

# Projet STL 2014



---

Co-génération automatique intelligente de  
niveau et musique pour jeu vidéo évolutif

**Maude Bellamy – Simon Tchernia – Jérôme Rahault  
encadrés par Philippe Esling**

## Table des matières

|  |    |
|--|----|
| Introduction.....                            | 3  |
| I.Rédaction du sujet et premières bases..... | 4  |
| II.Architecture du projet.....               | 6  |
| a) Modèle du jeu.....                        | 7  |
| b) Moteur du jeu : Slick2D.....              | 8  |
| c) Rendu visuel.....                         | 8  |
| d) Rendu sonore.....                         | 12 |
| e) Génération de niveau et de musique.....   | 13 |
| f) Générateur aléatoire.....                 | 14 |
| III.Futures amélioration.....                | 15 |
| Conclusion.....                              | 17 |

## Introduction

Notre projet a suivi une élaboration assez différente des autres puisqu'il n'existait pas lorsque nous avons dû en choisir un. En effet, nous avons choisi de rédiger nous-même notre sujet, paradoxalement ce choix ne nous a pas simplifié la tâche bien au contraire, ce fut une étape de plus dans la réalisation du projet. Par chance nous avons trouvé un encadrant partageant nos idées qui a su nous guider vers des concepts originaux et intéressants.

Afin de vous expliquer les étapes de développement, nous allons présenter la chronologie du jeu, de la conception au développement, puis le détail de ses composants, enfin les améliorations que l'on souhaite apporter au jeu.

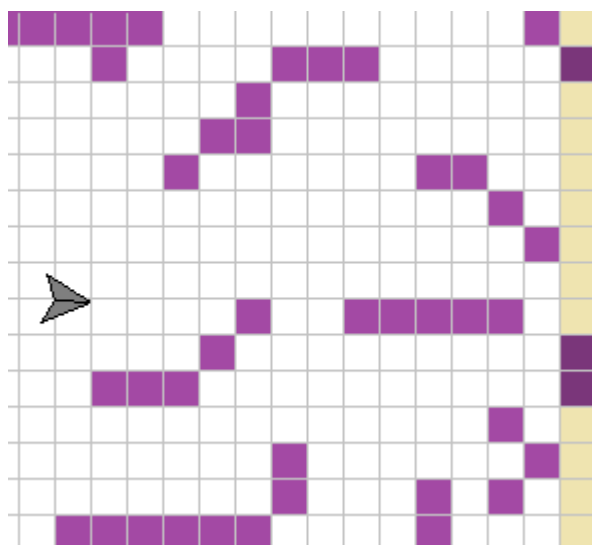
## I. Rédaction du sujet et premières bases

Notre équipe s'est formée à la suite d'un projet réalisé à six pour la matière d'ingénierie logiciel du premier semestre de master. C'est à cette occasion que nous avons fait connaissance et que nous avons appris à travailler ensemble. Après discussions, nous avons remarqué que nous avions pas mal de centres d'intérêts en commun et une envie commune de réaliser un projet sur notre temps libre. Mais comme nous devions choisir un projet pour une matière pstl, nous avons décidé de faire d'une pierre deux coups et de retravailler ensemble sur un projet personnel plutôt que sur un des projets proposés par nos enseignants.

Il nous fallait donc trouver un sujet et un encadrant. Notre première idée de projet, consistait en la création d'un outils permettant composer de la musique sur snes à l'aide d'un un mini langage qu'on aurait ensuite compilé en langage de processeur sonore de snes. Nous avons contacté un enseignant à même de nous aider dans le domaine musical : Philippe Esling de l'IRCAM. Lors de la première réunion, nous avons discuté du sujet et notre encadrant nous a orienté vers un autre projet qui nous intéressait tout autant, un jeu vidéo axé sur la musique :

- Les niveaux sont générés aléatoirement suivant un certain nombre de contraintes.
- Le joueur peut éventuellement influencer sur la musique (accélérer/ralentir le tempo, ajouter/supprimer des notes).
- La musique est générée en fonction du niveau tout en restant harmonieuse.
- La difficulté est évolutive selon une courbe avec des piques de difficultés tenant compte de l'état mental du joueur (équilibrant l'amusement et le stress).

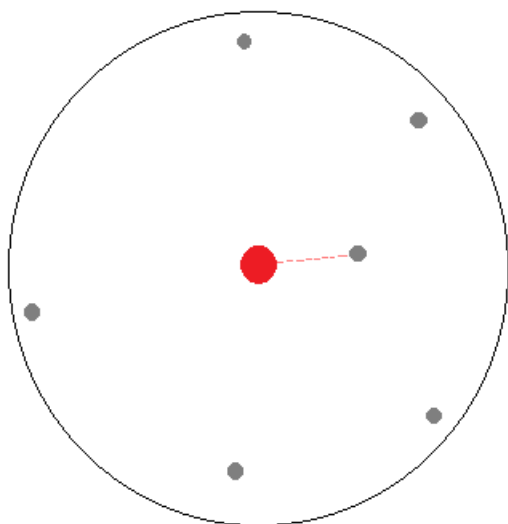
Il ne nous restait plus qu'à choisir le type de jeu et définir le cahier des charges. Au départ, la première idée fut le classique jeu de plate-forme mais la génération de musique ne s'applique pas très bien à ce mode de jeu. Un jeu à scrolling (par exemple un shooter) s'y prête beaucoup mieux puisque le niveau peut défiler au même rythme que la musique. Deux concepts se sont alors démarqués :



