléatoirement une valeur entre 0,1 et 0.5 à la variable de Velocity du Parameter d'index 0 (le paramètre qui nous intéresse).

Voici justement quelques exemples avec des valeurs différentes de Velocity:

♫ **Breathing_1.mp3:** Velocity = 0,15 → récupération lente.

♫ **Breathing_2.mp3:** Velocity = 0,275 → récupération moyennement rapide.

♫ **Breathing_3.mp3:** Velocity = 0,4 → récupération rapide.

Idéalement, il faudrait que l'effet d'occlusion soit appliqué en temps réel à tous les sons du jeu et pas seulement à la respiration et aux battements du cœur. Seulement, FMOD Designer n'est pas fait pour appliquer un même effet à tous les sons d'un même jeu. Dans ce cas là, c'est plus simple d'utiliser les fonctions proposées par la librairie de FMOD et d'appliquer le même effet au master ou à un groupe de sons dans le code source du jeu. Le procédé est expliqué plus loin dans la partie Musique.

• ***Système de communication vocale***

En mode multijoueur, pour simuler le brouillage radio dû aux véhicules automatisés des dispositifs de communication vocale des dikophones des 2 PJ, on peut utiliser simultanément quatre procédés:
-appliquer un band-pass aux voix des joueurs entre 300 Hz et 3.5 kHz,
-ajouter l'effet FMOD Distortion pour saturer légèrement les voix,
-ajouter des sons de parasites,
-baisser le volume des voix.

Le tout sera finement paramétré pour que la voix du coéquipier soit à peine compréhensible lorsqu'elle est brouillée en surface.

• ***Sound design et gameplay***

Via les ambiances et les effets sonores, le jeu envoie au joueur plusieurs informations, sous forme, notamment, de stimuli-réponses et de feedbacks:
-la proximité d'un danger: canalisations d'eau, conduites de gaz et véhicules automatisés.
-l'arrêt et le redémarrage d'un véhicule.
-la position du PJ: en ayant les dangers fixes (canalisations et conduites) et les deux ambiances distinctes comme repères (grâce au changement d'ambiance, le joueur peut estimer sa position verticalement).
-la santé du PJ: grâce à l'effet d'occlusion, à la respiration et aux battements du cœur lorsqu'il est blessé.
-plusieurs feedbacks audios en réponse aux actions du joueur: déplacements (course et creusage) et impact avec un véhicule, une canalisation ou une conduite (sons non produits).

Au fur et à mesure de sa progression dans le jeu, le joueur développera une écoute causale et sera ainsi de plus en plus apte à réagir vite face aux dangers et apprendra peu à peu à se déplacer en aveugle.

• ***Esthétique globale du sound design***

Le concept esthétique du sound design du jeu est de proposer une dichotomie sonore entre l'ambiance du sous-sol et celle de la surface. D'une part, le sous-sol nous propose un environnement confiné, servi par une atmosphère quelque peu angoissante, mais relativement discrète, et d'autre part, la surface, avec sa circulation bruyante et ses machines alimentant les buildings, plonge le joueur dans un sentiment d'hostilité, d'immensité urbaine et de permanence du danger.

Les différences de gameplay dans ces deux environnements dissemblables vient accentuer ce caractère dichotomique. En effet, en surface, les passages incessants de véhicules et l'oppressant volume sonore poussent le joueur à agir vite, alors que les phases en sous-sol incitent le joueur à progresser avec prudence, puisque rien ne peut lui arriver dans cette zone tant qu'il reste immobile.

♫ **Demo_1.mp3:** ici, j'ai fait une simulation sonore du niveau type présenté dans la partie Mécaniques de jeu. Une grande partie des sons est présente, sauf la musique, qui rappelons-le, permet de localiser le nid des créatures (cf. les phases du niveau en écoutant l'extrait).

□ Musique

• ***Aspect musical du gameplay***

La musique, dans le gameplay de Dikotomia, a une place fondamentale:
-localisation du nid des créatures: → estimation de la distance entre le PJ et le nid via le tempo et le volume de la musique.
-position verticale du PJ: → estimation de la profondeur atteinte par le PJ via la réverbération de la musique.
-brouillage des signaux des créatures: → en surface, la musique est brouillée via un pitch shift à fluctuations rapides, illustrant les interférences créées par les véhicules automatisés.

