Rapport de projet

ABIMN

Jeu de rôle

Projet de 1ère année

Rendu à Monsieur Martin Van Laere le 16 juin 2014

Groupe F0r M4g3

Laure *Lyloox* **Daumal** (*daumal_l*)
Viviane *DragonDatas* **Lair** (*lair_v*)
Nicolas *Pvut* **Lesquelin** (*lesque_n*)
Alexandre *Tsunami* **Manuel** (*manuel_d*)

Sommaire

1 Introduction

Depuis l'existence des jeux vidéos, on peut remarquer la présence de jeux de type jeu de rôle. Ceux ci permettent au joueur de vivre une aventure plus ou moins proche de la réalité. Le succès de ce genre de jeux tient dans le fait qu'ils permettent au joueur de s'échapper dans un univers parallèle. C'est pourquoi, nous, joueurs appréciant depuis notre plus tendre enfance ces aventures, avons décidé aujourd'hui d'en créer un à l'image de notre imagination la plus débordante. Créer un jeu de rôle est l'étape finale de cette évasion du monde réel. C'était l'occasion de ne plus être limité par l'univers d'un développeur que nous ne connaissons pas.

Notre groupe deprojet est constitué de quatre étudiants en première année à Epita. Laure DAUMAL, Viviane LAIR, Nicolas LESQUELIN et Alexandre MANUEL. Laure étant à l'origine de la création de ce groupe et de l'idée de créer un jeu de rôle, elle a naturellement été désignée chef de projet. En plus de satisfaire des rêves d'enfants, nous comptions bien acquérir une expérience non négligeable dans la création de jeu vidéo. La question de la 2D ou de la 3D s'est posée, et après une très longue reflexion d'environ dix minutes, nous avons décidé de rester en deux dimensions, ce qui nous permettrait d'avoir un jeu plus complet, mieux réalisé.

Il fut très vite remarqué que les jeux de rôles en vue de dessus manquaient incroyablement de dynamisme. C'est pourquoi nous avons choisi de créer un jeu moitié jeu de rôle, moitié jeu de plateforme, l'idée étant d'être un jeu de rôle classique devenant un jeu de plateforme à chaque combat, rendant les combats très dynamiques.

1.1 Récapitulatif

Rappelons le contexte dans lequel le projet s'est développé. Nous avions six mois pour faire notre projet entièrement. Durant ces six mois, trois soutenances étaient prévues. Pour chacune de ces soutenances nous nous étions fixés des objectifs que nous allons rappeler ici.

1.1.1 Première soutenance

	Lyloox	DDatas	Pvut	Tsunami	Total
Site Web	X	-	-	-	0%
Editeur	-	-	-	X	0%
Menu principal	X	-	-	-	50%
Editeur de personnage	-	X	-	-	40%
Interface d'exploration	X	X	X	-	50%
Menus ingame	-	-	X	-	50%
Interface de PNJ	X	-	-	-	0%
Interface de combat	-	-	X	X	30%
Intelligence artificielle	-	-	-	X	50%
Système de sauvegarde	-	X	-	-	0%
Ressources graphiques	X	X	X	X	30%
Ressources sonores	X	X	X	X	0%

Pour la première soutenance, le but est de pouvoir montrer un début de jeu. Le jeu devrait être jouable dans les bases. L'intelligence artificielle sera basique. Il devra être possible de jouer une partie et, au minimum, de la perdre.

1.2 Soutenance intermédiaire :

	Lyloox	DDatas	Pvut	Tsunami	Total
Site Web	X	-	-	-	100%
Editeur	-	-	-	X	100%
Menu principal	X	-	-	-	50%
Editeur de personnage	-	X	-	-	70%
Interface d'exploration	X	X	X	-	70%
Menus ingame	-	-	X	-	100%
Interface de PNJ	X	-	-	-	30%
Interface de combat	-	-	X	X	80%
Intelligence artificielle	-	-	-	X	70%
Système de sauvegarde	-	X	-	-	0%
Ressources graphiques	X	X	X	X	60%
Ressources sonores	X	X	X	X	50%

Pour la seconde soutenance nous prévoyons d'avoir un jeu quasi complet d'un point de vue gameplay. Certains points noirs subsiteront néanmoins (pas de sauvegarde, interfaçage avec d'autres personnages très basique, IA encore sur un seul niveau, ...)

1.2.1 Accompli jusqu'à présent

Pour la soutenance finale, le but était bien évidemment de mener tous nos objectifs à 100%. Malheureusement, tout ne s'est pas passé comme prévu et deux éléments manquent au jeu : le système de sauvegarde et l'éditeur de personnages. En revanche, nous avons implémenté un module de news dans le menu du jeu, ce qui n'était pas dans les exigences du cahier des charges. Les tâches ont également été redistribuées au cours de l'année. Voici donc le nouveau tableau en cette fin de projet :

	Lyloox	DDatas	Pvut	Tsunami	Total
Site Web	X	-	-	-	100%
Editeur	-	-	-	X	100%
Menu principal	X	-	-	-	100%
Interface d'exploration	-	X	-	X	100%
Menus ingame	-	-	X	-	100%
Interface de PNJ	-	X	-	-	100%
Interface de combat	-	-	-	X	100%
Intelligence artificielle	-	-	-	X	100%
Module de news	X	-	-	-	100%
Ressources graphiques	X	-	-	-	100%
Ressources sonores	X	X	X	X	100%

2 Choix techniques

Pour atteindre nos objectifs, il a bien sûr fallu faire des choix techniques. Ceux ci n'ont pas été faits sans réfléchir. Du langage de programmation aux logiciels de création de ressources en passant par la bibliothèque graphique, tout a été évalué.

2.1 Langage de programmation et librairie graphique : C# et XNA

De par les chartes du projet, nous ne pouvions choisir qu'entre deux langages de programmation : C# et Ocaml.
Pour choisir définitivement entre les deux outils (et trancher les désaccords qui rongeaient notre esprit d'équipe) nous avons effectuer quelques recherches sur les bibliothéques grahiques.
Passons en revue les différentes bibliothèques graphiques que nous avons trouvé pour ces deux langages :

OCaml :

- SDL
- Ocsfml
- ocaml-sfml
- librairie graphics par l'INRIA

C# :

- SDL.NET
- Binding SFML pour C#
- XNA

On voit déjà rapidement que la SDL et la SFML, qui sont deux bibliothèques graphiques qui sont portées en OCaml et en C#. Concernant la librairie graphics native d'OCaml que nous avons eu l'occasion de tester en TP, nous avons décidé de la laisser de côté car elle est très bas niveau et difficile à maîtriser par des novices en programmation.
Nous avons trouvé que le C#, par rapport à l'OCaml, est plus simple à prendre en main et plus proche d'une conception entièrement objet. Nous avons donc choisi le C# pour être notre heureux élu, tout d'abord pour des questions de confort.
Ajoutons à celà que pour développer en C#, Microsoft Visual Studio nous accompagne déjà bien plus que par exemple Emacs pour l'OCaml. Un environnement de développement du niveau de Visual Studio n'existant pas pour l'OCaml, nous avons encore penché en faveur du C#.

Revenons au choix de la librairie graphique. Il nous reste la SDL, la SFML et XNA. Parlons un peu d'XNA. Nous avons là une bibliothèque parfaitement en harmonie avec le langage car elle été développée spécifiquement pour le framework .NET Celle ci est entièrement objet et respecte donc bien la philosophie du C#. La SDL, elle, est réputée pour être une bibliothèque de nature impérative (elle a été créée pour le langage C à l'origine qui lui n'est absolument pas orienté objet). Celle ci étant une bibliothèque très bas niveau, nous avons également décidés de la laisser de côté.
La SFML elle, est belle et bien orientée objet, seulement, nous avons fini par préférer XNA pour plusieurs raisons :

1. Le support des élèves en Spé qui ont majoritairement utilisé XNA pour leur projet de Sup

2. La documentation sur msdn

3. Les différentes optimisations pensées pour le milieu du jeu vidéo par Microsoft

4. L'expérience acquise avec cette bibliothèque lors des TP d'Informatique Pratique

Concernant le choix de l'environnement de développement, nous nous sommes donc bien sûr tournés vers Visual Studio 2010, étant la plus récente version supportant encore XNA.

