Méta-morphosis – cahier des charges

Owl's team

Pierre Bourdon <bourdo_p@epita.fr>
Florent Ditte <ditte_f@epita.fr>
Tiphaine Petit <petit_t@epita.fr>
David Stavaux <stavau_d@epita.fr>

Table des matières

1 Présentation du groupe

1.1 Pierre Bourdon

Portant le rôle de chef de projet une année de plus, Pierre est avant tout l'initiateur du groupe et de l'idée de sujet que nous comptons développer. Il est décidé à ne pas répéter l'erreur de travailler sur un projet qui ne le motive pas comme il l'a fait l'an dernier, et à remettre en place l'équilibre de l'univers en réalisant un projet correct pour son année d'InfoSPÉ.

Intéressé par tout ce qui l'ammène à apprendre des choses, il maitrise la plupart des technologies que nous utiliserons pour la réalisation de ce projet : OCAML, C, MERCURIAL mais a également des connaissances concernant l'architecture de la PlayStation, point crucial pour mener à bien Méta-morphosis. Une de ses tâches sera donc également de former son groupe à de nombreuses choses nouvelles pour des étudiants d'InfoSPÉ, et d'ainsi transmettre les connaissances qu'il a acquis.

Il a également une vision d'ensemble des choses à réaliser pour le projet lui permettant de planifier et de répartir efficacement le travail à effectuer pendant les prochains mois où la Owl's team devra programmer. Ainsi, il aura un grand rôle dans la mise en place de l'architecture globale du projet, point crucial dans la programmation d'un logiciel sur plusieurs mois.

1.2 Florent Ditte

Passant enfin en InfoSPÉ après un redoublement de sa première année, Florent est, lui aussi, motivé et impliqué au maximum pour ce projet. Peu intéressé par l'idée de base du groupe (qui était la création d'un système de fichiers distribué), il fut cependant séduit par la deuxième idée qui vint à l'esprit des membres du groupe, et décida de se lancer dans l'aventure.

Voulant travailler dans le domaine du jeu vidéo, c'est de plus un sujet qui touche à son domaine favori et qui lui apportera un peu d'expérience, certes éloignée de la programmation de jeux vidéos en soi, mais tout de même intéressante pour un ingénieur.

Ce projet sera donc pour lui une occasion d'améliorer sa maitrise de l'algorithmique en travaillant sur des structures de données *on-disk* tels que des systèmes de fichiers, mais également de la programmation en général avec l'utilisation de langages comme le C et l'OCAML, et d'outils comme MERCURIAL ou BITBUCKET en passant par LaTeX.

1.3 Tiphaine Petit

Troisième membre a nous rejoindre pour la réalisation de ce projet dont le sujet l'a immédiatement accrochée, Tiphaine est motivée et prête à travailler à fond pour être utile au groupe et au projet, Méta-morphosis. Malgré moins de connaissances en informatique que le reste du groupe, elle compense par sa volonté d'apprendre les notions qui lui manquent et les technologies qu'elle ne maitrise pas encore.

Ce projet lui apportera de nouvelles expériences dans le domaine du travail en groupe, ainsi qu'une meilleure organisation, nécessaire pour réaliser un tel projet : gérer son temps pour la réalisation de certaines tâches est une des clés de la réussite d'un projet, et c'est un point qui lui fait parfois défaut.

D'un point de vue informatique, Méta-morphosis lui permettra également d'améliorer ses connaissances tout en suivant le programme de l'année d'InfoSPÉ. Elle espère ainsi pouvoir apprendre énormément de ce projet, si possible dans le maximum de domaines de travail concernés par le sujet, tels que les algorithmes de décompression ou l'architecture des consoles et des ordinateurs.

1.4 David Stavaux

Reparti pour une année qui n'était pas à l'origine dans ses prévisions, David est toujours prêt à apprendre de nouvelles choses. Ayant déjà participé à deux projets : un d'InfoSUP, le jeu le plus belge de la planète, *RollingFrite*, et un d'InfoSPÉ, *Ithree*, remplaçant la souris par une caméra et des mouvements, il a maintenant une vraie expérience dans la programmation de logiciels, que ce soit en C ou en OCAML.