#### Jeu 1 :

*A la manière d'un shooter, le scrolling défile à l'horizontale et le but est d'éviter les blocs. Chaque bloc est une note joué dès son apparition (passage dans la zone jaune) et on peut accélérer et/ou ralentir le scrolling.*

*Possibilité d'amélioration : des ennemis à éviter, des tirs pour les détruire, d'autres divers bonus.*



#### Jeu 2 :

*Le protagoniste est au centre et les ennemis arrivent de toute part, le but est de les éliminer en leur tirant dessus pour survivre le plus longtemps possible. Les ennemis jouent une note jusqu'à ce qu'ils meurent, si le joueur ne les élimine pas assez vite, l'accumulation de sons devient désagréable. On peut ralentir le temps pour éviter d'être débordé et donc ralentir la vitesse d'apparition des ennemis.*

*Possibilité d'amélioration : on peut se déplacer, les ennemis donnent divers bonus (meilleures armes, possibilité de déclencher un ralenti etc...), différents types d'ennemis.*

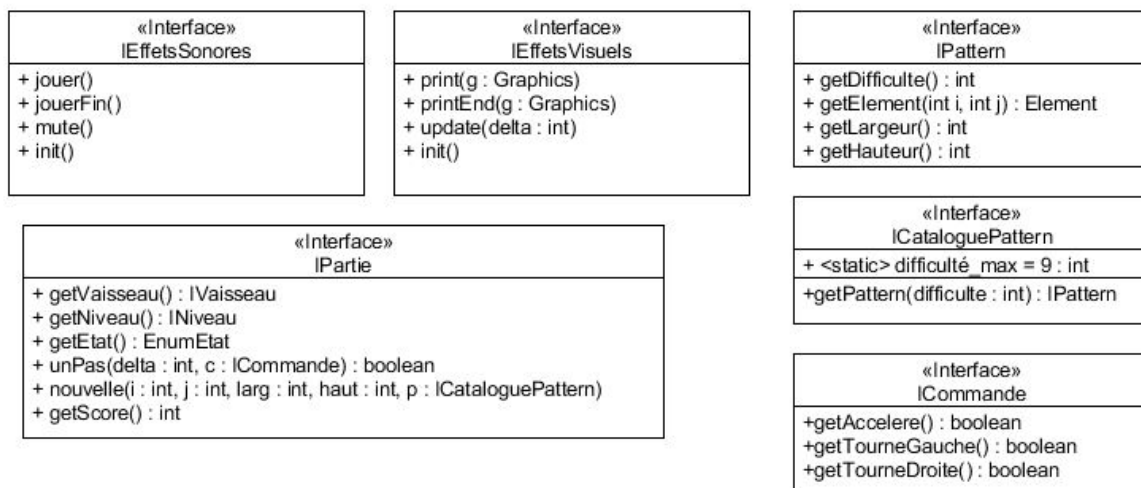
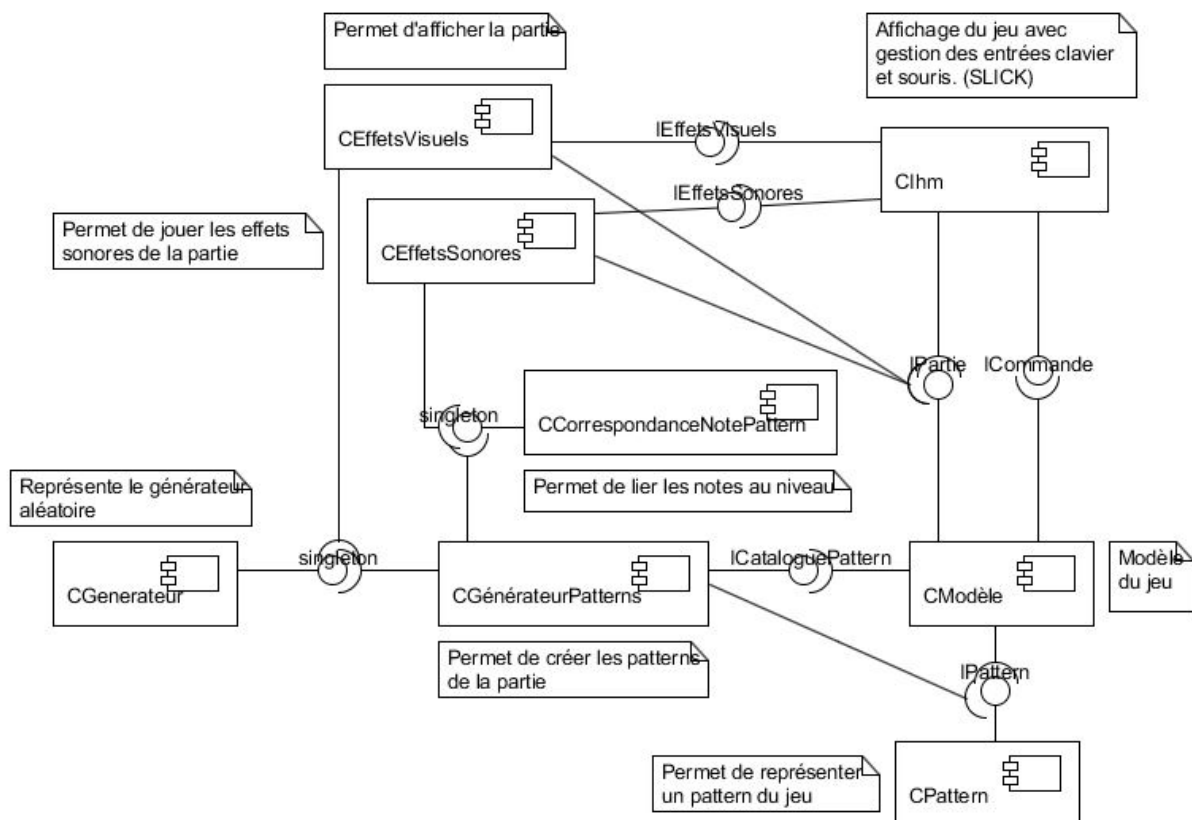
C'est le jeu 1 qui a été retenu. La génération de musique s'y prête mieux et la musique ne risque pas d'être cacophonique. De plus, dans le premier cas, la musique nous prévient de l'apparition des blocs, elle est notre allié, tandis que, dans le second cas on doit la faire disparaître pour avoir parce qu'elle devient désagréable, elle est notre ennemi.