2.2 Outils de création graphique

Afin de procurer à notre jeu une identité visuelle, nous avons créé toutes nos images nous mêmes. Pour y parvenir, nous avons pu compter sur Laure qui, disposant déjà d'une certaine maîtrise sur le logiciel Photoshop, s'est servie de celui ci pour nous fournir toutes les ressources visuelles du jeu.

2.3 Recherche et retouche de musiques

Pour les musiques du jeu nous avons choisi de tout faire pour respecter les droits d'auteur, c'est pourquoi nous avons sélectionné toutes nos musiques sur le site jamendo.com, site de musiques libres. Certaines musiques, mettant trop de temps à se mettre en route par exemple, ont été rognées avec le logiciel mp3 direct cut.

3 Un Abimn connecté

Aujourd'hui il devient crucial d'être connecté pour avoir de l'importance, c'est pourquoi nous n'avons pas négligé cet aspect important des choses.

3.1 Site web

La première chose à faire était bien évidemment un site web. C'est sur celui ci que toutes les informations sur le jeu sont, et également sur celui ci qu'il est possible de télécharger notre jeu. Il était donc essentiel d'avoir un site web à jour, tout en respectant la charte graphique du jeu pour poser l'univers de jeu dès l'arrivée sur le site.

3.1.1 Mise à jour vers du HTML5

Pour la soutenance intermédiaire, notre site web était codé en xHTML. Pour la soutenance finale nous l'avons passé en HTML5. Le but est d'avoir un site à la pointe de la technologie. Le HTML 5 permettant en plus d'avoir un code plus propre et aéré que le xHTML nous l'avons privilégié dès qu'on en a eu connaissance.

3.1.2 Validation W3C

Le World Wide Web Consortium, abrégé par le sigle W3C, est un organisme de normalisation à but non lucratif, fondé en octobre 1994 chargé de promouvoir la compatibilité des technologies du World Wide Web telles que HTML, XHTML, XML, RDF, SPARQL, CSS, PNG, SVG et SOAP. Celui ci propose de valider en ligne un site web par rapport aux normes qu'ils imposent sur validator.w3.org Nous avons respecté à

la lettre leurs recommandations et notre site est à présent validé par le W3C.

3.1.3 Structure du site

Le site a été réalisé en HTML5 et en CSS3. C'est donc un site statique, qui se présente actuellement sous 4 pages :

- "Abimn.html", l'accueil du site web. Elle présente l'univers du jeu.
- "telechargement.html", la page proposant différents moyens de télécharger le jeu.
- "credits.html", une page de présentation de la "Team" f0rm4g3, les développeurs !
- "screenshots.html", la page de présentation de différents aperçus graphiques du jeu.

Chaque corps des pages du site est guidé sous la même structure :

- Une en-tête
- Une barre de navigation (ou "menu")
- Un contenu
- Un pied de page

Ces éléments correspondent tous à des conteneurs en HTML. L'entête contient une image, ainsi utilisée comme logo du site. Et, en réalité, la barre de navigation est une liste de liens qui a mangé du CSS. De même, les titres et paragraphes du contenu y ont goûté.

Le travail essentiel du CSS a été de donner une taille fixe au contenu, de permettre au menu d'être horizontal, et de rendre plus agréable la lecture du site. Il permet d'éditer énormément d'attributs : couleurs, ombres, bordures, marges, ... Ce qui permet aux codeurs d'être de bons graphistes ! On peut même en profiter pour réaliser de petites animations (ici utilisées sur les liens et les titres) bien qu'il existe de meilleurs outils dans ce domaine.

Tous les rendus graphiques en CSS sont refaisables sous Photoshop, alors que l'inverse n'est pas vrai. C'est pourquoi nous aimons réaliser mes sites à base de jolies images. En CSS, notre background est de couleur fixe - nous avons choisi du violet, c'est la charte graphique qui nous l'oblige.

3.1.4 Effets graphiques

Les effets graphiques que l'on peut observer sont dûs au CSS (Cascade Sheet Source). Par certains mots-clés, on peut modifier l'apparence d'un conteneur dans sa forme statique et la faire évoluer en fonction de l'action de la souris de l'utilisateur sur certains éléments. C'est qui permet de donner de la beauté à un statique, et une utilisation plus dynamique.

3.2 Réseaux sociaux

3.2.1 Réseaux classiques

Pour mettre en avant notre jeu, nous avons créé une page Facebook et un compte Twitter pour notre jeu disponibles à ces adresses :
fb.com/pages/Abimn/280334482149265
twitter.com/Abimn_F0rM4g3

3.2.2 Github

GitHub (exploité sous le nom de GitHub, Inc) est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Ce site est développé en Ruby on Rails et Erlang par Chris Wanstrath, PJ Hyett et Tom Preston-Werner. GitHub propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres.

Le nom Github est composé du mot "git" faisant référence à un système de contrôle de version open-source et le mot "hub" faisant référence au réseau social batît autour du système Git.

C'est ce dernier (le réseau social) qui nous a fait choisir GitHub. Celui ci est pour nous une opportunité de présenter facilement nos travaux à de futurs employeurs.

3.3 News dans le jeu

3.3.1 Webservice

Un service web est un programme informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.

Ici c'est celui qui fournit le XML à notre jeu vidéo.

3.3.2 Structure XML

Nous utilisons un fichier XML afin de pouvoir partager nos nouvelles informations (actualités, mises à jour, etc) à la fois sur notre support web et sur la page d'accueil de notre jeu. Il se présente sous la structure suivante :

<news>
<title>Titre</title>
<date>Date</date>
<content>Contenu</content>
</news>

4 Un Abimn organisé

4.1 Architecture

Dès le début, nous avons compris qu'XNA seul ne suffirait pas. Il fallait ajouter une surcouche à XNA. Quelque chose pour simplifier la bibliothèque. Nous n'avons pas cessé de faire évoluer cette architecture au cours de l'année pour avoir finalement une surcouche proche de la perfection. Cette architecture est décomposée en plusieurs modules détaillés ci dessous.

4.1.1 Gestion des ressources

C'est sans doute le module qui a le plus changé au cours de l'année. A l'origine, celui ci ne gérait que les images. Celles ci étaient chargées au démarage du jeu et stockées dans un tableau linéaire. Chaque image était ainsi repérée par un identifiant. Néanmoins, cette technique avait beaucoup de défauts :

- Tous les fichiers devaient être nommés avec des chiffres
- Il fallait indiquer précisément dans le code le nombre d'images total dans le projet. En cas d'erreur dans le nombre, le jeu plantait.
- Impossible d'avoir deux images avec le même numéro
- Les numéros devaient être réservés dans l'ordre

Notre principal soucis à ce moment là a été de communiquer au reste du groupe l'ajout de chaque image en temps réel. Lorsque nous codions tous ensemble, ceci réduisait drastiquement notre productivité.
Nous avons ensuite changé d'approche en passant sur un tableau bidimensionnel. Le principal avantage était que l'on pouvait maintenant ajouter chacun des images de notre côté sans avoir à en informer les autres de manière immédiate.

La première dimension était représentée par une énumération. Chaque élément de l'énumération correspondait à un tableau d'images. Malheureusement, un chiffre qui n'était pas à jour à la première soutenance a fait planter le jeu sur l'affichage d'une image qui n'était donc pas chargée.
Pour la deuxième soutenance, nous avons légèrement simplifié le système. En effet, il fallait auparavant modifier trois fichiers différents pour ajouter un tableau d'images au grand tableau. Nous avons simplifié la démarche en permettant de ne modifier qu'un seul fichier.
Pour la troisième soutenance, nous avons entièrement revu le système. Pour notre nouvelle gestion des ressources, nous avons utilisé une hashtable.

Hashtable

Une hashtable est une structure de données permetant de lier des paires de clés/valeurs entre elles avec une compexité minimale ($o(1)$). Une hashtable est en fait un tableau de valeurs dont les clés, une fois transformées par les fonctions de la hashtable, deviennent les indices.
Le principe d'une hashtable repose sur deux fonctions détaillées ci dessous :

- Une fonction de hachage (qu'on appellera $h_h()$) : C'est elle qui va transformer la clé en une suite de bits la plus unique possible, ce sera le hach code de la clé
- Une fonction de compression (qu'on appellera $h_c()$) : Cette fonction va transformer le hach code en un entier borné de manière à entrer dans la tableau qui représenta la hashtable

La condition que la hashtable doit respecter est la suivante :

$$k_1 = k_2 \iff h_h(k_1) = h_h(k_2)$$

Si l'implication de la gauche vers la droite est simple à mettre en place, la réciproque est impossible à mettre en oeuvre. Les clés pouvant être des suites infinies de bits et les hach codes étant limités en taille, on trouvera forcément des collisions.