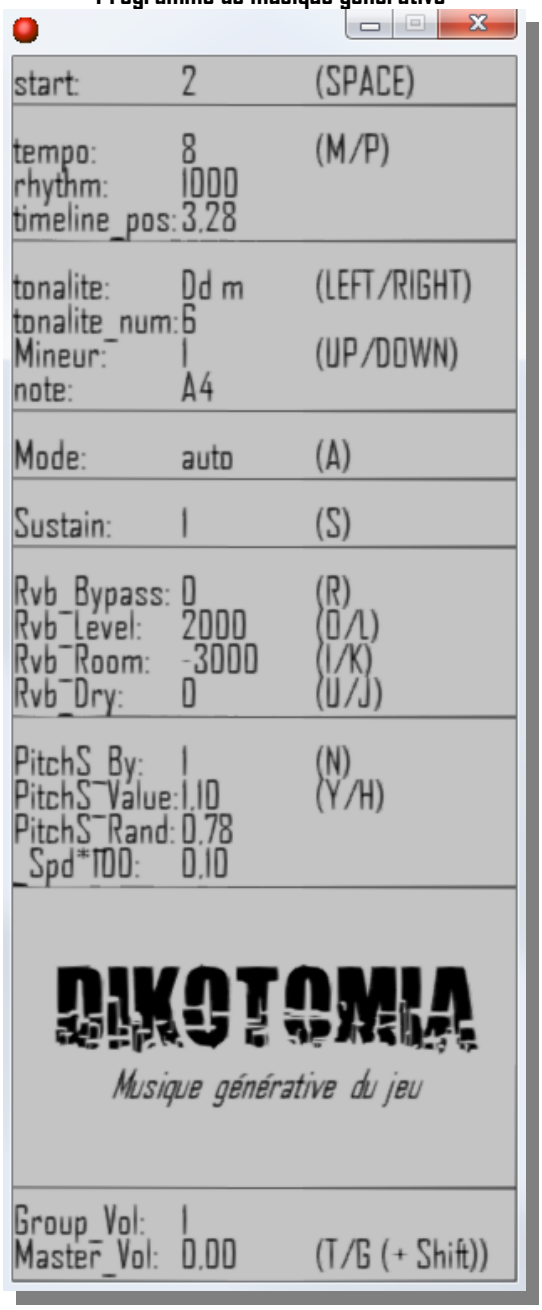
L'univers musical de Dikotomia réutilise les concepts traditionnels de mélodie et d'harmonie qui permettent la création d'une musique un minimum immersive et « easy-listening », mais qui n'ont aucun impact sur le gameplay. Seules des variations de timbre, de volume et de tempo sont impliquées dans le gameplay. Autrement dit, la mélodie et l'harmonie servent l'esthétisme alors que le timbre et le tempo servent le gameplay: c'est le caractère dichotomique de la musique du jeu.

• ***Adaptabilité et générativité musicale***

Comme nous le verrons bientôt plus en détails, la musique de Dikotomia est adaptative, car son évolution dépend des actions du joueur. De plus, elle est générative, parce qu'elle est stochastique (soumise à l'aléatoire). En effet, la musique n'est pas composée, mais elle est générée directement dans le jeu, par le programme, en temps réel.

Le caractère stochastique de la musique s'applique à la mélodie, à l'harmonie et aux formules rythmiques (esthétisme) alors que son aspect déterministe est lié à des états et des évènements du gameplay.

Programme de musique générative



Note: vous pouvez télécharger une version compilée du programme de générativité musicale ici: **http://martin.bussy.free.fr/ENJMIN/Concours/**

- ***Méthode de travail***

La programmation de la musique générative de Dikotomia s'est faite sur le logiciel Game Maker, à l'aide de la librairie de fonctions FMOD.

Note: tous les fichiers musicaux joints au dossier sont issus directement du programme écrit sur Game Maker, qui permet de générer la musique en temps réel et de la gérer avec les effets sonores et les ambiances.

- ***Système rythmique***

Note: je ne m'attarderai pas sur des détails de programmation qui nous importent peu ici, comme l'initialisation de FMOD, le chargement des sons en mémoire, le gestionnaire d'évènements ou l'interface visuelle du programme.

