

Projet de Langage C – 3EII 2015 – INSA Rennes

# Le Rêve de Jacques

par

Antoine Pazat - Alexis Béch   
Klervi Menoret - Hugo Miomandre



# Sommaire

Sommaire .....	0
Introduction .....	0
1. Cahier des charges .....	1
2. Analyse fonctionnelle.....	2
3. Architecture du logiciel .....	3
4. Interface graphique.....	4
5. En conclusion... ..	5

# Introduction

Le Rêve de Jacques est un projet de jeu de plate-forme en 2D inspiré du jeu Métröïd (figure 1) et fonctionnant sous Windows. Codé en langage C sur les logiciels Eclipse et Visual Studio, le programme s'appuie sur différentes librairies comme SDL\_1.2, SDL\_image, SDL\_ttf ou encore Fmod.

En règle générale, le jeu a été codé en utilisant le langage C comme s'il était un langage orienté objet. Un grand nombre de structures, de types et d'énumérations a été nécessaire à son fonctionnement.

Le but du jeu est simple : l'unique joueur contrôle un personnage afin d'explorer la carte pour en trouver l'issue. A chaque fois que le joueur trouve l'issue, la carte change et le personnage débloquent une nouvelle compétence. Le joueur devra donc revisiter certains lieux une fois qu'il aura acquis une nouvelle compétence. Le jeu se termine lorsque le personnage découvre la dernière porte.

La carte est également parcourue par des personnages non joueurs hostiles, qui peuvent infliger des dommages au personnage contrôlé. Le joueur, pour se défendre, peut tirer des projectiles pour éliminer les ennemis, qui varient selon les compétences acquises au cours du jeu. Si le joueur reçoit trop de dommages, il perd le jeu. Des personnages non joueurs neutres donnent des indications au joueur pour lui indiquer l'endroit où il doit se rendre.

Nos premiers objectifs se concentraient autour de la création d'une plateforme de jeu basique, c'est-à-dire avec une gravité crédible et une fluidité permettant le jeu. Si nous ne sommes pas parvenus à mettre au point autant de cartes et de compétences que prévu, nous avons réussi à développer une première version jouable.

# 1. Cahier des charges

Le cahier des charges est présenté ici tel qu'il a été rédigé en début de projet. L'ensemble des objectifs était sans doute ambitieux, surtout vis-à-vis de notre ignorance concernant les bibliothèques utilisées. Nous avons ensuite revu ce cahier des charges de manière plus raisonnable. Il est cependant intéressant de comparer le rendu final avec ces premières idées.

## A. L'environnement

### **Zone de jeu**

Le monde du Rêve de Jacques est composé de différentes "Salles" qui communiquent par des portes. Les salles sont regroupées par zones. Le nom de la zone sera indiqué, pour que le joueur puisse se situer. Chaque zone possède un thème visuel propre. Chaque salle est constituée de plateformes fixes sur lesquels le joueur peut se déplacer. Les blocs qui forment les plateformes sont fictifs, et permettent de créer l'impression de gravité et de collision.

### **HUD (Heads Up Display)**

La fenêtre en jeu se divise en trois zones. Une barre supérieure indique l'état de santé du personnage joué, le nom de la zone ; une zone est réservée à l'affichage du jeu ; et une barre inférieure permet d'annoncer le texte des dialogues, qui n'apparaîtra que lors de l'interaction avec les PNJ.

## B. Les personnages

### **Jacques, le personnage principal**

Le joueur contrôle Jacques, un personnage unique. Ce personnage est placé au centre de l'écran, et la vue du jeu suit son trajet. Il est capable de marcher, sauter, et tirer des projectiles dès le début du jeu. Il pourra acquérir de nouvelles compétences en évoluant dans le monde.