Vient alors le système de gestion des collisions. On trouve ainsi trois principales manières de gérer les choses :

- Le sondage linéaire : Le principe est simple, en cas de collision, on prend la première case libre du tableau à partir de l'indice obtenu.

Cette technique a certains désaventages comme le fait que les clés s'aglutinent en grappes dans le tableau ce qui peut réduire les performances. Une solution pour éviter cela est de faire un sondage quadratique mais si le problème est minimisé, il n'en est pas moins présent.

- Le double hachage : Peut être vu comme un extension du hachage quadratique. Au lieu d'élever une partie de l'indice final au carré, on va le hacher à nouveau pour l'ajouter à l'indice précédent. Si la fonction de hachage est bonne, il devient très facile d'éviter les grappes. En revanche, cela implique d'avoir une fonction de hachage qui passera par tous les indices possibles au moins une fois et cela peut drastiquement réduire les performances si la hashtable est trop remplie.
- Le chaînage linéaire : Ici l'idée est d'avoir une file sur chaque élément du tableau. Ainsi, en cas de collision, on ajoute simplement notre élément à la file sans le décaler dans le tableau. Ceci va avoir pour inconvénient de faire grimper la complexité de recherche et insertion en flèche si le nombre de collisions est trop important.

Le langage C# nous propose une implémentation générale de hashtable. Celle ci ne se base pas sur un tableau de taille fixe (on peut y insérer jusqu'à deux milliards d'éléments dynamiquement)

L'idée a donc été de créer des couples clés/valeurs de chaînes de caractères/ressources. Ainsi, au lieu de repérer une ressource par un numéro, on la repère par un nom (accessoirement le chemin d'accès sur le disque)

Scan récursif de l'arborescence

Pour simplifier encore plus le système, au lieu d'ajouter chaque fichier à la main, le programme va analyser l'arborescence du dossier de ressources du jeu pour y trouver tous les fichiers présents. En ouvrant le répertoire de ressources, le programme va aller charger séparément les images, les sons, les musiques et les polices d'écriture. En ouvrant un dossier, il va charger tous les fichiers présents à l'intérieur puis relancer l'opération sur tous les répertoires présents dans le dossier.

Surcouche d'utilisation

Une fois que la hashtable contient bien toutes les ressources, pour utiliser ces ressources, nous avons créé une surcouche afin de ne pas avoir à accéder à la hashtable en dehors de cette surcouche. On retrouve ainsi une surcouche pour les images, les musiques, les sons et le dessin de texte. On trouvera dans le reste du code uniquement des ordres donnés aux surcouches.

4.1.2 Système d'entités

Voilà encore une surcouche du système. Nous sommes partis du principe que tout élément qui peut se dessiner à l'écran est une entité. Ainsi, monstres, héros, case de la carte, bouton, tout est une entité.

Généralisation par rapport aux Button

A la base, nous avions séparé le concept de bouton du concept d'entité (les boutons ayant trois références d'images, une image normale, une image lorsque le bouton est survolé et une dernière lorsque le bouton est enfoncé) mais nous nous sommes vite rendus compte qu'il était dans notre intérêt de généraliser complètement le principe d'entité. Les images étant représentées par des chaînes de caractères ne prenant que peu de mémoire, mettre une ou trois images dans chaque entité ne change pas vraiment les besoins en mémoire.

Les caractéréstiques techniques des entités

On peut spécifier pour notre entité le fait qu'elle soit visible ou non, un vecteur de déplacement sur l'écran, un zoom à appliquer, une rotation en radians, un événement lors du clic, etc Les entités doivent être capables de gérer toutes les demandes du programmeur.

4.1.3 Pile d'états de jeu

Premièrement, nous avions envisagé une architecture de code plutôt vieille. Tout était une approche impérative du problème.

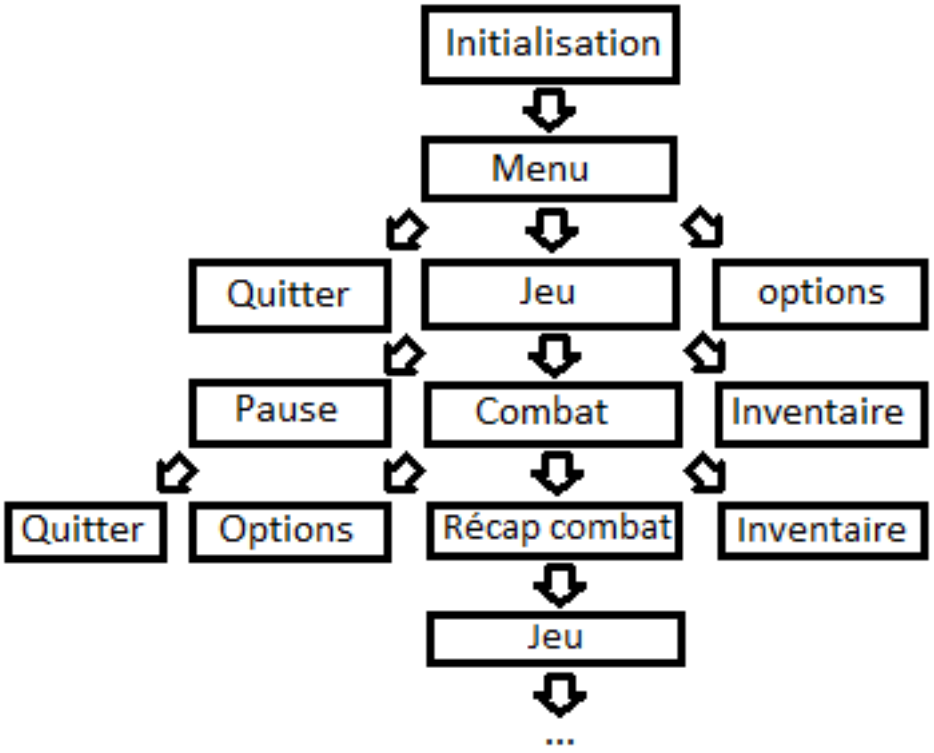

FIGURE 4.1 – Architecture originelle du jeu

Le programme commençait par faire des initialisations générales puis appelait une fonction qui en appelait une autre qui en appelait elle même une autre, etc.....En bref, on peut voir sur la figure 4.1 quel était le schéma de fonctionnement du jeu.

Cependant, cette architecture empêchait clairement d'utiliser la puissance d'XNA. En effet, il est ainsi bien difficile de dissocier calculs de jeu, du dessin des entités à l'écran alors qu'XNA bénéficie d'optimisations intéressantes sur ce point de vue (telles que la diminution de FPS quand le jeu perd le focus)

C'est pourquoi nous avons décidé de changer de système pour gérer tout ceci différemment, de manière plus adaptée au fonctionnement d'XNA.

Tout est basé sur une pile. C'est l'élément au sommet de cette pile qui définit quel est l'élément du jeu à faire fonctionner à un instant t. Comme un schéma vaut mieux qu'un long discours, la figure 4.2 illustre mes propos.