Emballé dès le départ par le sujet du projet, mais sceptique sur la capacité d'un groupe d'InfoSPÉ à le réaliser, il s'est rendu compte qu'un découpage efficace permettrait d'avancer correctement et est maintenant plus motivé que jamais pour travailler aussi bien sur le programme en lui-même que sur d'autres tâches comme la réalisation du site web du projet.

Pour lui, ce projet est une expérience de plus, lui permettant de gagner en connaissances et en assurance. C'est également une occasion de se perfectionner ou d'apprendre à utiliser les différentes technologies dont l'utilisation est prévue dans Méta-morphosis, mais aussi un moyen d'appréhender l'architecture des consoles et des ordinateurs et d'aller dans les détails sur ce sujet.

2 Objectif du projet

2.1 Présentation du sujet

Notre projet, nommé *Méta-morphosis*, consiste globalement à écrire un programme capable de traduire un exécutable compilé pour la PlayStation de Sony en un autre exécutable, lui compilé pour fonctionner sur un PC muni d'un des OS supportés[1]. Malgré cette définition assez simple, c'est une idée qui contient de nombreuses tâches et qui comporte de nombreux problèmes algorithmiques à résoudre, dont nous parlerons dans la prochaine partie.

2.1.1 Vue d'ensemble de la PlayStation

La PlayStation est une console de jeu créée par Sony à la fin des années 1990, qui est considérée comme une console de 5$^{\text{ème}}$ génération[2]. Son architecture se base sur un processeur MIPS R3000A, qui a un jeu d'instructions très réduit, et sur de nombreuses puces spécialisées dans certains domaines utiles au jeu vidéo : le Graphics Transformation Engine (GTE) qui permet d'exécuter des opérations vectorielles et matricielles, le Data Decompression Engine (DDE) permettant de décompresser des données utiles au jeu (notamment les images JPEG, et par conséquant les vidéos au format MJPEG), la Graphics Processing Unit (GPU) traitant les vertices pour les afficher à l'écran, et la Sound Processing Unit (SPU) qui gère la décompression, la modulation et le mixage des sons et des musiques.

Cette console est également dotée de 2 Mio de RAM à vocation générale, d'un lecteur de CD-ROM, de deux ports permettant de brancher des contrôleurs de jeu et de deux ports pour les cartes mémoires, mesurant chacune 128 Kio[3].

L'accès à la plupart des ressources matérielles et des périphériques est à la charge du BIOS, qui est la propriété de Sony et ne peut être redistribué. Ainsi, un des rôles de notre projet sera d'émuler les fonctions du BIOS de la PlayStation, qui permettent l'accès au système de fichier du lecteur CD[4], aux manettes ou aux cartes mémoires.

2.1.2 Les différentes tâches du projet

La première et la plus importante des choses à réaliser pour ce projet est la partie traduction de binaire. En plus de convertir les binaires COFF pour

[1] Nous visons sur ce point le support d'au moins FreeBSD, Linux et Windows.
[2] Tout comme la Nintendo 64, la Saturn de Sega et la Jaguar d'Atari.
[3] Et dont la gestion du système de fichier est à la charge du jeu.
[4] Qui est, comme pour la plupart des CD-ROM, du ISO-9660 multipiste.

MIPS R3000A en binaire ELF ou PE32 pour architecture X86[5], ce traducteur doit également convertir les appels au BIOS qui se font via des syscalls en appels de fonctions C, et convertir les accès en lecture ou écriture aux adresses mémoires mappées vers un registre CPU, GPU, SPU ou d'une autre puce. Ainsi, le binaire sortant dépendra de bibliothèques dynamiques servant au support du BIOS, du GPU, etc. qui sont autant de pièces du puzzle à reprogrammer.