Nous avons choisi de programmer en Java, étant donné que nous n'avons pas prévu de faire un jeu particulièrement gourmand en ressource et qu'il est portable. Java nous permet de le rendre accessible sur navigateur web et/ou sur smartphone.

## II. Architecture du projet

Notre encadrant nous a proposé deux architectures : le Component Based Engine et le Functional Reactive Programming (FRP). Nous avons opté pour l'architecture classique en composant qui nous semblait plus simple à mettre en place.

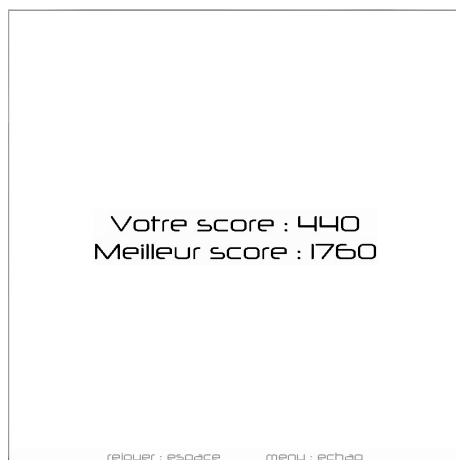
Voici le digramme composant du projet :



## *a) Modèle du jeu*

Nous définissons ici le modèle du jeu qui se compose d'une partie qui gère le niveau et sa mise à jour ainsi que le vaisseau.

La partie a pour rôle de mettre à jour à chaque pas le niveau et le vaisseau et de vérifier les événements du jeu. Lorsque le vaisseau touche un item, l'état de partie change suivant le type d'item (mode psychédélique et mode inverse avec commandes inversées) et revient en état normal au bout d'un certain temps. Elle calcule le score du joueur en fonction de la distance parcourue par le vaisseau. Ce score croît plus rapidement lorsque le vaisseau évolue dans un mode. Elle passe en état « collision mur » ou « collision ennemi » lorsque le vaisseau rentre dans ces derniers.



*Affichage en fin de partie :*

*On affiche le meilleur score atteint par le joueur et le score qu'il vient d'obtenir au cours de la partie. Le record est stocké dans un fichier afin de le conserver. Cela incite le joueur à persévérer pour battre son meilleur score.*

Le vaisseau est capable d'accélérer en ligne droite et en diagonale en fonction des commandes du joueur passées via la partie. Pour que cette accélération paraisse réaliste et agréable à jouer, nous avons fait en sorte qu'elle soit progressive en stockant un certain nombre de valeurs d'accélération dans un tableau. Ces valeurs suivent une courbe asymptotique pour simuler une accélération de voiture. Lorsque le joueur cesse d'accélérer, le vaisseau garde une certaine inertie. Pour les déplacements latéraux, nous utilisons la vitesse courante afin qu'ils soient proportionnels à la vitesse du vaisseau. C'est-à-dire que le vaisseau tourne rapidement quand il va vite. On représente le vaisseau par ses quatre extrémités. Elles sont resserrées lorsque le vaisseau tourne pour donner une sensation d'inclinaison et pour que les collisions correspondent exactement à la forme du vaisseau, ce qui ne serait pas le cas si nous utilisons simplement des sprites.

Le niveau gère le terrain, les ennemis et les items. Il met à jour à chaque pas les ennemis. Le terrain est une suite de pattern que l'on pioche dans un catalogue en fonction d'une difficulté. Les patterns sont des tableaux deux dimensions contenant dans chaque case un mur, un ennemi, un item (de différents types) ou rien.