Tout d'abord, j'ai utilisé le séquenceur Cubase et l'instrument virtuel Miroslav Philarmonik pour exporter une unique note de piano. J'ai ensuite conçu un système métrique simple, basé sur des mesures de 4/4, grâce au système de Time Lines inclue dans Game Maker.

Génération d'une nouvelle cellule rythmique

```
// Réinitialisation de la variable de rythme
global.rhythm = "1";

// Détermination du nouveau rythme
while (string_length(global.rhythm) < 4)
{
    global.rhythm = string_insert(string(irandom(1)), global.rhythm, 2);
}
```

Le principe est simple: il s'agit d'associer une variable contenant 4 caractères à une mesure. Chaque caractère représente un temps de la mesure et peut prendre comme valeur 0 ou 1.
Le programme, à chaque temps (correspondant à chaque Step de la Time Line), va lire la variable et déterminer si oui ou non il faut jouer le temps en question.
Ainsi, si global.rhythm = « 1111 », tous les temps de la mesure seront joués.
Pour une question de musicalité, j'ai fait en sorte que le 1er temps soit toujours joué, d'où la 2ème ligne de code: global.rhythm = « 1 »;
La boucle qui suit permet de définir les 3 autres caractères de la variable: 0 ou 1, et ce, de manière équiprobable.
On a donc au total 8 combinaisons possibles (2x2x2), soit 8 formules rythmiques.
Pour faire varier le tempo, il suffit de changer la vitesse de la Time Line.

♫ **Music_Rhythm.mp3:** démonstration du système rythmique en bloquant le programme sur la même note et en désactivant le système de tonalités. Augmentation du tempo tout au long de l'extrait.

Systématisme du démarrage et de l'arrêt de la musique sur le 1er temps

```
if (global.start = 0) && (timeline_position > 0) && (timeline_position < 1)
{
    timeline_running = 0;
}

if (global.start = 1)
{
    timeline_position = 0;
    timeline_running = 1;
    global.start = 2;
}
```

Pour optimiser la musicalité du programme, j'ai fait en sorte que l'arrêt de la musique par le joueur implique un arrêt effectif au 1er temps de la mesure suivante. Ainsi, la musique ne s'arrête jamais au milieu de la mesure. De la même manière, le déclenchement de la musique se fera toujours en début de mesure.

- ***Système mélodique***

Tirage de la note suivante

```
1  // Changement ou non de note selon le rythme
2  if (string_char_at (global.rhythm,2) = "1")
3  {
4      // Arrêt de toutes les notes en cours de lecture si Sustain désactivé
5      if (global.sustain = 0) FMODAllStop ();
6      global.play = 1;
7
8      // Détermination de la note suivante
9      if (string_lettersdigits(global.note) = "C3")
10     {
11         global.note = irandom(3);
12         if (global.note = 0)
13         {
14             global.note = "C3";
15         };
16         else if (global.note = 1)
17         {
18             global.note = "F3";
19         };
20         else if (global.note = 2)
21         {
22             global.note = "E3";
23         };
24         else if (global.note = 3)
25         {
26             global.note = "D3";
27         };
28     };
29     else if (string_lettersdigits(global.note) = "D3")
30     {
```

La ligne 2 du code fait référence au système de cellules rythmiques vu précédemment. Il s'agit ici du 2ème temps de la mesure, donc si le second caractère de la variable global.rhythm est égal à 1, le programme exécute le code qui suit et va donc jouer une note.
Comme on peut le voir à partir de la ligne 9, si la note précédente est un Do3, selon le résultat du tir aléatoire, la note jouée sera un Do3, un Fa3, un Mi3 ou un Ré3. Ici, on est dans un cas d'équiprobabilité, mais ce n'est pas toujours ce qui se produit.
Si la note précédente n'est pas un Do3, le programme ignore cette partie et passe à la suite pour tester une nouvelle condition: est-ce que la note précédente est un Ré3? Et ainsi de suite, jusqu'au Do6, soit 3 octaves. J'aurais pu programmer un système plus élaboré me permettant d'éviter les 483 lignes qui suivent pour toutes les notes des 3 octaves, mais la méthode choisie m'a permis une grande souplesse et j'ai ainsi eu la possibilité de choisir au cas par cas les notes possibles selon la note précédente.
De cette manière, j'ai pu faire en sorte que l'octave 4 soit plus jouée que les octaves 3 et 5. Ainsi, les notes extrêmes sont rares et le joueur les savourera sans doute davantage.
Au niveau des intervalles, les plus fréquents sont la seconde et la tierce, mais on peut aussi, plus rarement, entendre des quartes. Les secondes est les tierces peuvent être mineures ou majeures et les quartes peuvent être diminuées, justes, ou augmentées, selon la tonalité.