Cette pile est l'équivalent de la pile d'appel de fonctions dans la première architecture. Voyons cependant ce qui change. Premièrement, chaque pierre de la pile est en fait liée à un ensemble de fonctions qui correspondent (comme par hasard) aux fonctions de l'architecture classique

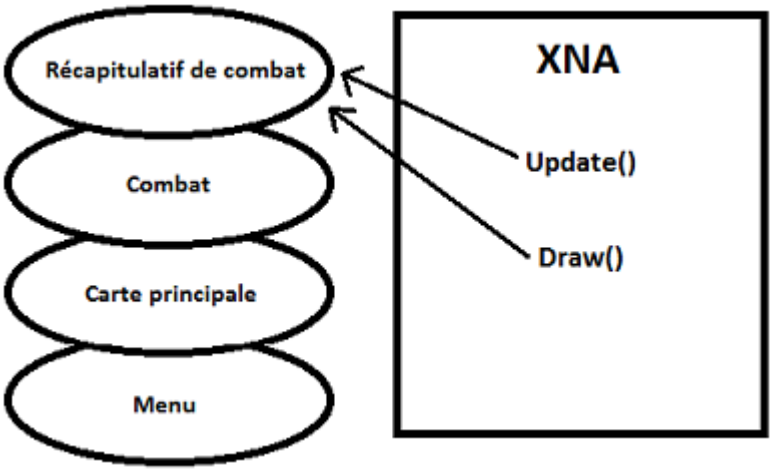

FIGURE 4.2 – Seconde architecture du jeu

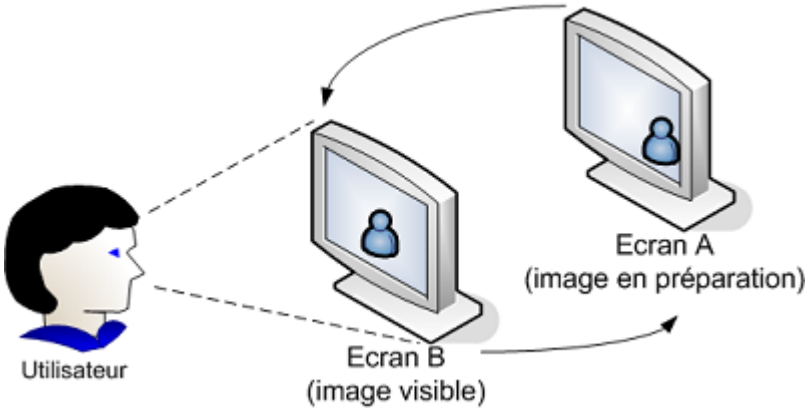

FIGURE 4.3 – Le principe du double buffering

d'un jeu fait avec XNA.
Chaque pierre est donc composée d'une fonction Update, d'une fonction Draw et d'une fonction Initialize. Cette dernière est appelée lors de la pose de la pierre concernée sur la pile. Quand à Update et Draw, seules sont appelées celles de la pierre du haut de la pile. Il est en effet inutile de continuer de dessiner le menu alors que nous sommes en plein jeu.
Seulement un problème s'est alors posé : XNA utilise une technique appelée le double buffering expliqué sur la figure 4.3
Le principe est d'avoir deux écrans en mémoire. Le premier (écran A) n'existe qu'en mémoire contrairement au deuxième (écran B) qui représente l'écran en préparation. Quel est l'intérêt de cette technique ? La phase de dessin d'une frame peut être assez longue. Ainsi, pour ne pas dessiner sous les yeux de l'utilisateur et ainsi faire scintiller son écran,

FIGURE 4.4 – Le menu n'occupe pas tout l'écran

on préfère préparer tous nos dessins dans un endroit caché en mémoire (écran A) pour ensuite, à l'affichage demandé de la frame, échanger les deux écrans. L'échange d'écrans étant extrêmement rapide, c'est invisible pour l'utilisateur et il n'aura ainsi pas l'impression de voir la frame se construire.

Cette technique a un petit défaut pour nous. XNA, à l'échange des deux écrans, commence par effacer l'écran qui se retouve à l'arrière afin de nous permettre de partir d'une base de dessin propre. Seulement dans le cas du récapitulatif de combat par exemple, on aimerait garder la dernière frame du combat dessinée derrière le récapitulatif. Celui-ci n'occupe pas la totalité de l'écran comme on peut le voir sur la figure 4.4

Il en devient ainsi nécessaire de remonter dans la pile pour dessiner ce qui se trouve en dessous de notre élément flottant. Pour ne pas dessiner d'éléments qui ne seraient de toute façon pas visibles, toutes les pierres ont un attribut qui détermine si celle ci prend la totalité de l'écran ou non. On va donc dessiner les pierres en commençant par la première pierre qui occupe tout l'écran et non pas en commençant par la pierre la plus profonde. On se retrouve donc avec le schéma de la figure 4.5

Bien évidemment on ne veut qu'un aperçu de ce qui se trouvait en dessous du récapitulatif. On ne veut pas que le combat continue de se dérouler. C'est pourquoi seule la méthode Draw est appelée sur la pierre Combat et pas la méthode Update (on voit ici l'intérêt de séparer le dessin du calcul).

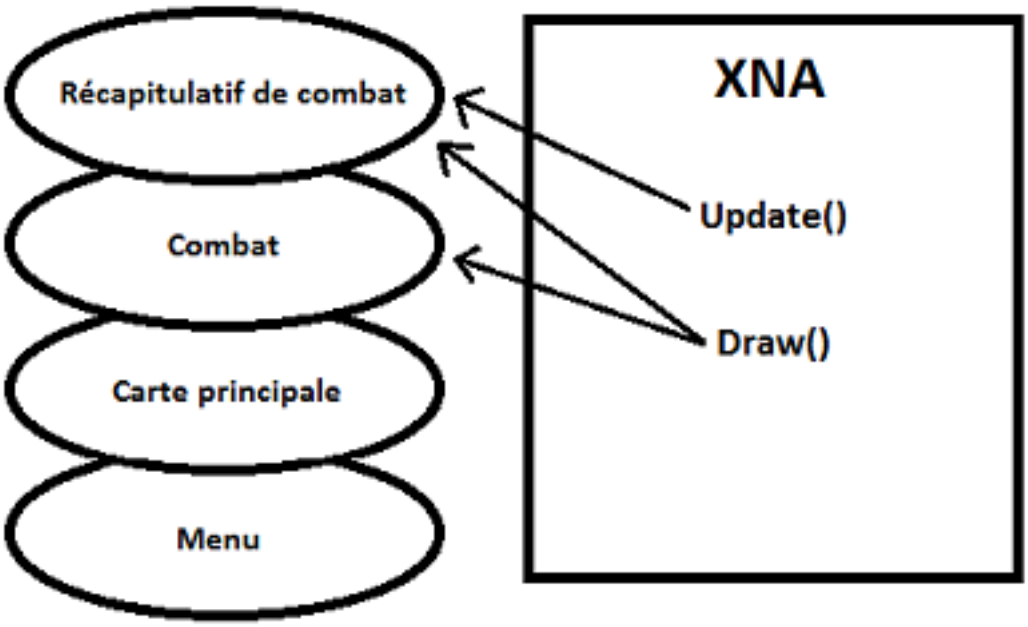

FIGURE 4.5 – Correctif de la seconde architecture du jeu

4.1.4 Capture des événements

Ici encore, XNA nous laisse avec peu de moyens. Il a donc fallu ajouter une couche de code pour corriger cela.

Système de flash des périphériques

Nous avons la possibilité grâce à XNA de prendre comme une photo de l'état des périphériques d'entrée (clavier, souris, ...) mais d'une part ce flash est lent et d'autre part il est impossible de savoir de cette manière depuis combien de temps une touche est enfoncée. Pour régler les problèmes de vitesse, la prise de cette photographie est donc faite une seule fois par passage dans la boucle de jeu.

Buffering des flashs

Afin de savoir si une touche vient d'être enfoncée à un instant t, il faut non seulement qu'elle soit enfoncée à l'instant t, mais également qu'elle ait été en position haute à l'instant t-1. C'est pourquoi il est nécessaire de garder en mémoire deux flash. Le flash de l'instant t et le flash de l'instant précédent. On se rapproche très fortement de la psychologie d'un double

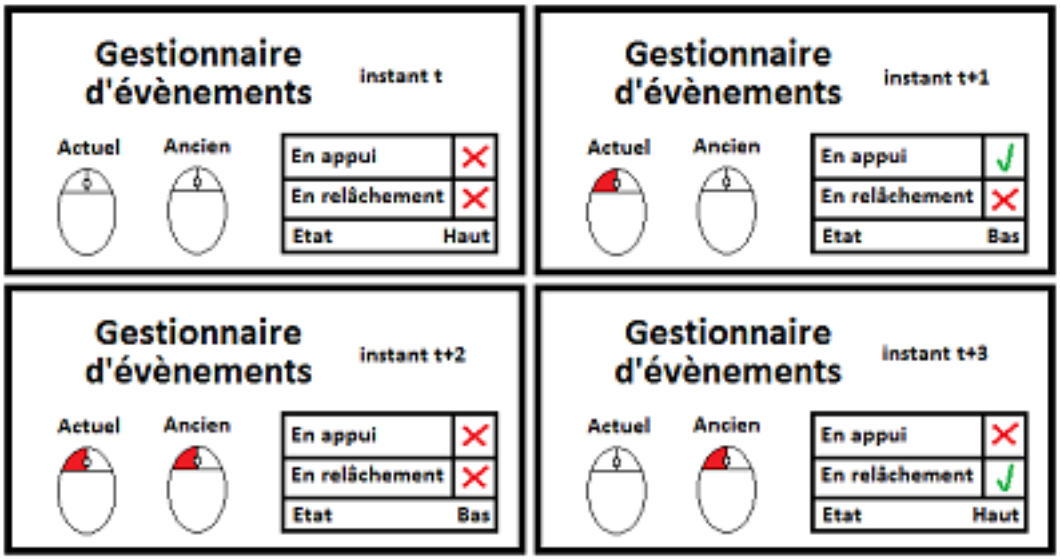

FIGURE 4.6 – Fonctionnement de la souris

buffering, mais à l'envers. Ce qui est conservé dans les treffonds cachés de la mémoire, c'est l'ancien flash et non la future frame. Le principe est évidemment le même pour les boutons de la souris. Le schéma de la figure 4.6 illustre le fonctionnement de la souris pour son clic gauche.