Au cours de la partie, la difficulté augmente de deux façon, les ennemis sont de plus en plus nombreux et la vitesse de défilement horizontale (scrolling forcé) croît. Cela est géré par une fonction de difficulté. La difficulté est certes progressive, mais elle fonctionne selon l'idée qu'après avoir atteint un certain pallier, il faut la réduire durant un court laps de temps pour donner du répit au joueur avant une nouvelle augmentation. Cette approche de la difficulté nous a été proposée par

notre encadrant car elle garantit plus d'amusement qu'une évolution en dent de scie ou strictement croissante. La fonction de difficulté fournit donc au niveau la vitesse de scrolling forcé et la difficulté des patterns à tirer.

### ***b) Moteur du jeu : Slick2D***

Nous voulions utiliser un moteur de jeu pour simplifier la gestion et la visualisation du jeu et pouvoir se concentrer sur le modèle du jeu et la génération de musique et de niveau qui nous semblaient plus centraux.

Le moteur de jeu que nous avons trouvé pour Java est Slick2D, très fiable et très complet pour les jeux en deux dimensions. C'est une surcouche de la librairie LWJGL (api Java pour OpenGL) facile à utiliser et ne gérant que la deux dimensions.

Le principe consiste à définir des états tels que l'écran de chargement, le menu ou la partie et les transitions entre eux. Chaque état possède une fonction d'initialisation, une de mise à jour et une pour l'afficher.

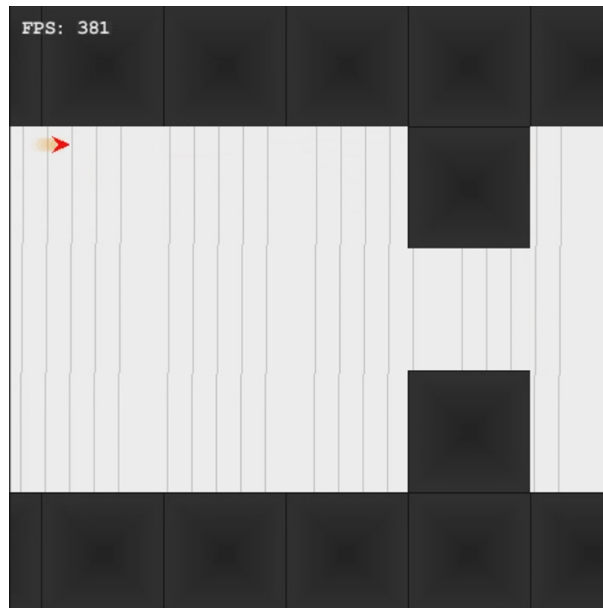
L'état le plus important est celui de la partie, il prend en argument une partie déjà créée. Il utilise deux composants, un permettant d'afficher le jeu et un deuxième permettant de jouer le son du jeu. Cet état est lancé par l'état de chargement qui initialise la partie dans un thread et qui passe la main lorsqu'il a fini. A chaque rafraîchissement de l'état de partie, celui-ci fait faire un pas à la partie avec en argument les commandes du joueur récupérées via les entrées clavier (flèche du haut : à gauche, flèche du bas : à droite, flèche de droite:accélérer). Lorsque la partie est perdue, il passe la main à l'état de fin de partie qui affiche l'animation de fin de partie via les deux mêmes composants. Pendant, ces deux états, le joueur peut choisir de recommencer la partie avec la touche espace ou revenir au menu avec la touche échap.

### ***c) Rendu visuel***

Il est évident de dire que la visualisation d'un jeu est aussi importante que son *gameplay*. C'est pourquoi nous avons accordé de l'importance à la partie visuelle.

Nous avons créé une visualisation contenant une liste d'observateurs qui possèdent un sujet à afficher, basé sur le pattern observer. Ce système est plutôt générique car on peut ajouter facilement plein de type d'observateurs (avec des sujets différents ou bien le même sujet et un affichage différent) et on pourra le réutiliser facilement pour un autre projet.