Notre BIOS, exportant l'intégralité des syscalls supportés par le BIOS de la PlayStation, permettra l'accès aux lecteurs CD du système et à la lecture de leur système de fichiers[6], mais fournira également un allocateur de mémoire (`malloc` / `free` / `realloc`), des fonctions de gestion des chaines de caractères et d'accès aux cartes mémoires émulées dans des fichiers. Il utilisera la bibliothèque SDL pour accèder aux controlleurs de jeu via les API du système afin de permettre de jouer avec une manette.

Le GPU sera, lui, émulé par une bibliothèque utilisant l'API la plus commune pour la programmation graphique sur PC, OpenGL. Fonctionnant via le même système de primitives et de gestion des textures, il devrait être assez facile de faire correspondre les différents appels au GPU avec ceux fournis par OpenGL.

Le SPU a un rôle dans la PlayStation très proche de celui qu'a la bibliothèque OpenAL de Creative. Ayant une API très ressemblante à celle d'OpenGL[7], elle permet le mixage des sons, le streaming des musiques (via une file de lecture des buffers de son), mais également le positionnement des sons qui est géré par le SPU de la PlayStation.

Enfin, le GTE et le DDE sont deux puces qui réalisent des calculs matriciels et vectoriels pour la première, et de décompression pour la deuxième. Les algorithmes de décompression gérés par le DDE sont documentés et reprogrammables facilement en OCaml ou en C, tout comme les algorithmes de calcul matriciel et vectoriel du GTE.

2.2 État de l'art

De nos jours, le moyen empirique pour jouer à des jeux de PlayStation sur un PC est d'utiliser un émulateur. Malgré des améliorations constantes, un émulateur reste un moyen assez lent et demandant beaucoup de performances pour faire tourner des applications faites pour fonctionner avec un

[5] En passant surement par un backend commun comme le langage C.

[6] Les systèmes d'exploitations ne gèrent pas communèment le multipiste sur les CD-ROM.

[7] Les deux bibliothèques sont maintenant maintenues par le Khronos Group.

processeur de 33 MHz. En effet, les configurations requises pour les émulateurs sont souvent aux alentours des 1.4 GHz et 256 Mio de RAM, plus de 40 fois [8] ce qui était requis par la PlayStation en son temps.

Malgré l'apparition récentes de Just In Time compilers (JIT) dans les émulateurs, cette compilation n'est effectuée qu'à l'exécution d'un jeu, et est refaite à chaque lancement de ce même jeu, alors qu'elle pourrait être faite une seule fois, certes plus lentement, mais donnant en sortie un exécutable plus rapide et natif.

Avec le développement des ordinateurs mobiles tels que les netbooks et les tablet PC qui fonctionnent avec des processeurs peu performants (Intel Atom, AMD Geode), ce projet prend tout son intérêt : ces processeurs à faible cadence ne suffisent souvent pas à faire fonctionner un émulateur, alors qu'un exécutable natif, certes demandant plus de ressources que 33 MHz, fonctionnerait peut-être sans problèmes.

2.2.1 Premiers émulateurs : PSemu et Sope

À chaque sortie d'une nouvelle console, les communautés de *hackers* ont toujours un objectif : utiliser toute la puissance de la machine à des fins non prévues par le constructeur. Encore de nos jours, certaines personnes travaillent d'arrache pied pour installer, par exemple, des logiciels de media center sur leurs consoles, alors que ça ne serait pas possible en restant avec les choses fournies par les constructeurs de console.

Pour travailler efficacement, ces personnes travaillent la plupart du temps sur leur ordinateur, et testent également sur un ordinateur via un émulateur. Ainsi, un an après la sortie de la PlayStation [9], une équipe de passionés sortaient le premier émulateur PlayStation, nommé *PSEmu Pro*. Cet émulateur, disponible dans un premier temps pour WINDOWS 95 et MS-DOS, puis par la suite pour WINDOWS 98 était lent, primitif et ne permettait pas de faire tourner la plupart des jeux commerciaux à cause de la faible compatibilité avec le système. Il fut maintenu jusque 2001, après quoi les programmeurs de cet émulateur rejoignirent différentes autres équipes programmant des émulateurs plus complexes et plus évolués dans le but de faire fonctionner les jeux commerciaux sur PC.