### **Les « mobs »**

Les mobs sont les personnages mobiles qui ne sont pas contrôlés par le joueur. Ceux-ci sont hostiles. Il y aura 3 types de mobs différents :

- Un mob qui suit un itinéraire fixe, et occasionne des dégâts au personnage principal en le touchant. Son comportement de dépend pas de celui du joueur.
- Un mob qui suit un itinéraire fixe, et tire des projectiles de façon régulière. Le personnage principal reçoit des dégâts s'il touche ce mob ou l'un de ses projectiles. Son comportement ne dépend pas de celui du joueur.
- Un mob qui adapte son déplacement à celui du joueur pour essayer de se maintenir à une distance fixe, et tire des projectiles. Le personnage principal reçoit des dégâts s'il touche ce mob ou l'un de ses projectiles.

### **Les Personnages Non Joueurs (PNJ)**

Les PNJ ne sont pas hostiles. Ils peuvent communiquer au joueur des informations concernant son avancée dans le jeu et ses objectifs.

## C. Mécaniques de jeu

### Le système de dégâts

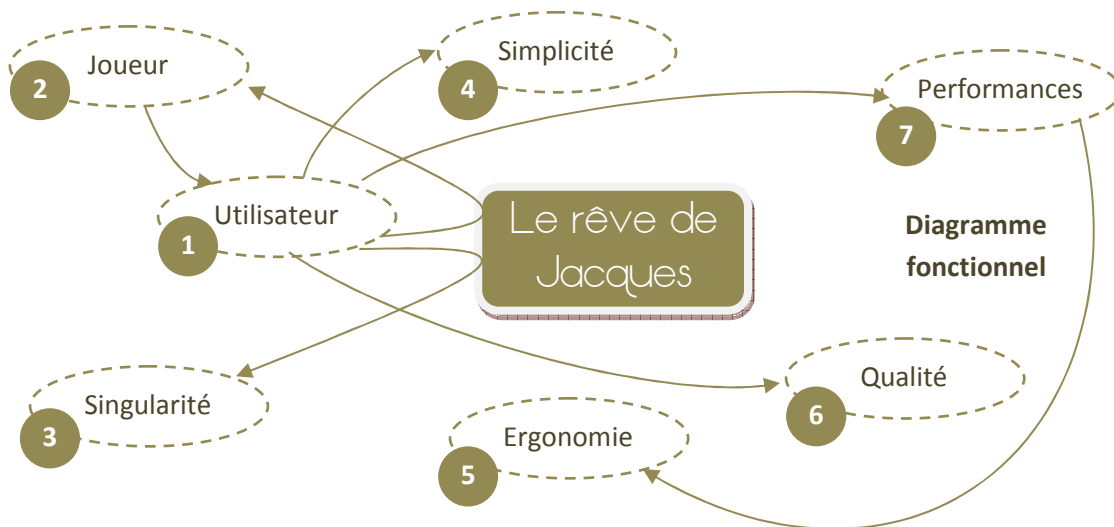
Personnage	Points de vie	Dégâts infligés	Type d'attaque
Joueur	100	4	distance
Mob1	20	10	contact
Mob2	40	15	contact+distance
Mob3	60	30	contact+distance

En début de jeu, le Joueur possède 100 points de vie et peut infliger 4 points de dégâts aux mobs qu'il rencontre. Les mobs ont respectivement 20, 40 et 60 points de vie et infligent 10, 15 et 30 points de dégâts selon leur type.

### Compétences

Des compétences peuvent être acquises par le personnage principal au cours de la partie, comme escalader certains éléments du décor, s'accroupir pour accéder à de nouveaux endroits, sauter plus haut pour atteindre de nouvelles plateformes, ou encore marcher sur l'eau...

## 2. Analyse fonctionnelle



### Liste des fonctionnalités issues du diagramme fonctionnel

F. 1-2	Permettre une facilité de prise en main Permettre une utilisation sur PC Assurer un contrôle par clavier et manette XBox	F. 1-7	Assurer un temps d'interaction homme-machine faible (fluidité) Permettre un lancement du jeu rapide
F. 1-3	Offrir un jeu à l'interface graphique unique	F. 7-5	Obtenir un fichier final de taille raisonnable
F. 1-4	Permettre une compréhension simple du jeu Permettre une installation simple		
F. 1-6	Assurer une qualité audiovisuelle		

### 3. Architecture du logiciel

Afin de simplifier le codage pour l'ensemble du groupe, nous avons réparti chaque point distinct dans une classe spécifique. De cette manière, il est aisé d'intervenir lorsqu'un problème apparaît. Excepté le fichier main.c, chaque fichier source possède un fichier header qui lui est propre. Au total, seize fichiers sources et six structures ont été nécessaires à la production du jeu.