♫ **Music_Melody.mp3:** démonstration du système mélodique en bloquant le système de tonalités.

- ***Système harmonique***

Ici, il a fallu intégrer au programme le système du tempérament égal et ses 12 tonalités majeures et mineures.

Définition des 24 tonalités

```
if (global.tonalite_num = 0) && (global.minor = 0) global.tonalite = "C M";
else if (global.tonalite_num = 0) && (global.minor = 1) global.tonalite = "A m";

else if (global.tonalite_num = -1) && (global.minor = 0) global.tonalite = "F M";
else if (global.tonalite_num = -1) && (global.minor = 1) global.tonalite = "D m";
else if (global.tonalite_num = -2) && (global.minor = 0) global.tonalite = "Bb M";
else if (global.tonalite_num = -2) && (global.minor = 1) global.tonalite = "G m";
else if (global.tonalite_num = -3) && (global.minor = 0) global.tonalite = "Eb M";
else if (global.tonalite_num = -3) && (global.minor = 1) global.tonalite = "C m";
else if (global.tonalite_num = -4) && (global.minor = 0) global.tonalite = "Ab M";
else if (global.tonalite_num = -4) && (global.minor = 1) global.tonalite = "F m";
else if (global.tonalite_num = -5) && (global.minor = 0) global.tonalite = "Db M";
else if (global.tonalite_num = -5) && (global.minor = 1) global.tonalite = "Bb m";
else if (global.tonalite_num = -6) global.tonalite_num = 6;

else if (global.tonalite_num = 1) && (global.minor = 0) global.tonalite = "G M";
else if (global.tonalite_num = 1) && (global.minor = 1) global.tonalite = "E m";
else if (global.tonalite_num = 2) && (global.minor = 0) global.tonalite = "D M";
else if (global.tonalite_num = 2) && (global.minor = 1) global.tonalite = "B m";
else if (global.tonalite_num = 3) && (global.minor = 0) global.tonalite = "A M";
else if (global.tonalite_num = 3) && (global.minor = 1) global.tonalite = "Fd m";
else if (global.tonalite_num = 4) && (global.minor = 0) global.tonalite = "E M";
else if (global.tonalite_num = 4) && (global.minor = 1) global.tonalite = "Cd m";
else if (global.tonalite_num = 5) && (global.minor = 0) global.tonalite = "B M";
else if (global.tonalite_num = 5) && (global.minor = 1) global.tonalite = "Gd m";
else if (global.tonalite_num = 6) && (global.minor = 0) global.tonalite = "Fd M";
else if (global.tonalite_num = 6) && (global.minor = 1) global.tonalite = "Dd m";
else if (global.tonalite_num = 7) global.tonalite_num = -5;
```