Il faut noter que *PSEmu Pro* réalisait une performance assez incroyable en n'étant que deux à trois fois plus lent que la PlayStation quand il fonctionnait sous MS-DOS. Cependant, cette vitesse est expliquable par le fait que MS-DOS n'était pas un système multi-tâche, et ainsi toute la puissance de

[8] Facteur néanmoins assez peu représentatif vu la différence de vitesse entre architectures RISC et CISC.

[9] En 1996, donc

l'ordinateur était dédiée au jeu et à l'émulateur.

PSEmu Pro fonctionnait via un système de plugins qui permettait d'utiliser différentes bibliothèques pour afficher à l'écran (plugin GPU), pour rendre le son (plugin SPU), pour communiquer avec les manettes (plugin PAD) ou pour communiquer avec le lecteur CD-ROM. Un travail conséquent avait été réalisé sur l'API commune à tous ces plugins, et cette API est de nos jours encore utilisée dans tous les émulateurs modernes. Ainsi, des plugins développés pour *PSEmu Pro* fonctionnent encore avec *epsxe* ou *pcsx*.

En parallèle[10] avec cette histoire, une autre équipe de développement codait un émulateur nommé *Sope*. Cet émulateur, sorti en 1998 et dont le développement s'arrêta en 2000, fonctionnait sur les systèmes Unixoides (dont, notamment, GNU/LINUX) et sur les architectures X86 et ALPHA. Il est tout de même intéressant d'en parler car, même s'il était beaucoup moins complet et fonctionnait moins bien en pratique que *PSEmu Pro*, il était le premier émulateur PlayStation à avoir des fonctionnalités de recompilation dynamique, ou JIT [11], lui permettant d'être presque aussi performant que la PlayStation. Ces deux émulateurs inspirèrent les créateurs de nouvelles équipes qui fonderont respectivement les trois compilateurs utilisés de nos jours : *epsxe*, *pSX* et *pcsx*. Notons également que même si *PSEmu Pro* était un logiciel propriétaire, *Sope* ne l'était pas, et une partie de son code source fut réutilisé ensuite.

2.2.2 Évolutions : epsxe, pSX, pcsx

Se basant sur tout ce qui a été réalisé dans les émulateurs de génération précédente, ces trois émulateurs, toujours en utilisation de nos jours, sont d'une qualité sans précédent. Permettant notamment de jouer à de nombreux jeux commerciaux, ils sont toujours maintenus et fonctionnent sur les systèmes d'exploitations modernes tels que WINDOWS SEVEN ou GNU/LINUX, ou encore NETBSD [12].

Implémentant la globalité du matériel de la console et recodant parfois même le BIOS de la PlayStation au lieu de demander un dump binaire venant d'une opération difficile à réaliser nécessitant une console physique, ces émulateurs sont également suffisamment rapides pour être utilisables sur des machines modernes, et sont totalement modulables, en utilisant le système de plugins mis en place dans *PSEmu Pro*. Ils permettent de plus le support des manettes de PlayStation et des vibrations, point crucial pour obtenir une expérience de jeu satisfaisante.

[10] Mais en se croisant quand même...
[11] Just In Time compiler.
[12] Pour jouer sur votre cafetière !

FIG. 1 – Un jeu commercial, *Final Fantasy 9*, fonctionnant sur l'émulateur *pSX*.

Ces trois émulateurs, en plus d'avoir réutilisé le système de plugins de *PSEmu Pro*, se sont également inspiré de *Sope* au niveau de la recompilation dynamique. Perfectionnée, cette méthode est en application dans tous ces émulateurs, permettant d'obtenir une vitesse correcte même dans les jeux repoussant les limites de la PlayStation.