Structures	Contenu
<b>Bouton</b>	Les boutons composent les menus. Un bouton possède en attributs des éléments de type SDL_Surface relatifs aux différentes images de sélections possibles ainsi que deux entiers définissant sa hauteur et sa largeur.
<b>PNJ</b>	Un PNJ reste immobile et lance un dialogue la première fois que le joueur se place en face de lui.
<b>Tile</b>	Une Tile (tuile) est une case comportant des informations sur le terrain qui compose le jeu (elle peut être traversable ou solide, et avoir des effets sur le joueur).
<b>Room</b>	La Room est la salle dans laquelle évolue le joueur. Elle contient des informations sur les tuiles qui la composent, les mobs qui y sont présent, ainsi que sur les portes.
<b>Mob</b>	Mob (personnage mobile) représente le joueur et les autres personnages qui se déplacent

Fichiers .c	Contenu
<b>main.c</b>	Initialisation des bibliothèques et lancement du jeu.
<b>game.c</b>	Initialisation des éléments du jeu (salles, personnages...), lancement de la boucle du programme et traitement du jeu divisé de la façon suivante : <ul style="list-style-type: none"> <li>– Récupération des inputs (clavier, manette) ;</li> <li>– Mise à jour des données du jeu (positions, vie, etc) ;</li> <li>– Affichage des éléments du jeu ;</li> </ul>
<b>level.c</b>	Gestion des niveaux.
<b>menu.c</b>	Génération la fenêtre du menu, gère les boutons en fonction de la sélection, Redirection du joueur vers différentes sections. <ul style="list-style-type: none"> <li>• Structure Bouton</li> </ul>
<b>mob.c</b>	Fonctions relatives à l'initialisation, au déplacement et aux interactions des mobs. <ul style="list-style-type: none"> <li>• Structure Mob</li> </ul>
<b>pnj.c</b>	Fonctions relatives à l'initialisation, au déplacement et aux interactions des PNJ. <ul style="list-style-type: none"> <li>• Structure PNJ</li> </ul>
<b>program.c</b>	Gestion de l'affichage en plein écran du jeu. L'écran est de dimension fixe, et sa position est calculée à partir de la dimension de l'écran de l'utilisateur
<b>projectile.c</b>	Fonctions relatives à l'initialisation et à la gestion des projectiles. <ul style="list-style-type: none"> <li>• Structure Projectile</li> </ul>
<b>room.c</b>	Fonctions relatives à l'initialisation, à la définition et à la mise à jour des instances en considérant des ajouts de portes, de mobs et d'images de fond. <ul style="list-style-type: none"> <li>• Structure Room</li> </ul>
<b>tile.c</b>	Fonctions relatives à la définition de la structure et à la gestion de l'initialisation de ses instances. <ul style="list-style-type: none"> <li>• Structure Tile</li> </ul>
<b>util.c</b>	Fonctions permettant d'initialiser plus rapidement les champs de structures SDL.
<b>animation.c</b>	Gestion des animations à travers des fonctions de lecture d'animations sur une sprite sheet ne fonction permettant de déterminer quand jouer quelle animation
<b>input.c</b>	Récupération des informations envoyées par le clavier ou la manette.
<b>ia.c</b>	Gestion d'une intelligence artificielle. Simulation des inputs pour contrôler les actions des mobs.
<b>scrolling .c</b>	Gestion du mouvement de caméra.
<b>collision.c</b>	Gestion de la physique du jeu (gravité, collision contre les murs, vitesses de déplacement des mobs).