Surcouche d'utilisation

Toujours pour simplifier la vie des programmeurs du projet, tout ceci est bien caché. Le programmeur va donc non pas évaluer lui même des différences entre les deux flashs, mais va demander à la surcouche si une touche vient d'être enfoncée ou relâchée. C'est cette surcouche qui se chargera de faire toutes les opérations de manipulations des flashs.

4.1.5 Curseur

Afin d'avoir un curseur personnalisé en jeu (chez nous c'est une épée) nous avons créé une surcouche pour le curseur. Il est ainsi possible de rendre la souris visible ou non à la volée partout dans le programme. L'idée a été d'avoir une entité représentant le curseur de la souris pour pouvoir, si on le désire, faire en sorte que la souris soit une image classique.
Une fois l'ordre de changement de ressource pour représenter le curseur de la souris donné, les surcouches se chargeront de le changer.

5 Un Abimn modulable

Pour améliorer l'expérience de jeu et sa durée de vie, nous avons tenté de rendre le jeu le plus modulable possible. C'est ainsi que nous avons pensé notre projet en fonction de constantes et que nous avons créé un éditeur de cartes.

5.1 Editeur

Nous avons pensé à créer un éditeur uniquement à l'attention des utilisateurs expérimentés. En effet, la modification de certaines cartes par un joueur lambda pourrait nuire au gameplay en lui même (si le joueur décide délibérément de laisser la carte vide, il ne pourra plus profiter du jeu).

5.1.1 Ergonomie

Afin de garder une productivité optimale pendant la création du contenu du jeu, nous avons essayé de rendre l'éditeur particulièrement ergonomique. Ainsi, on distinguera dans l'éditeur le fond d'une case sur la carte et les décorations posées dessus. Le curseur de la souris devient la case que l'on souhaite poser. Un clic droit permet de passer des fonds de cases aux décorations et vice versa, un clic gauche permet de poser l'entité sélectionnée sur la case de la carte survolée, un clic molette sélectionne le bloc de la case visée, un appui sur la touche effacer permet de supprimer toutes les décorations posées au préalable sur la case survolée, un appui sur la touche S permet de sauvegarder la carte, un appui sur la touche R permet de faire revenir la carte à son état lors de la dernière sauvegarde et un appui sur la touche échap permet de revenir au menu. Dans le cas où la carte n'est pas sauvegardée et que l'utilisateur tente

de retourner au menu avec Echap, celui ci est prévenu que ses derniers changements ne seront pas enregistrés.

5.1.2 Caméra

Au départ, nous avions mis en place un déplacement de la caméra lorsque la souris atteignait de trop près un des bords de l'écran de l'éditeur. Pour des questions de fluidité et de confort d'utilisation, nous avons rendu la vitesse de la caméra dépendante de la distance entre la souris et le bord de l'écran. Ainsi, plus la souris est proche du bord, plus la caméra bouge vite. Celle ci peut aussi se déplacer avec les flèches directionnelles du clavier, et encore une fois, la vitesse de déplacement de la caméra est variable. Plus l'appui est long, plus la caméra accélère. Nous avons évidemment dû établir des bornes de vitesse et de position de la caméra. Le point central de la caméra est donc de manière systématique un point de la carte. On aura donc au maximum les trois quarts de l'écran sur du vide (si on positionne la caméra sur un coin) mais jamais la totalité. La vitesse, elle, est fixée entre 1 et 50 pixels par seconde.

5.1.3 Sauvegarde

Au départ, nous avions pensé à un système de sauvegarde assez complexe. Toutes les cartes étaient enregistrées en clair dans un fichier, une carte par ligne, avec la structure suivante :

largeur ;bloquant :sol :deco1 :deco2 :[...] :decoN ;bloquant :sol :deco1 :[...]

Largeur représentait la largeur de la carte (qui est soit dit en passant forcément rectangulaire), bloquant pouvait être égal à y ou à n pour un case bloquant ou non, sol correspondait à l'identifiant du sol de la case et suivaient déco1, déco2, etc qui étaient d'éventuelles image sur fond transparent qui sont ajoutées sur la case. On pouvait donc par exemple trouver dans le fichier ce type d'informations :

5 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;
n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1 ;n :1
qui correspond donc à la carte sur la figure 5.1

Par la suite, et grâce aux TP de C# d'Informatique Pratique, nous avons appris l'existence de la sérialisation. Ce procédé permet de sauve-

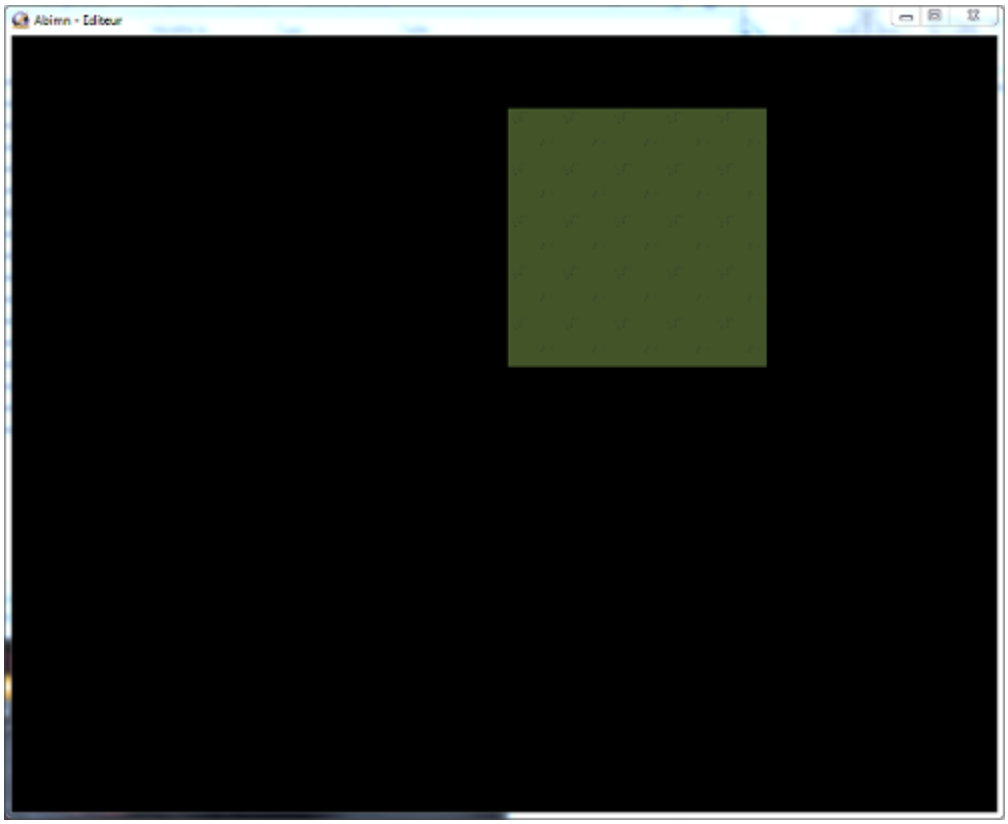

FIGURE 5.1 – Capture d'écran de l'éditeur

garder un objet dans un fichier sans se soucier de la manière dont il est enregistré. Le procédé symétrique, qui nous permet de charger une carte depuis un fichier s'appelle la désérialisation. Le principe de ce procédé est de réduire un objet logique en suite d'éléments atomiques. L'enregistrement de l'objet étant une suite de bits, le langage C# se charge des problématiques d'alignement des données et d'endianness. Ainsi, on ne retrouve aucun alignement, les encodages des types C entiers se font en fonction de leur empreinte mémoire tous au format big-endian et les nombres en virgule flottante utilisent la norme IEEE 754.