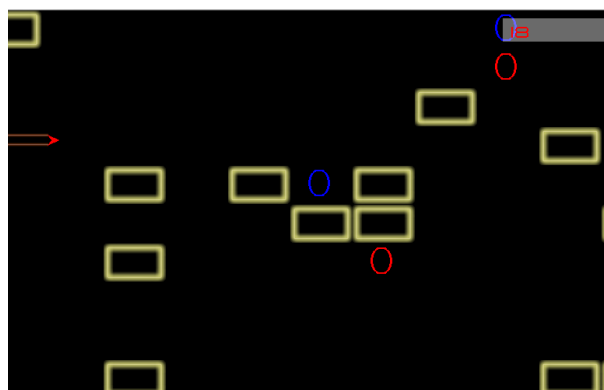




*Première version jouable de Soundle*

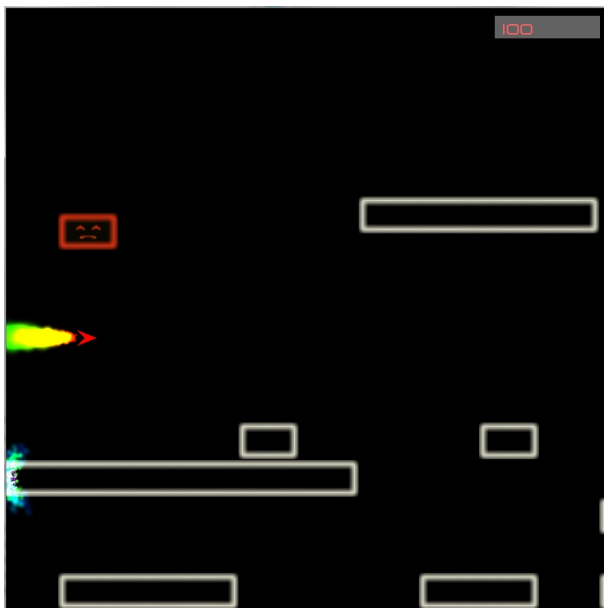
Le rendu visuel du jeu à ses débuts était assez rudimentaire voire austère. L'utilisation de noir et blanc était assez monotone et pouvait ennuyer le joueur. Nous avons alors pensé que les items que nous voulions incorporer pour modifier la musique, modifieraient également le visuel. Ainsi sont nés les modes *psychédélique* et *inverse*.

Afin d'améliorer le rendu du jeu, notre encadrant nous a alors suggéré l'utilisation d'effets simples qui rendraient des formes géométriques plus attrayantes.



*Application des premiers effets pour rendre le jeu attrayant*

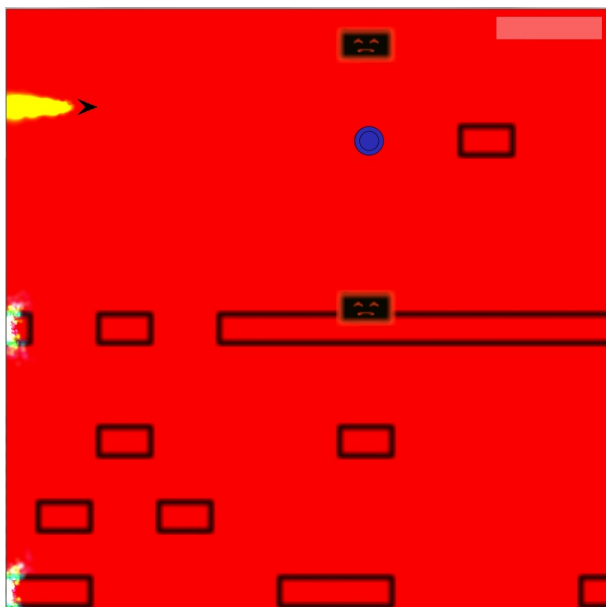
C'est pourquoi nous utilisons des sprites avec un effet de néon pour les blocs et les ennemis, ainsi qu'un effet de particules pour la traînée qui suit le vaisseau et celle lorsque l'on joue une note. Les items sont des sprites d'une pièce qui tourne pour donner une impression de profondeur au jeu. On différencie les items par leur couleur.



*Version actuelle de Soundle :*

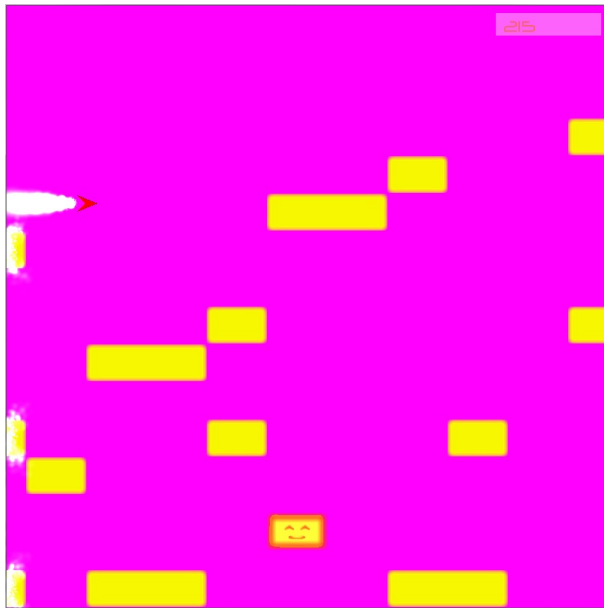
*Mode normal :*

*Lorsqu'au moins deux blocs sont consécutifs, on affiche des sprites différentes pour donner l'illusion d'un unique grand bloc.*



*Mode inverse :*

*On échange les couleurs du vaisseau et du fond ainsi que les commandes pour désorienter le joueur. Pour monter le vaisseau il doit appuyer sur la flèche du bas et pour le descendre, sur la flèche du haut.*

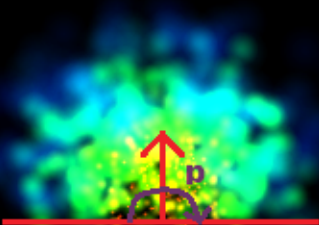


*Mode psychédélique :*

*Les couleurs du fond et des blocs changent aléatoirement et les ennemis sourient.*