Choix des altérations en fonction de la tonalité sélectionnée (ici, Ré bémol Majeur)

```
    else if (global.tonalite = "Db M")
    {
        if (global.note = "C3") FMODSoundPlay (global.C3_50);
        if (global.note = "D3") FMODSoundPlay (global.Cd3_50);
        if (global.note = "E3") FMODSoundPlay (global.Dd3_50);
        if (global.note = "F3") FMODSoundPlay (global.F3_50);
        if (global.note = "G3") FMODSoundPlay (global.Fd3_50);
        if (global.note = "A3") FMODSoundPlay (global.Gd3_50);
        if (global.note = "B3") FMODSoundPlay (global.Ad3_50);

        if (global.note = "C4") FMODSoundPlay (global.C4_50);
        if (global.note = "D4") FMODSoundPlay (global.Cd4_50);
        if (global.note = "E4") FMODSoundPlay (global.Dd4_50);
        if (global.note = "F4") FMODSoundPlay (global.F4_50);
        if (global.note = "G4") FMODSoundPlay (global.Fd4_50);
        if (global.note = "A4") FMODSoundPlay (global.Gd4_50);
        if (global.note = "B4") FMODSoundPlay (global.Ad4_50);

        if (global.note = "C5") FMODSoundPlay (global.C5_50);
        if (global.note = "D5") FMODSoundPlay (global.Cd5_50);
        if (global.note = "E5") FMODSoundPlay (global.Dd5_50);
        if (global.note = "F5") FMODSoundPlay (global.F5_50);
        if (global.note = "G5") FMODSoundPlay (global.Fd5_50);
        if (global.note = "A5") FMODSoundPlay (global.Gd5_50);
        if (global.note = "B5") FMODSoundPlay (global.Ad5_50);
        if (global.note = "C6") FMODSoundPlay (global.C6_50);
    }
```

Tout d'abord, j'ai programmé le cycle des quintes, comme on peut le voir sur la première capture d'écran. Toutes les tonalités sont dans l'ordre, du côté des dièses comme du côté des bémols.
Ensuite pour chaque tonalité, j'ai dû associer les altérations correspondantes, comme c'est indiqué sur la seconde capture d'écran. Pour Ré bémol Majeur, par exemple, on a Sib, Mib, Lab, Réb et Solb. Ainsi, à chaque note, j'ai lié les sons correspondants. Pour « D3 », on a « Cd3 », soit un Ré bémol à la place du Ré bécarre. Pareil pour « E3 » qui devient « Dd3 », et ainsi de suite.

Changement aléatoire de tonalité

```
1  // Random tonalités
2
3  if (global.mode = "auto")
4  {
5      global.rand = irandom(59); // le 0 est inclu, on a donc 60 possibilités
6      if (global.rand = 0) || (global.rand = 1)
7      {
8          global.tonalite_num += irandom(2)-1;
9      }
10     else if (global.rand = 2)
11     {
12         if (global.minor = 0) global.minor = 1;
13         else global.minor = 0;
14     }
15 }
```

Le mode « auto » permet une gestion automatique des tonalités alors que le mode « manuel » offre à l'utilisateur le choix des modulations, en temps réel (cf « Mode: » sur la capture d'écran du programme en cours d'exécution page 24).
Sur l'image ci-dessus, le changement de tonalité se fait de manière automatique (global.mode = « auto »).
-on tire aléatoirement un nombre (variable global.rand) entre 0 et 59 inclus (ligne 5).
-si ce nombre est égal à 0 ou 1 (ligne 6), un nouveau tir aléatoire est fait et la musique module dans le sens des bémols (-1), dans le sens des dièses (+1) ou ne module pas (+0) → ligne 8.
-si global.rand est égal à 2 (ligne 10), la tonalité est « majorisée » (global.minor = 0) ou « minorisée » (global.minor = 1) selon si elle était majeure ou mineure.

Étude des probabilités:
(2/60)x(1/3) = 2/180 = 1/90
Ainsi, on a 1 chance sur 90 de changer de tonalité dans le sens des dièses à la fin de la mesure (idem dans le sens des bémols).

On a par contre 1 chance sur 60 de « majoriser » ou de « minoriser » la tonalité.

2/90+1/60 = (4+3)/180 = 7/180
Ce qui fait un total de 7 chances sur 180 de moduler (changer de tonalité) à chaque fin de mesure.

♫ **Music_Harmony.mp3:** démonstration du système harmonique. On peut entendre 5 modulations en tout.

- ***Éléments musicaux liés au gameplay***

Tempo:
La gestion du tempo, vu précédemment, permet d'associer la distance entre le PJ et le nid des créatures à la vitesse d'exécution de la musique.

Réverbération:
La reverb, quant à elle, est liée à la position verticale du PJ. En effet, plus celui-ci creuse profondément dans le sous-sol de Dikotomia et plus la musique est réverbérée.