5.2 Constantes déclarées

Une des manières d'éviter de passer des heures à chercher tous les endroits du code à modifier dès que l'on veut modifier une simple valeur comme la largeur de l'écran est de définir des constantes et de les rassembler. Ainsi, dans tout le reste du code, aucune valeur n'est posée arbitrairement. Si l'on veut simplement modifier une valeur précise, il

faudra passer par le fichier de constantes. On y retrouve ainsi :

- La largeur et la hauteur de la fenêtre de jeu
- Les chemins d'accès aux dossiers de ressources
- La taille d'une cellule de la carte
- Les dialogues de jeu

6 Menu du jeu

6.1 Respect de la charte graphique

Nous avons une charte graphique dont la couleur dominante est un violet sombre. Afin de contraster cette couleur, nous avons pris des teintes claires secondaires à base d'orange et de bleu. La charte graphique permet de récupérer le code exact de ces couleurs, ce qui me donne la possibilité de créer toujours de nouvelles images sans donner une impression de "décalage" vis-à-vis des plus anciennes créations basées sur la même charte.

6.2 Lecture des news avec le webservice

L'affichage des news n'étant pas du tout prévu par XNA, nous avons dû recoder une sorte de petit moteur de rendu. Ainsi, après le téléchargement du fichier XML par notre programme, celui ci l'analyse grâce à l'objet XDocument fourni par le langage C# et nous le mettons en forme dans le cadre associé aux news.

6.3 Options

6.3.1 Hitboxes

La hitbox des boutons se calcule automatiquement en fonction de leur dimension et de leur position. Sans elle, nos boutons ne seraient pas capable de rediriger l'utilisateur et ainsi de remplir leur rôle.

Nous avons, en effet, créé une classe spécifique à tous nos boutons qui

permet de simplifier nos calculs et de les rendre aussi automatiques que l'on peut – si je décide de changer tous mes boutons, leur forme et leur format, leur hitbox en jeu s'adaptera de manière optimale.

6.3.2 Gestion de la molette et du clavier

Nos options permettent de gérer la luminosité du jeu ainsi que le volume de la musique. Pour cela, l'utilisateur a deux possibilités :

- Il peut bouger le curseur avec sa souris, qu'il place là où il le souhaite.
- Il peut sélectionner le critère qui l'intéresse et modifier sa valeur à l'aide de sa molette de souris ou de son clavier.

De cette manière, l'utilisateur pourra gérer ses options de la manière qui lui paraît la plus naturelle.

7 Carte générale : Exploration

7.1 Tile Mapping

7.1.1 Choix de la représentation graphique

L'un des premiers choix auquel à été confronté notre équipe lors de la création de notre projet à été le choix du type de carte que nous voulions utiliser et le type d'exploration pour notre personnage L'alliance des deux devait être suffisament agréable pour le joueur, avec une prise en main facile et surtout possédant un niveau de code à la portée de tous les membres du groupe. Plusieurs choix s'offraient à nous :

1. Tout d'abord, savoir si nous devions faire le jeu en 2D ou 3D. La jeu en 3D aurait, dans l'absolu, donné un bien meilleur rendu et aurait donné au côté exploration un aspect peut-être plus attractif. Mais compte tenu des possibilités de notre équipe en création artistique et manipulation d'image, rendre les graphismes suffisamment fluide et fourni pour faire des paysages agréables et immersifs compatible avec le type d'univers onirique que nous recherchions, aurait été tr'es complexe pour ne pas dire pratiquement impossible. Par égard pour les yeux de nos futur joueurs, nous avons donc très vite rayé cette possibilité

2. Restait encore à choisir entre la 2D et la 2D isométrique. Des jeux trés récents ont fait le choix de revenir a un style de graphisme en 2D isométrique, preuve s'il en est du plébiscite de ce système,utilisé dans un nombre incalculable de jeu, aussi bien des jeux de console que des jeux d'ordinateur RPG tel que *le légendaire Baldur's Gates*. Mais l'idée de tabler sur un décalage d'utiliser le côté extrèmement

rétro du graphisme 2D plat pour trancher sur l'univers (pouvant accueillir aussi bien fantastique que science fiction) était également a envisager.

3. Là dessus est venu le problème du second mapping : l'écran de Combat l'Utilisation de la 2D, isométrique ou non, n'allait pas avec le dynamisme que nous voulions donner á nos séquences de combat. Nous avions décidé de faire des affrontements en temps réel et non au tour par tour, plus statique, strategique et calculatoire que purement ludique. Comptant sur l'effet addictif des combats pour redynamiser l'ensemble du jeu, une idée nous est venue : pourquoi ne pas utiliser une instance de map totalement différentes. Nous avons donc décider de présenter les combat de profil, comme un MarioBros, alors que l'exploration et les actions sociales seraient gérés depuis une carte plus conventionnel et trés siimple. Les deux seraient en pixel art et 2D pour garder le côté décalé.

7.1.2 Les classe Cell/Map pour le TileMapping

Le plus simple pour créer la carte d'exploration principale était le tile mapping. Détaillons ensemble un peu cette technique.
Le principe d'avoir une mao constituée d'un tableau de cases Chaques case est iddentifées par un couple de coordonnées. Il devient aunsi très simple de faire une gestion de collisions entre le personnage et son environnement. Le personnage ne fait que se déplacer de case en case, dans les cases qui veulent bien l'accepter.
Cette technique permet :

- Une gestion simplifiée des collisions
- une économie non négligeable de mémoire d'un point de vue quantité d'images
- Un import/export simplifié de cartes dans des fichiers
- L'existence d'un éditeur puissant
- Une carte fractionnée, et donc affichable par partie

Afin d'avoir une grande carte, le héros se déplace dans une carte qui bouge selon ses mouvements. Grâce au Tile mapping, seule la partie visible de la carte est affichée. Le seul défaut de cette technique est

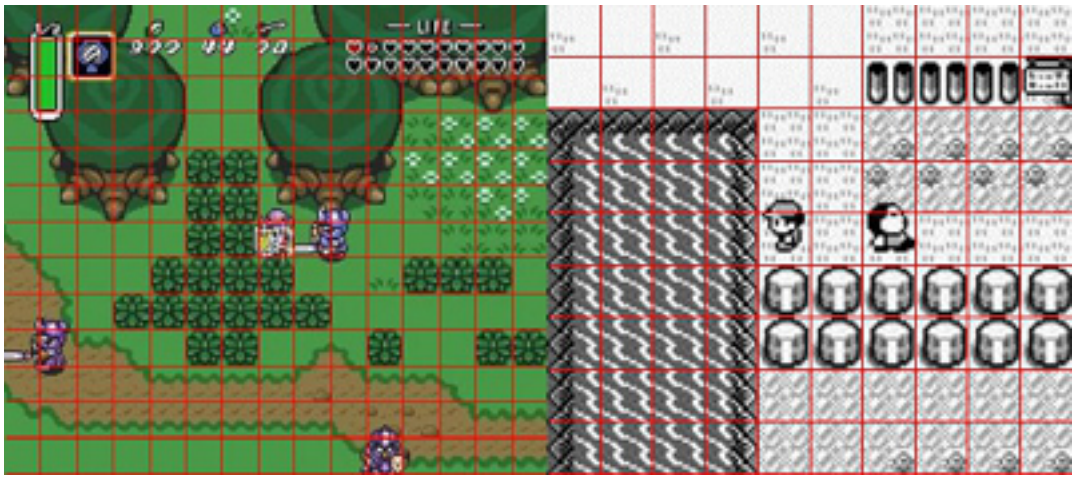

FIGURE 7.1 – Capture d'écran retouchée de Zelda et Pokémon

un manque de liberté graphique dans la carte du jeu. Néanmoins, cette technique est très utilisés encore de nos jours dans de très nombreux jeux vidéos en deux dimensions.
L'utilisation de cette technique est flagrante dans des jeux tels que les premiers Pokémons ou les premiers Zelda comme sur la figure 7.1

7.2 Gestion des cellules : files de décorations

La particularité de notre TileMapping est que chaque cellule est représentée par une file contenant les indice du fond et des différentes décoration à y ajouter. Cette méthode permet de dessiner simplement une superposition d'objet aux propriétés diverses à une position bien précise dans notre tableau de tuile. Chaque Cellule a en plus un indice booléen pour savoir si elle bloque les déplacements ou non.