*Cependant ils sont toujours aussi menaçants.*

Un des outils que nous avons beaucoup exploité est le générateur de particule (code utilisé trouvé sur le tutoriel du site : <http://frums.nl>), qui nous a permis d'avoir des effets convaincants pour la traînée du vaisseau, la visualisation des notes jouées ainsi que l'explosion du vaisseau lors de son crash. Les trois effets sont assez différents et pourtant nous n'avons pas eu à adapter énormément le code, faire varier les nombreux paramètres a suffi. Cela permet de concevoir des effets complexes comme de la fumée, des flammes, des fontaines, des explosions voir des effets d'arcanes pour des sortilèges magiques. L'avantage est que nous n'avons pas besoin de contrôler une à une chaque particule, nous en contrôlons un groupe avec certains facteurs aléatoires prédéfinis. Le principe consiste à générer un assez grand nombre de points (ou autre forme simple) mobiles qui changent de manière aléatoire mais extrêmement contrôlée en définissant entre autres une zone de naissance (spawn), des indications de directions, des variations de couleurs, une durée de vie etc...

|  |  |
|--|--|
| <p>➡ <b>facteur de vent : 0</b><br/>(-&gt; : 5 / &lt;- : - 5)</p> <p>⬇ <b>facteur de gravité : 0</b><br/>(↓ : 5 / ↑ : - 5)</p> | <p><b>interval de naissance :</b><br/><b>max=10 min=10</b></p> <p><b>facteur de grossissement : 30</b></p> <p><b>nombre de particules :</b><br/><b>max=1000 min = 1000</b></p> <p><b>durée de vie : false</b></p> <p><b>distance initiale :</b><br/><b>min=0 max=0</b></p> <p><b>couleurs :</b><br/><b>pas[0] = RGB(1,0,0), offset=0</b><br/><b>pas[1] = RGB(0,1,0), offset=0.5</b><br/><b>pas[2] = RGB(0,0,1), offset=1</b></p> |
|   |  |
| <p><b>Légende :</b><br/>(...) = autres<br/>valeurs possibles</p>   | <p><b>inclinaison : 0</b><br/>( -&gt; : 90/&lt;-  : 270)</p> <p><b>p = angle de<br/>propagation : 180</b><br/>(0 : 360 / L : 90)</p>   |

#### *Exemple d'une génération de particule simple*

Parlons des paramètres les plus utilisés : les facteurs de vent et de gravité (absents de cet exemple (=0)) permettent d'orienter le déplacement pseudo aléatoire des particules (horizontalement avec le vent et verticalement avec la gravité). Par exemple, pour la traînée du vaisseau, nous avons mis une valeur de vent négative qui augmente négativement lorsque le vaisseau accélère. L'angle de propagation est de 45° pour la traînée alors qu'il est de 360° pour l'explosion et le facteur de grossissement est beaucoup plus important pour l'explosion que la traînée. La durée de vie n'a pas été utilisée, elle concerne le générateur et pas les particules individuellement et permet simplement de cesser la génération au bout d'un certain nombre de milliseconde, nous l'utiliserons sûrement dans de prochains effets. Les points de couleurs apportent cet effet dégradé très élaboré grâce à un simple offset indiquant quand la couleur doit prendre de l'importance sur les autres.

#### *d) Rendu sonore*

Les observateurs sonores fonctionnent selon le même principe que les observateurs visuels. Ici nous utilisons deux types de sons. Les sons midi et des *samples* au format wav.

L'idée générale est qu'il y a une note associée à une ligne de jeu. Ensuite, on définit un observateur par ligne, il y en a ici 16. Dès qu'un bloc touche le bord gauche de l'écran la note qui lui est associée est jouée en boucle et dès que le bloc disparaît de l'écran, la note s'arrête. Pour nos trois modes visuels et sonores nous utilisons différents observateurs et sons. Le mode normal utilise le midi.

Le midi est un format sonore de synthèse de son qui exploite la carte son de l'ordinateur. C'est notre encadrant qui nous a proposé d'utiliser ce format, simple à exploiter et dont l'étendue de

sons est assez large pour pouvoir choisir ce qu'il nous plaît. Nous utilisons pour le mode normal des sons de vibraphone et de percussions à peau.

Les modes *8-Bits* et *Psyché* utilisent des samples. Nous avons décidé d'utiliser des samples de notes synthétisées sous formes d'ondes carrées pour représenter le mode 8-Bits. Rappelant ainsi les vieilles puces sonores des consoles des années 80 et le projet original que nous avons imaginé. Pour le mode Psyché, nous utilisons des samples de voix humaine et de percussion électronique.

Nous avons utilisé des bruitages pour créer une ambiance. Le vaisseau produit en permanence un bruit de moteur pour signifier qu'il avance et dès que l'on attrape un item un son est joué.

Enfin, lorsque la partie est perdue, le vaisseau, on entend le bruit de l'explosion du vaisseau.

### *e) Génération de niveau et de musique*

Le cœur algorithmique et musical du projet se trouve ici. En voici les étapes :

➤ **choix de la tonalité de départ de la partie :**

L'algorithme de génération de blocs et donc de notes est un improvisateur dans une tonalité. L'improvisateur choisit parmi les notes de sa gamme celle qu'il va jouer après la note courante. Afin de choisir une note, on utilise des règles d'harmonie garantissant une certaine musicalité.