Il faut également noter que sur ces émulateurs toujours maintenus, deux sont sous licence propriétaire (*epsxe* et *pSX*), et un sous licence libre GNU GPL (*pcsx*, récemment repris par de nouveaux développeurs sous le nom de *pcsx-df*).

3 Intérêts algorithmiques

Notre projet d'année d'InfoSPÉ doit contenir un intérêt algorithmique pour être validé. Sans surprise, Méta-morphosis contient de nombreux points algorithmiques représentant une part importante du projet, qui nous font penser que c'est une idée tout à fait appropriée comme projet d'InfoSPÉ.

3.1 Systèmes de fichiers

La programmation d'une bibliothèque émulant le BIOS de la PlayStation inclut le codage de fonctions d'accès au lecteur CD-ROM du PC et donc au système de fichier qui y est contenu. Le système de fichiers ISO-9660 est une structure de données faites pour être stockée sur des disques où l'accès à des données contigues est plus simple que l'accès à des données disparses, et est surtout utilisé en lecture seule, pouvant donc se permettre d'être peu efficaces pour l'ajout de données. Comme la plupart des systèmes de fichiers utilisés de nos jours, il se base sur une structure arborescente adaptée au problème, et des tables d'indices pour optimiser les accès aux répertoires.

De plus, la PlayStation utilise des CD-ROM multipistes, dans lesquels le système de fichier diffère légèrement, et dont peu d'implémentations existent. Cependant, ce format étant normalisé[13], il est abordable d'en programmer une implémentation pour la bibliothèque d'émulation du BIOS de Méta-morphosis.

3.2 Algorithmes de décompression

Le Data Decompression Engine de la PlayStation est une puce spécialisée dans la décompression des données utilisées par les jeux. En effet, avec la faible puissance du CPU, il aurait été peu envisageable de réaliser ces décompressions sur la même puce que celle gérant la logique du jeu.

Ce DDE gère deux algorithmes de décompression de données, très différents mais utilisés dans différents formats de textures et de vidéos présents dans les jeux de PlayStation :

- *Run Length Encoding*, une méthode de compression très simple et adaptée aux images présentant beaucoup de pixels identiques, utilisée notamment pour décompresser des bitmaps ;
- *Inverse Discrete Cosine Transform*, aussi nommé iDCT, qui est notamment l'algorithme utilisé pour la compression des images JPEG, et donc par conséquent des vidéos au format MJPEG[14] et au format H.261, plus

[13] Donc, par conséquent, documenté de manière exhaustive.
[14] Qui est, pour rappel, une simple suite d'images JPEG disposées linéairement.

adapté aux grosses vidéos telles que les cinématiques des jeux.

Ces deux algorithmes, et notamment l'iDCT, font partie des algorithmes classiques de décompression qu'il est donc intéressant pour un étudiant d'InfoSPÉ de reprogrammer : l'iDCT est en effet partiellement liée aux séries de Fourier et à la transformée de Fourier qui sont étudiées dans le programme de mathématiques de cette année. C'est également un algorithme qui travaille sur des signaux vidéos YCbCr.

3.3 Allocateur de mémoire

Le BIOS de la PlayStation fournit des appels systèmes réalisant le rôle de 4 fonctions standards C : `malloc`, `free`, `realloc` et `calloc`. Ces quatre fonctions d'allocation et de libération de mémoires opèrent dans la RAM de la PlayStation, ce qui fait qu'il est impossible d'utiliser les fonctions fournies par le système d'exploitation pour les émuler dans nos bibliothèques. Ainsi, il sera nécessaire de les recoder.

Plusieurs algorithmes peuvent être utilisés pour cela, via plusieurs structures de données gérant les blocs de mémoire alloués. Un grand classique grâce à sa simplicité est la gestion par liste doublement chainées : chaque bloc mémoire alloué contient un en-tête avec des liens vers le bloc suivant et le bloc précédent, ainsi que des informations sur le bloc (est-il libre, quelle taille mesure t-il). L'allocation consiste à chercher un bloc libre suffisamment grand dans la liste et à le casser en deux parties : une utilisée et une libre. La libération consiste, elle, à fondre le bloc avec les blocs libres présents à ses côtés.