Initialisation de la reverb

```
3  //SFX REVERB 18
4  global.reverb = FMODGroupAddEffect (1, 18);
5
6  //Dry Level //Mix level of dry signal in output in mB.  Ranges from -10000.0 to 0.0.  Default is 0.
7  FMODEffectSetParamValue(global.reverb,0,0);
8  //Room //Room effect level at low frequencies in mB.  Ranges from -10000.0 to 0.0.  Default is 0.0.
9  FMODEffectSetParamValue(global.reverb,1,-3000);
10 //Room HF //Room effect high-frequency level re. low frequency level in mB.  Ranges from -10000.0 to 0.0.  Default is 0.0.
11 FMODEffectSetParamValue(global.reverb,2,0);
12 //Room Rolloff //Like DS3D flRolloffFactor but for room effect.  Ranges from 0.0 to 10.0. Default is 10.0
13 FMODEffectSetParamValue(global.reverb,3,0);
14 //Decay Time //Reverberation decay time at low-frequencies in seconds.  Ranges from 0.1 to 20.0. Default is 1.0.
15 FMODEffectSetParamValue(global.reverb,4,6);
16 //Decay HF Ratio //High-frequency to low-frequency decay time ratio.  Ranges from 0.1 to 2.0. Default is 0.5.
17 FMODEffectSetParamValue(global.reverb,5,0.50);
18 //Reflections //Early reflections level relative to room effect in mB.  Ranges from -10000.0 to 1000.0.  Default is -10000.0.
19 FMODEffectSetParamValue(global.reverb,6,-10000);
20 //Reflect Delay //Delay time of first reflection in seconds.  Ranges from 0.0 to 0.3.  Default is 0.02.
21 FMODEffectSetParamValue(global.reverb,7,0);
22 //Reverb //Late reverberation level relative to room effect in mB.  Ranges from -10000.0 to 2000.0.  Default is 0.0.
23 FMODEffectSetParamValue(global.reverb,8,2000);
24 //Reverb Delay //Late reverberation delay time relative to first reflection in seconds.  Ranges from 0.0 to 0.1.  Default is 0.04.
25 FMODEffectSetParamValue(global.reverb,9,0);
26 //Diffusion //Reverberation diffusion (echo density) in percent.  Ranges from 0.0 to 100.0.  Default is 100.0.
27 FMODEffectSetParamValue(global.reverb,10,100);
28 //Density //Reverberation density (modal density) in percent.  Ranges from 0.0 to 100.0.  Default is 100.0.
29 FMODEffectSetParamValue(global.reverb,11,100);
30 //HF Reference //Reference high frequency in Hz.  Ranges from 20.0 to 20000.0. Default is 5000.0.
31 FMODEffectSetParamValue(global.reverb,12,5000);
32 //Room LF //Room effect low-frequency level in mB.  Ranges from -10000.0 to 0.0.  Default is 0.0.
33 FMODEffectSetParamValue(global.reverb,13,-200);
34 //LF Reference //Reference low-frequency in Hz.  Ranges from 20.0 to 1000.0. Default is 250.0.
35 FMODEffectSetParamValue(global.reverb,14,250);
```

Ci-dessus, on applique la reverb au groupe contenant tous les sons de piano (ligne 4) et on définit les paramètres initiaux de l'effet SFX Reverb. Une fois le programme lancé, le paramètre « Dry Level » est mis à jour en temps réel selon la position verticale du PJ.

♫ **Music_Reverb.mp3:** simulation d'une descente du joueur en profondeur puis de sa remontée vers la surface. Présence de l'ambiance souterraine et des sons de creusage.

Pitch Shift:
Effet responsable du brouillage du signal musical dû aux véhicules automatisés lorsque le PJ est en surface, le pitch shift permet de rajouter une contrainte gameplay au joueur lorsqu'il émerge du sous-sol. En surface, il peut courir, et se déplace ainsi beaucoup plus vite que sous-terre. Garder la musique dans ce cas là aurait eu pour effet de casser le gameplay en facilitant grandement la tâche du joueur, qui aurait contourné le plus possible le sous-sol.

La musique se voit donc ici appliquer, lorsque le PJ est en surface, un pitch shift à fluctuations rapides qui empêche le joueur d'utiliser cet élément de gameplay pour situer le nid des créatures. Ainsi, il est contraint de prévoir ses futurs déplacements en surface avant de sortir du sous-sol.