7.3 Animation du personnage

Dans la Map d'exploration, nous devions avoir une vue d'ensemble du monde dans lequel évolue le personnage. Comme nous l'avons vu précédemment, la méthode la plus pratique de pouvoir explorer de grandes cartes tout en gardant l'effet d'ensemble était de centrer la caméra sur le héros.

Au début, nous avions choisit le Tile Mapping car il nous permettait une implémentation des déplacements relativement faciles grâce au système de cases, une gestion simple des collisions grâce à un ensemble de cases bloquantes et non bloquantes ainsi qu'un dessin du fond relativement simple a obtenir et peu coûteux en mémoire. Mais nous sommes rapidement aperçut que ce système avait ses limites à cause de la grande taille de nos tuiles. Cela nuirait grandement à la fluidité des mouvements de nos personnages et de la caméra, rendant l'exploration ennuyeuse voire désagréable.

Entre les deux premières soutenances, nous avons donc décidé de fixer le héros, non plus par rapport à une tuile mais par rapport à des coordonnées. Tout a été recentré sur le pixel d'origine du sprite du personnage principal et les mouvements traduits en pixel par pixel. Le mouvement obtenu est alors très fluide malgré un problème de vitesse de déplacement (réglée à l'aide de timer, ce qui a permis d'ajouter une possibilité de course) et de cohérence graphique au niveau des collisions.

Un autre avantage du positionnement par pixel est de pouvoir plus facilement varier la vitesse du personnage et instaurer ainsi une possibilité de changement marche/course. En plus de changer de sprite en fonction de la direction que le personnage empreinte, nous avons décidé d'implémenter une série différente de sprite pour la marche et la course a pied au lieu d'accélérer simplement le défilement des images. La variation graphique est plus agréable à l'oeil.

7.4 Gestion des collisions

Au début très simple grâce au TileMapping, les collisions ont du être corrigées entre les deux premières soutenances à cause du nouveau style de mouvements.

Le système de base est un principe de cases déterminées comme bloquantes ou non lors de la création de la Map sur laquelle évolue le personnage. Quand le héros veut entrer dans une case, l'autorisation est donné en fonction de cette variable et les collisions se faisaient automatiquement. Mais avec un décalage de trois pixel par trois pixels, les mouvements du personnage ne correspondaient plus exactement aux Tuiles. Les cases bloquantes n'étaient pas graphiquement respectées. Pour y remé-

FIGURE 7.2 – Zone de visibilité du personnage

dier, nous avons du régler manuellement les condition de changement de Tuile.

7.5 Zone de visibilité du personnage

Le déclenchements des événements et des interaction avec les PNJ reposent sur un système de champ de vision du personnage. En appuyant sur la touche d'interaction pendant qu'un PNJ ou toute autre sorte d'événement se trouve dans la zone de visibilité, l'instance correspondant à l'interaction est déclenchée. Pour avoir une visibilité assez large nous avons choisit de faire une zone de plusieurs cases en avant du personnage. Voir la figure 7.2 pour plus de détails.

L'instance lancée à lors de l'interaction avec la plus part des PNJ appelle les lignes spécifiques au personnage comme on appellerait une Map spécifique lors d'un changement.

7.6 Changement de carte

L'un des buts de l'interface d'exploration est certes de bouger sur une carte et de pouvoir interagir dans une certaine mesure, avec son environnement, mais il faut penser également à tous les à côtés, et notamment à pouvoir changer de carte, sans quoi le monde du jeu sera très réduit. Avec notre méthode de création de carte, les fichiers restaient relative-

ment légers. Le temps de chargement n'était donc pas un problème. Pour déclencher l'appelle d'une nouvelle map dans le l'interface d'exploration, il suffit de passer sur certaines cases de voyage spécifiées lors de la création de la carte. Ces cases sont associées à l'identifiant de la map qui doit être appeler et changent.

Pour éviter les boucle de changement de map (le personnage reste sur la case ou avance d'un coup sur l'autre map et passe a travers un autre portail) nous avons rajouter un timer. Le personnage ne peut bouger qu'après un laps de temps prédéfinie. Ainsi le joueur n'est pas désorienté au changement de carte et n'a pas l'impression de continuer à avancer alors que le paysage change spontanément autour de lui.

8 Combat

Ce qui rend notre jeu dynamique, c'est en partie ses combats instanciés. Lors d'un combat, la vue de dessus disparaît pour laisser place à une vue de côté. Cette séparation intrinsèque entre les combats et le reste du jeu nous a obligé à travailler plus (on se retrouve avec l'équivalent de deux jeux en un) avec d'autres problèmes spécifiques qui sont présentés ici.

8.1 HUD

Entre la deuxième et la troisième soutenance, notre interface de combat a bien changé. Aparavant nous n'affichions que la vie en pourcentages. A présent, la vie du héros et du monstre ennemi sont représentées par des barres de vie, contenant elles même la quantité de vie restante de manière numérique. Nous avons également rajouté un menu d'action en bas de la fenêtre pour indiquer au joueur toutes les possibilités qui s'offrent à lui.

8.2 Génération de terrain aléatoire

Les algorithmes de génération aléatoire de terrain en 2D vue de côté ne sont pas extrêmement nombreux sur internet. C'est pourquoi nous avons décidé de recréer cette génération nous même en nous inspirant de techniques utilisées dans la 3D. Nous avons donc pris un terrain plat, puis lui avons appliqué le bruit de Perlin. Celui ci permet en partant d'une structure lisse, d'obtenir diverses hauteurs de terrain mais des hauteurs cohérentes entre elles (on évite ainsi les falaises qui casseraient un peu le combat) Tout ceci se rapproche de la génération de fractales vue en début d'année lors de nos travaux pratiques de Caml en informatique pratique. Après avoir obtenu une courbe de terrain, nous repassons dessus

pour avoir un sol varié. On retrouve ainsi d'une part de la poussière de l'abimn mais aussi de la pierre et d'autres matériaux. Enfin, on repasse une dernière fois sur le terrain pour y ajouter des arbres, plantes, et divers éléments qui ne font pas intrinsèquement partie du sol (un rocher par exemple) On arrive ainsi à un rendu à peu près réaliste de terrain.

8.3 Tile Mapping pixel par pixel

Contrairement à la carte générale, ici il est important d'avoir un terrain détaillé. C'est pourquoi nous avons décidé que nos cellules feraient toutes un pixel seulement. Pour éviter des problèmes de ralentissements comme lors de la première soutenance, nous avons ici décidé que la carte ne serait pas un tableau de valeurs classique mais directement une image. Ceci accélère grandement le processus et permet de toujours avoir un rendu fluide.

8.4 Saut réaliste

En combat, on voit la scène de côté, ce qui permet au personnage de faire des sauts. Le problème de réalisme du saut se pose alors. On ne procèdera pas comme une entité classique dans ce cas donc.
Un saut lié à une courbe linéaire serait trop irréaliste, et à ce moment là plusieurs options s'offrent à nous :

- Implémenter un moteur physique externe pour calculer les sauts
- Calculer les sauts avec des fonctions mathématiques du second degré
- Implémenter la seconde loi de Newton

La première solution a vite été mise de côté lorsque l'on a cherché des moteurs. La plupart sont plus adaptés à des calculs en 3D, et ils sont de toute façon bien trop complets et complexes pour s'en servir seulement pour faire un saut.
Il nous restait alors le choix des deux derniers. La seconde loi de Newton a l'avantage de nous fournir une parabole réaliste sans que l'on ait à chercher l'équation de la dite parabole.
Nous avons donc, bien sûr retenu la troisième solution.

Afin d'y parvenir, il faut utiliser le GameTime donné à la fonction Update de la pierre du combat pour faire un saut en fonction du temps.
On trouvera en utilisant les axes paramétriques les formules suivantes :

$$x_(t) = v_x \times t$$
$$y_(t) = v_y \times t - \frac{gt^2}{2}$$

Ici t représente le temps en millisecondes écoulées depuis le début du saut, g la constante gravitationelle que nous définirons en dessous de 9.81N/kg pour permettre à notre personnage de sauter plus haut (en diminuant la gravité donc)
v représente le vecteur vitesse initial que nous calculerons comme suit :

$$v_x = v_0 \times \cos\theta$$
$$v_y = v_0 \times \sin\theta$$

avec v_0 un vecteur qui représentera la force dans les jambes du personnage et θ l'angle de saut que nous définirons à 60° en mouvement et à 90° à l'arrêt. Nous avons finalement pour cette troisième soutenance décidé d'apporter un peu de liberté au joueur dans son saut. Il pourra ainsi légèrement rectifier sa trajectoire en vol. Malgré l'irréalisme physique dont ceci fait preuve, c'est un apport non négligeable de confort de jeu.