Cependant, cet algorithme est simple mais a tendance à fragmenter la mémoire. Ce n'est pas envisageable pour un programme censé fonctionner avec une mémoire RAM de 2 Mio. Ainsi, il sera surement nécessaire d'utiliser des algorithmes plus perfectionnés.

À noter également que la reprogrammation d'un tel allocateur de mémoire fait partie des projets à réaliser en année d'ING1. Même si ce que nous allons faire ici sera surement plus simple qu'un allocateur devant faire fonctionner Firefox[15], c'est un bon exercice et l'expérience acquise sera utile plus tard dans nos études.

[15]Les programmes PlayStation utilisent assez peu d'allocations mémoire d'après les traces réalisées par les émulateurs.

3.4 Analyse de graphes de code

Le dernier algorithme exposé dans cette partie mais également le plus importante et le plus difficile est celui en rapport avec l'analyse du code à transformer. En effet, dans un exécutable, tout n'est pas code : des données sont parfois entremélées, et du code mort[16] peut être présent. Ainsi, il sera nécessaire d'analyser le graphe d'exécution du code, construit à partir des jumps et des appels de fonction, pour trouver les zones accessibles et celles qui ne le sont pas.

Malgré une explication simple, c'est un problème très complexe qui est en rapport direct avec la théorie de la compilation, et qui est encore sujet de recherches aujourd'hui : en effet, pouvoir détecter complètement les branchements dans un code et leur destination possible serait quelque chose de très intéressant pour les outils de désassemblages professionnels[17]. Même si ce programme n'est pas totalement résoluble vu qu'il dépend parfois de données externes et d'interactions de l'utilisateur, il l'est au moins partiellement, et suffisamment pour pouvoir traduire entièrement un programme de PlayStation.

De plus, la fin des années 1990 avait une problèmatique de moins qu'aujourd'hui qui nous facilite énormément la tâche : le piratage de jeux étant peu développé à l'époque, le code assembleur des jeux n'a jamais été obfusqué pour le rendre plus difficile à comprendre, ce qui simplifie énormément le travail du traducteur.

Il sera nécessaire pour la détection du code de pouvoir déterminer au moment de la traduction les différentes valeurs que peuvent prendre un registre ou une adresse mémoire, qui sont bien souvent déterminables quand ces valeurs sont utilisées pour accèder à du code (dans le cas des jump tables indexées par des énumérations, par exemple). Ainsi, il faudra backtracer ces valeurs dans le graphe d'exécution pour déterminer leurs bornes ou leurs ensembles de valeurs possibles.

Cette partie, qui est la plus importante du projet, est hautement algorithmique et sera également la plus longue et la plus difficile à réaliser. Néanmoins, nous espérons que notre motivation nous aidera à mener à bien ce projet.

[16] Du code mort est du code ne pouvant être atteint à aucun moment durant l'exécution, utilisé par exemple pour du padding ou généré par un mauvais compilateur.

[17] Cependant, ces outils se débrouillent très bien, comme par exemple le grapheur d'IDA Pro.

4 Organisation et répartition du travail

4.1 Communication intra-groupe

Échanger des mails est peut-être [18] un moyen tout à fait valable de communiquer et de partager le code source sur un petit projet avec deux ou trois personnes, mais devient très alambiqué dès que le groupe ou l'ampleur du projet grandit un tant soit peu. Ainsi, réflechir à la manière la plus efficace de communiquer est quelque chose d'important et qui permettra d'augmenter les chances de succès de notre projet.

Cette partie est découpée en deux moyens de communication différents qui ne concernent pas les mêmes échanges : tout d'abord, les échanges de code source, puis les échanges de texte de documentations acquis et synthétisés par les membres du groupe.