➤ **calcul des tons voisins :**

Pour plus de variété, nous voulons pouvoir moduler, c'est-à-dire changer de tonalité. On ne peut pas changer de tonalité n'importe comment. En effet, chaque tonalité à trois tonalités dites voisines, avec une altération (dièse, bémol ou bécarré) d'écart en plus ou en moins, dans lesquelles le compositeur peut piocher pour colorer son morceau. L'utilisation de cette technique nous procure non seulement une modulation mélodique mais également une modification du niveau puisque d'autres notes, qui ont pu ne pas sortir, vont être jouées. Nous nous restreignons à six tonalités, les trois voisines de celle de départ plus les relatives mineures des deux extrêmes.

➤ **choix du rythme d'accompagnement à la percussion :**

Pour enrichir la mélodie, nous décidons d'utiliser des rythmes de percussion. Pour cela, nous avons défini un certain nombre de rythmiques à quatre percussions différentes comme étant des masques de patterns. Lors de la création des patterns, nous choisirons au hasard une rythmique pour ensuite masquer le pattern avec le rythme.

➤ **construction du catalogue de patterns, pour chaque tonalité choisie on crée un certain nombre de patterns de la façon suivante :**

➤ on tire une liste de rythmes :

Pour le rythme de la musique nous avons défini des rythmes courants associés à une certaine probabilité. Cette probabilité est choisie aléatoirement entre deux bornes que nous avons fixées. Cela permet de rendre chaque partie différente. En effet, une partie peut avoir des probabilités plus élevées pour des rythmes rapide qu'une autre, elle nous paraîtra plus vive et la deuxième plus posée. Nous avons ensuite défini nos rythmes comme étant une suite d'unités rythmiques associées à des chiffres, où la plus petite unité est la double croche qui vaut 1 bloc de pattern, la plus grande étant la blanche qui vaut 8, puis ce sont des multiples de 2.

➤ on applique des notes à ces rythmes :

Après avoir sélectionné un rythme, nous choisissons une note ou un silence par unité de rythme. Pour choisir une note, on applique un intervalle à la note précédente pour trouver la note courante. Ces intervalles sont associés à des probabilités d'apparition qui sont tirées sur le même principe que les rythmes. Nous avons choisi les bornes de façon suivant leur fréquence d'utilisation en musique selon nous. Par exemple, une tierce est un intervalle courant en musique *classique* ou populaire, tandis qu'une quarte augmentée plus particulière à l'écoute et donc moins présente. La sélection d'un intervalle se fait également dans un écart restreint, c'est-à-dire qu'on ne le choisit pas si les deux blocs correspondants sont trop éloignés spatialement car une grande différence de hauteur n'est pas belle si elle apparaît trop souvent dans une mélodie.

➤ on applique le masque de percussion :

Maintenant que l'on a la mélodie, on peut appliquer la rythmique choisie à ce motif. Les rythmiques sont toutes compatibles avec les mélodies créées.

➤ on duplique le motif et on ajoute aléatoirement des ennemis et des items afin d'obtenir des patterns de plusieurs difficultés

L'utilisation de patterns nous permet de rendre le niveau et donc la musique modulaire. En effet, lors de l'initialisation de la partie, nous créons un certain nombre de patterns dans les tonalités possibles. Ainsi, lorsque l'on tire un pattern dans le catalogue, il peut sortir à nouveau et donc être entendu à nouveau. Cela a deux effets bénéfiques : la musique a une couleur générale globale puisqu'elle est harmonieuse et les schémas reviennent comme un refrain, ce qui est courant en musique. De plus, le joueur a la sensation qu'il écoute un morceau original et joue à une partie différente à chaque fois.

## **f) Générateur aléatoire**

Une partie du projet est basée sur de l'aléatoire encadré. La mise en place d'une instance unique de générateur nous permet de gérer la partie aléatoire mais son unicité garantit une éventuelle partie identique à une précédemment jouée. Ainsi cela permet de penser que si la musique nous a plu, en sauvegardant la graine du générateur on peut recréer cette musique.

### III. Futures améliorations

A ce jour, tous les objectifs que nous nous étions fixés ont été atteints mais un grand nombre d'idées a fleuri dans nos esprits au cours du développement du projet et nous n'avons pas eu le temps de toutes les exploiter. Nous sommes néanmoins satisfaits du résultat atteint bien qu'il ne soit pas parfait. C'est pourquoi, nous avons déjà décidé de poursuivre Soundle pendant les vacances d'été où nous aurons la possibilité d'y consacrer beaucoup de temps. Nos idées étant suffisamment nombreuses, nous avons décidé de consacrer une partie à leur présentation.