8.5 Attaque à distance

Depuis le début, nous n'avions qu'une attaque à l'épée disponible pour notre personnage. Nous avons donc voulu ajouter une attaque à distance. Pour des raisons de simplicité nous avons simplement implémenté une boule de feu venant d'un sceptre du joueur. Celle ci se déplace sur une ligne droite ce qui nous a évité des calculs physiques encore plus nombreux.

8.6 Destruction progressive du terrain

Pour ajouter du dynamisme au jeu, nous avons décidé de permettre au joueur de détruire le terrain. Il est ainsi possible en lançant une boule

de feu sur un arbre de brûler celui ci, ou encore d'enflammer le sol pour faire des dégâts à l'ennemi lorsqu'il passe dessus. Il est également possible de détruire certaines parties du sol comme la poussière de l'abimn avec du feu.

8.7 Intelligence artificielle

Le terrain étant destructible et généré aléatoirement, il a fallu créer une intelligence artificielle qui s'adapte au terrain. L'ennemi a donc comme stratégie de base de se diriger vers le joueur, en sautant si c'est nécessaire, et en attaquant à distance dès qu'il en a la possibilité (si celui ci en a la capacité)

8.8 Caméra adaptative

Pour une meilleur immersion en jeu, nous avons fait en sorte que la caméra suive le joueur et le monstre du mieux qu'elle peut. On a ainsi un zoom qui se fait si le joueur est proche du monstre et inversement si celui ci en est particulièrement éloigné. La caméra se déplace horizontalement afin de suivre le joueur et l'ennemi le mieux possible. La position et le zoom de la caméra sont recalculés à chaque frame pour une fluidité maximale.

9 Conclusion

La finalite de notre projet porte sur la réalisation d'un jeu. Pour nous, il s'agissait de réussir a développer notre jeu avec nos propres règles, quelque chose dont on pourrait être fiers, pour les autres, d'acquérir le savoir faire nécessaire ainsi qu'un peu d'expérience dans ce domaine. En réalite, c'est bien plus que celà.

Quel que soit le projet, toutes les étapes de realisation sont importantes. Notemment le début : pour bien commencer, il faut avoir de bonnes bases, et ainsi creer une equipe "optimale". Il faut en effet se mettre d'accord sur les objectifs et les aptitudes de chacun, de manière à ne pas se retrouver bloqué par manque de motivation - en cas de désaccords dans le groupe, qui pourraient mener à des situations compliquées - ou de compétences (ou par manque de temps de formation à ces compétences) si le groupe n'est pas assez polyvalent.

Ensuite, nous avons besoin d'avoir une vision globale du projet. Nous savons partiellement ce que nous voulons, mais pas comment le mettre en place. Il faut ainsi se mettre d'accord sur la facon de procéder, déterminer le temps de réalisation afin de placer des priorités, puis répartir les tâches. Une bonne planification est le début d'une bonne gestion d'équipe - c'est, du moins, ce que mon experience m'a appris. J'ai pense que les plus competents dans un domaine pourraient les prendre en charge pour assurer un gain de temps. Cependant, ce n'est pas le plus enrichissant.

9.1 Laure

En tant que chef de projet, le recrutement est une etape delicate et determinante. Pour notre projet, je pense que nous avons bien coordonné nos rôles respectifs et que tout le monde a pu trouver sa place.

Etant la seule a maitriser le pouvoir de Photoshop, j'ai realisé tous les graphismes du jeu et du site, qu'il s'agisse d'un background sobre, des sprites de personnages en pixel art, des boutons de redirection ou d'options. Je savais faire certaines choses, mais je n'ai jamais eu un temps limité. Il est certain que ce projet ne m'a pas laissé le choix. Il en est de même pour la réalisation du site qui a été in extremis alors que je n'avais pas fait grand chose dans ce domaine jusque là.

De plus, je vois en moi une très nette amélioration au niveau du code... Je ne savais vraiment pas grand chose avant d'entrer à Epita. Dans le passé, je codais des choses qui ne bougeaient pas (j'entends par là que je ne recupérais quasiment aucune information de l'utilisateur), je n'avais jamais pour ainsi dire programmé. Maintenant, je maîtrise tout ce qu'on a pu m'apprendre durant les TPs, tout ce qu'on a utilisé pour notre jeu (ce qui se recoupe entre eux, ne serait-ce que vis à vis de XNA) et je n'aurai aucun mal dans le futur à apprendre encore et encore au fur et à mesure de mes expériences.

9.2 Viviane

Pour quelqu'un qui n'avait jamais tapé une ligne de code avant son entrée à EPITA, un projet de jeu aussi complet qu'un RPG était un véritable défi grâce auquel j'ai appris a gérer les bases de la programmation orientée objet. De plus, le choix d'un jeu demande de savoir utiliser une architecture précise pour éviter les bugs et surtout de comprendre des concepts comme l'héritage, l'instanciation ou les types génériques. Des concepts comme ceux là servent dans presque tous les programmes. Celà m'a aussi vite fait comprendre l'importance de la factorisation du code, de la séparation des classes et l'utilité des commentaires. Travaillant à plusieurs, le dialogue entre les développeurs est vite devenu un point aussi important que problématique. Comprendre le code des autres n'est pas toujours aisé et organiser nos temps de travail les uns par rapport aux autres relevait du conseil de guerre. Après une telle expérience, et en considérant que ce genre de problèmes de communication existent toujours au sein d'une équipe, je pense que la plus grande difficulté lors d'un projet est de gérer les relations humaines. Les retards et les erreurs se sont souvent accumulés et lors des premières soutenances, nous avons réussi à boucler notre travail sur le fil du rasoir. Travailler avec d'autres per-

sonnes m'a pourtant permi de voir d'autres angles d'approche, de voir les méthodes de création d'un site web, ou de graphismes, ce que je n'avais jamais vu auparavant.

Au final, et malgré quelques moments pénibles, cette expérience a été extrêmement bénéfique, autant d'un point de vue technique qu'humain. Partager un même projet et créer un univers en collaboration avec d'autres personnes est très gratifiant. Le développement ne demande pas seulement des capacités techniques mais également de l'imagination et de l'organisation. C'est peut-être même les parties plus plus intéressantes de ce métier.

9.3 Alexandre

Pour ma part, je considère ce projet comme une réussite. Certes, nous n'avons pas réussi à atteindre tous nos objectifs techniques, mais nous avons toujours gardé une bonne entente entre les membres du groupe. Tout le monde a travaillé, a touché au code du jeu, a participé à sa création. C'est cette partie qui me satisfait le plus. Je suis extrêmement fier que nous ayons tous pu gagner en expérience à travers ce projet.

J'avais déjà beaucoup programmé avant d'arriver à Epita, mais ceci n'a pas empêché mes collègues de rester dans le coup. Ils se sont vite mis au niveau, ont toujours compris plutôt rapidement des principes assez complexes, et je suis pour cela très content d'avoir été dans ce groupe.

Aujourd'hui, si nous devions refaire ce projet entièrement, nous n'aurions certainement pas besoin d'autant de temps (ce qui prouve bien qu'il y a eu apprentissage).

Je compte bien réutiliser la surcouche développée pour XNA dans le cadre de ce projet. Malheureusement je ne pourrais pas en faire profiter les futures promotions à cause de l'interdiction d'utilisation de cet outil (il n'est plus officiellement supporté par Microsoft) mais je compte tout de même distribuer cette architecture sur internet pour permettre à tout apprenant motivé de créer un jeu vidéo facilement.

Un de mes principaux défauts en informatique a toujours été d'avoir une approche très bas niveau, en somme, d'avoir dix ans de retard. A travers ce projet j'ai pu découvrir toutes sortes de structures de données

et de techniques propre à l'informatique d'aujourd'hui.

Si tout ceci était à refaire, j'accepterais sans hésiter.

Table des figures