4.1.1 Partage de code

Pour permettre de programmer efficacement, il faut bien évidemment au groupe un moyen de partager le code que chaque membre a réalisé. Pour cela, nous avons décidé d'utiliser un DVCS [19], et plus précisément MERCURIAL.

L'utilisation d'une telle technologie façe aux VCS plus classiques tels que SUBVERSION nous permet premièrement de programmer n'importe où, même sans accès à Internet, tout en ayant à disposition l'intégralité de l'historique du code et en pouvant réaliser des *commits*, ce que SUBVERSION ne permet pas (ces opérations doivent être réalisées avec une connexion internet).

Un dépot central sera placé sur l'hébergeur gratuit de dépots MERCURIAL nommé *Bitbucket* [20], afin de garder une version synchronisée régulièrement en ligne. Ce dépot sera lié au système de *bug tracking* fourni par *Bitbucket*, afin de pouvoir réaliser des actions comme créer un bug ou fermer un ticket directement dans le message décrivant le *commit*, donnant ainsi la possibilité d'avoir une productivité maximale.

4.1.2 Partage de documentation

Bien que la PlayStation soit une vielle console, la firme à l'origine de sa création, Sony, n'a pas donné au public de documentation sur son fonctionnement interne. Ainsi, les seules informations que nous pouvons trouver sur

[18] Ça se discute...
[19] Distributed Version Control System.
[20] Voir `http://bitbucket.org/` pour plus d'informations.

ce point sont disponibles à deux endroits :

1. Dans le code source des émulateurs libres déjà programmés : c'est en effet pour nous la source la plus riche d'informations étant donné que notre projet est sur certains points très proche du travail réalisé par les émulateurs.

2. Sur les sites des équipes de *hackers* qui programmaient pour la PlayStation aux alentours des années 2000. Malheureusement, ces sites sont peu nombreux, souvent non maintenus et n'existent parfois même plus. De plus, la documentation qui peut y être trouvée peut être en partie fausse et/ou incomplète.

Pour avancer rapidement et efficacement, il est donc indispensable que quand quelqu'un trouve de l'information, il soit capable de la synthétiser et de la donner aux autres membres du groupe. Le partage de ces documentations se fera via un logiciel de wiki, fourni lui aussi par notre hébergeur de projet, *Bitbucket*. Cela permettra ainsi à tout membre du groupe de modifier les erreurs qu'il trouve, et également de rajouter sa propre documentation dans un format simple.

4.2 Répartition des tâches et objectifs graduels

4.2.1 Première soutenance

	Pierre	Florent	Tiphaine	David
GPU	50%			50%
SPU		0%	0%	
DDE			0%	0%
GTE	25%		25%	
Désassembleur	75%			
Traducteur	25%			
Filesystem		25%		
Gestion des manettes		25%		25%
Allocateur de mémoire		25%	25%	

Notre objectif pour cette soutenance est de faire tourner une démo de jeu simple comme ceux réalisées par *Candus*[21]. Ces programmes sont fournis avec leur code source, nous permettant de voir un peu plus clair dans ce que nous exécutons en premier lieu.

[21] Voir `http://www.candu.co.uk/psx/index.shtml` pour plus d'informations.

4.2.2 Deuxième soutenance

	Pierre	Florent	Tiphaine	David
GPU	75%			75%
SPU		25%	25%	
DDE			50%	50%
GTE	50%		50%	
Désassembleur	100%			
Traducteur	50%			
Filesystem		75%		
Gestion des manettes		75%		75%
Allocateur de mémoire		50%	50%	

4.2.3 Troisième soutenance

	Pierre	Florent	Tiphaine	David
GPU	100%			100%
SPU		100%	100%	
DDE			100%	100%
GTE	100%		100%	
Désassembleur	100%			
Traducteur	100%			
Filesystem		100%		
Gestion des manettes		100%		100%
Allocateur de mémoire		100%	100%	