- Intégration d'un mode où l'on joue sur une musique de sa liste de lecture. Celle-ci servira alors de base pour générer le niveau. Nous pourrions utiliser les formats midi pour le créer et faire jouer un fichier mp3, Wav ou autre pour que le joueur entende sa musique.
- Amélioration de l'influence du joueur sur la musique. Nous allons continuer dans cette voie en ajoutant de nouveaux items.
- Intégration d'une base de données musicale par style pour pouvoir varier le style du niveau et donc de la musique. Cela nous permettra d'utiliser un logiciel que notre encadrant a développé. Il s'agit d'un « plagieur » qui analyse un fichier midi et qui recompose un morceau selon ce qui apparaît le plus souvent. L'idée serait donc d'associer plusieurs morceaux du même genre et de se balader dans le morceau composé par le plagieur. Ainsi, le joueur aurait la sensation d'une grande improvisation dans un genre qui lui convient et qui ressemble à des morceaux connus. Ensuite on pourrait intégrer dans le menu options le choix du style musical.
- Modification plus fréquente des instruments pour varier la musique.
- Pouvoir revenir en arrière. C'est-à-dire faire défiler la musique dans le sens inverse pendant un laps de temps.
- Ajout de tirs. Sur le gameplay en lui-même, il était question d'ajouter des tirs pour faire évoluer la musique en détruisant des blocs. Cependant, une capacité de tir infinie permettrait au joueur de détruire tous les blocs devant lui en ligne droite et de ne pas jouer. On ajoutera donc une jauge de tir et/ou des blocs incassables.
- Ajout de nouveaux bruitages. Par exemple, les blocs ennemis (ceux mobiles) pourraient eux aussi produire une note une fois évité. On a imaginé pouvoir les détruire grâce à un bonus, la note ne serait alors pas jouée ou une autre serait jouée au moment de la destruction, voir même jouer une note désagréable lorsqu'on ne les a pas détruits.
- Peaufinage du visuel. Par exemple, rajouter des éléments dans le fond comme des étoiles ou autres ainsi que réaliser un sprite soigné pour l'apparence du vaisseau.
- Rendre les niveaux toriques, c'est-à-dire que le vaisseau se retrouverait en bas lorsqu'il sort en haut et inversement pour apporter une possibilité d'esquive plus complexe.
- Ajout de zone plus difficiles correspondant à des boss.

- Enfin, rendre le projet portable. En effet, cela tenait à cœur à notre encadrant. Ce type de jeu est adapté pour ordinateur mais également pour smartphone ou tablette. C'est pourquoi nous allons en réaliser un portage Objective-C pour Iphone et Ipad, intégrer l'API Android au projet actuel pour les machines Android, et éventuellement en C# pour les Windows Phone.

On se rend compte que le jeu est déjà bien avancé mais qu'on pourrait toujours en ajouter davantage pour en faire une version *distribuable*. Cependant, La version présente ici est jouable et agréable et correspond non seulement au sujet mais également à au résultat que nous voulions atteindre.



## Conclusion

Même sans penser à l'aspect technique, ce projet nous a beaucoup apporté. Nous avons réellement dû nous impliquer dans chaque étape de son élaboration. Nous avons eu à rédiger le sujet et donc à concevoir le cahier des charges du projet en nous mettant d'accord avec notre encadrant, qui ne nous a jamais limité, mais surtout entre nous. Par chance nous étions trois, et de ce fait lors de désaccord le vote « à main levée » a permis de trancher.

En plus de l'expérience du travail d'équipe, nous avons aussi pris conscience de la difficulté de réaliser tous les objectifs dans les délais impartis et ainsi de revoir l'organisation et les priorités en passant plus de temps sur les tâches les plus primordiales plutôt que les tâches insignifiantes ou demandant trop de travail par rapport à leur utilité. On peut le voir dans l'évolution visuelle de notre jeu, les brouillons nous ont permis d'avoir le résultat actuel mais nous ont aussi demandé beaucoup de travail qui n'apparaît plus dans le résultat final. D'ailleurs beaucoup de lignes de code n'ont finalement pas été utilisées.

L'intégration de la musique n'a pas été une tâche facile. Bien que deux d'entre nous connaissent la musique, nous n'avions jamais programmé sur un sujet musical. Par exemple la possibilité de générer un niveau grâce à une musique déjà existante est une bonne idée mais est trop longue à mettre en œuvre et n'est pas le cœur du projet. Nous avons passé du temps à décortiquer un fichier midi et à le lire. Mais la création d'un niveau à partir d'un fichier midi est trop complexe pour un simple ajout. Mais ce travail n'est pas perdu étant donné que nous comptons l'utiliser pour améliorer le projet.

Nous avons également appris à développer un jeu en s'appuyant sur une bibliothèque existante (Slick2D) et d'autres morceaux de code, et donc à réutiliser et à adapter tout un catalogue de méthodes pour nous simplifier la tâche. Cela s'approche davantage d'une conception logicielle moderne où l'on cherche à réinventer la roue le moins possible.

Nous sommes vraiment satisfaits du rendu de notre jeu à l'heure actuelle même si nous aurions aimé avoir eu le temps de l'adapter pour smartphone ou de faire une applet pour navigateur internet pour le rendre jouable par n'importe qui. Mais comme nous avons décidé de le mener à bien par la suite, nous espérons que tous nos objectifs initiaux seront atteints et que le jeu deviendra proche d'un résultat commercialisable.