Initialisation et bypass du pitch shift

```
39 //PITCHSHIFT 11
40 global.pitchS = FMODGroupAddEffect (1, 11);
41
42 //Pitch
43 //Pitch value.  0.5 to 2.0.  Default = 1.0.
44 //0.5 = one octave down, 2.0 = one octave up. 1.0 does not change the pitch.
45 FMODEffectSetParamValue(global.pitchS,0,1);
46 //FFT size
47 //FFT window size.  256, 512, 1024, 2048, 4096.  Default = 1024.
48 //Increase this to reduce 'smearing'.
49 //This effect is a warbling sound similar to when an mp3 is encoded at very low bitrates.
50 FMODEffectSetParamValue(global.pitchS,1,4096);
51 //Overlap
52 //Window overlap.  1 to 32.  Default = 4.  Increase this to reduce 'tremolo' effect.
53 //Increasing it by a factor of 2 doubles the CPU usage.
54 FMODEffectSetParamValue(global.pitchS,2,32);
55 //Max channels
56 //Maximum channels supported.  0 to 16.
57 //0 = same as fmod's default output polyphony, 1 = mono, 2 = stereo etc.
58 //See remarks for more.  Default = 0.  It is suggested to leave at 0!
59 FMODEffectSetParamValue(global.pitchS,3,0);
60
61 // Bypass
62 FMODEffectSetBypass(global.pitchS,global.pitchS_bypass);
```

Comme pour la reverb, on applique le pitch shift au groupe contenant tous les sons de piano (ligne 40), puis on définit les paramètres initiaux de l'effet. Notez que les paramètres FFT Size et Overlap sont réglés à leur maximum, ce qui fait que le pitch shift est de très bonne qualité, au détriment des performances du programme.

Gestion des fluctuations du pitch shift

```
1  //Pitch
2  //Pitch value.  0.5 to 2.0.  Default = 1.0. 0.5 = one octave down, 2.0 = one octave up.
3  //1.0 does not change the pitch.
4  FMODEffectSetParamValue(global.pitchS,0,global.pitchS_value);
5
6  if (abs(global.pitchS_value - global.pitchS_rand) < 0.008)
7  {
8      global.pitchS_rand = (irandom(150)/100)+0.5;
9      global.pitchS_speed = (irandom(7)+1)/1000;
10 }
11 else
12 {
13     if (global.pitchS_value < global.pitchS_rand) global.pitchS_value += global.pitchS_speed;
14     if (global.pitchS_value > global.pitchS_rand) global.pitchS_value -= global.pitchS_speed;
15 }
```

Le système de fluctuation du pitch shift est très simple:
-une valeur de pitch B est définie aléatoirement,
-une vitesse V est définie aléatoirement pour atteindre cette valeur de pitch,
-à chaque boucle du programme, le pitch effectif A se rapproche de la valeur de pitch B, à la vitesse V,
-lorsque A=B, on réaffecte aléatoirement de nouvelles valeurs à B et V, et ainsi de suite.

♫ **Music_PitchS_1.mp3:** transitions entre le mode normal et le mode « brouillage ». Musique seule.

♫ **Music_PitchS_2.mp3:** transitions entre le mode normal et le mode « brouillage » avec les autres sons du jeu.

• ***Apport de la musique dans l'immersion du joueur***

Les éléments musicaux indépendants du gameplay, comme le système de modulations harmoniques ou les variations de cellules rythmiques permettent plusieurs choses:
-proposer une musique relativement variée,
-proposer au joueur, grâce à la musique, de s'évader de l'environnement oppressant dans lequel il se trouve. Plus on s'approche du nid, plus la musique est présente et entraînante. Le joueur, pour fuir les dangers et les ambiances angoissantes, va être attiré par le caractère doux et agréable de la musique en opposition avec le reste dans le bande son → sorte d'« ascenseur émotionnel ».

♫ **Demo_2.mp3:** ici, j'ai fait une simulation musicale du niveau type présenté dans la partie Mécaniques de jeu. J'ai mixé cet enregistrement à « Demo_1.mp3 » qui simule une bonne partie des sons du niveau. Ainsi, on a un bon exemple de la bande sonore du jeu au sein d'un niveau entier. La gestion en temps réel de tous les évènements sonores étant assez délicate, je n'ai pas pu lier la réverbération de la musique à la position verticale du joueur, comme le prévoit le jeu.
Il est fortement conseillé, en écoutant cet extrait sonore, de relire les différentes phases du niveau type dans la partie Mécaniques de jeu

□ Considérations techniques

• ***Compression des exports sonores***

Les fichiers sons du dossier sont tous des MP3 – 320 kbits/s vu que seuls les .mp3 et les .wav étaient autorisés.
Idéalement, les sons de Dikotomia devraient être en OGG VORBIS – 256 kbit/s, ce qui permettrait d'avoir à peu près la même qualité, pour un poids moindre.

Le MP3 et le VORBIS sont des algorithmes de compression destructeurs, ce qui signifie que même en 320 kbit/s, le signal est altéré. J'aurais pu utiliser des formats de compression non-destructeurs comme le FLAC, mais la compression avec ce format est trop faible pour pouvoir être vraiment intéressante.

On peut également compresser les sons contenus dans la banque sonore exportée avec le projet FMOD Designer en vue de l'intégration. Ces banques peuvent être de plusieurs types:
-stream from disk: le streaming permet de lire de gros fichiers sons sans les charger entièrement en RAM (idéal pour la musique). Demande un accès constant au support contenant le fichier son (disque dur, CD, DVD…). Gourmand en CPU.
-load into memory: les sons sont chargés en mémoire puis décompressés si nécessaire au moment de leur lecture (par défaut).
-decompress into memory: les sons sont décompressés avant d'être chargés en mémoire. Technique la moins gourmande en CPU (sauf au moment du chargement en mémoire), au détriment de la RAM.

Pour Dikotomia, tous les sons sont de type « load into memory » sauf les gros sons d'ambiances qui sont « streamés ». Ainsi, on a un bon compromis entre la charge CPU et RAM, et les accès au disque dur sont limités.

• ***Effets FMOD***

L'utilisation de technologies conçues pour la gestion du son en temps réel permet d'alléger sensiblement la charge en mémoire vive au détriment de la consommation CPU. En effet, bien que bien optimisés, certains effets sont quand même gourmands. C'est le cas de la SFX Reverb et du Pitch Shift de FMOD notamment.

Pour bien faire, il aurait fallu que je n'applique pas la SFX Reverb aux sons dont la réverbération ne varie pas en fonction des actions du joueur pour alléger la charge CPU. A la place, la reverb pourrait être appliquée en amont, dans le séquenceur ou l'éditeur audio. J'ai fait ainsi pour faciliter les réglages de l'ensemble des sons.

Cela dit, Dikotomia étant un jeu sonore, on peut se permettre un budget audio CPU/RAM bien plus important que pour un projet de jeu basé sur le visuel.

• ***Estimation de la charge CPU/RAM***

Estimations et mesures de la charge CPU/RAM avec compression OGG VORBIS 256 kbits/s pour tous les sons:
-Sons – HDD (streaming): 14,5 Mo
-Sons – RAM: 4,5 Mo
-FMOD Designer – RAM: 11 Mo
-Game Maker – RAM: 33,5 Mo
-FMOD Designer – CPU: 13%
-Game Maker – CPU: 10%

-Total – RAM: 49 Mo
-Total – CPU: 23 % (4 cœurs – 2,53 GHz)

On constate que la consommation processeur et mémoire vive, rien que pour l'audio, est très importante au regard des pratiques actuelles de l'industrie du jeu vidéo, où on est parfois limité à 15 ou 20 Mo de sons chargés en RAM, même pour une production sur consoles HD (PS3/Xbox360 → limitées certes à 512 Mo de RAM).
Sans utiliser Game Maker, qui est visiblement mal optimisé, et en se servant plus intelligemment des effets FMOD, on pourrait nettement alléger la mémoire vive et le processeur (RAM: 10 Mo / CPU: 6%).

□ Les Outils

Softwares:
-Cubase (+ VST/VSTi)
-Sound Forge
-FMOD Designer
-FMOD Event Player
-Game Maker + FMOD (librairie)
-Banques de sons