## 4. Interface graphique

### A. L'environnement

La carte ne sera pas affichée à l'écran. Elle contient un tableau répertoriant toutes les salles du jeu. Une salle est modélisée par un système de tuiles. On peut la représenter par un grand quadrillage où chaque case représente un élément du décor, qui peut interagir avec les personnages. Ce sont ces cases qui sont appelées « tuiles ». La salle a aussi une ou plusieurs images de fond, qui représentent l'arrière plan. Globalement, une salle est décrite par plusieurs tableaux qui contiennent des informations sur les tuiles, les images, les mobs, les coordonnées des mobs, les portes et les coordonnées des portes.

Les tuiles sont rectangulaires et de deux tailles différentes. Elles sont représentées par un Sprite, c'est-à-dire une image, qui sera lue sur une sprite sheet. Une tuile ne peut pas changer de propriété pendant le jeu (sauf pour le booléen `firstTime`). Plus généralement, une tuile est définie par un Sprite, des booléens qui définissent si le personnage l'a déjà traversé ou peut ou non la traverser ou l'escalader et des entiers qui représentent les dégâts qu'elle inflige, les compétences qu'elle offre ou son type (vide, solide, échelle, spéciale...).

Ces affichages sont gérés avec la bibliothèque `SDL_1.2`. Afin de pouvoir facilement superposer des images, nous utilisons aussi la bibliothèque `SDL_image` qui traite le PNG. Tous les visuels ont été réalisés en aquarelle et traités avec le logiciel Adobe Photoshop.

### B. Sprite Sheet

Une sprite sheet est une image qui regroupe toutes les sprites d'un élément du jeu. Une sprite est quant à elle une image représentant un élément du jeu. Si cet élément est animé, il y'aura une sprite pour chaque image de l'animation. Dans notre cas, sur une même sprite sheet, toutes les sprites auront la même taille. Les positions seront donc comptées en taille de sprite et non en pixels.

IMAGE SPRITE SHEET



## 5. En conclusion...

Tout au long de notre projet et de notre apprentissage, nous nous sommes heurtés à des problèmes d'organisation, de code ou de gestion. Nous avons dû adapter nos idées premières, modifier nos prévisions initiales, ou même supprimer des objectifs qui ne semblaient plus réalisables. Nous avons fait le choix d'établir une première version fonctionnelle, qui posséderait les caractéristiques que nous avons jugées « primaires ». Après de nombreux essais (notamment de gravité et de collision) sur des cartes réduites avec des formes cubiques, nous sommes parvenus à obtenir des résultats convaincants. Suite à cela, nous avons pu établir un premier menu, une ambiance sonore, une zone de texte, une identité graphique, etc...

Malgré tout le temps passé à chercher des solutions à nos problèmes de code, nous avons appris beaucoup de choses de nos erreurs, en particulier sur :

- La gestion de la mémoire avec la SDL :  
Corrigée par une meilleure utilisation de la fonction `free` de SDL, et une optimisation de la taille de nos fichiers.
- La gestion de la mémoire sur certaines structures :  
Corrigée en prenant le temps d'allouer la mémoire dynamiquement.
- La fluidité du jeu en affichant de grandes images :  
Corrigé en utilisant des fonctions s'adaptant au format de l'écran de la machine.

En plus des solutions trouvées à certains de nos problèmes, nous sommes capables après ces quelques mois de projet d'établir un rapport d'étonnement en lien avec ce travail :

- Un cahier des charges chargé décharge.  
Bien que le temps passé à rédiger notre cahier des charges nous a paru très long, ce dernier s'est avéré être un outil incontournable pour notre travail en groupe. Il nous a permis de bien identifier nos rôles respectifs dans l'élaboration du projet, mais aussi de sans cesse nous remettre en question par rapport à nos objectifs.
- Une documentation vaut mille mots.  
Lire avec attention la documentation des outils et bibliothèques s'avère souvent plus utile que de chercher directement la solution d'une erreur de compilation. De la même façon, choisir ses sources et ses cours peut éviter de longues heures de débogage.
- Un esprit sain dans un fichier source sain.  
Bien organiser son espace de travail est nécessaire. Après quelques jours à travailler dans le fichier `main`, nous avons envisagé de travailler avec un fichier source par élément. Après cela, il fut nettement plus facile de trouver des solutions à nos problèmes...
- Les bons commentaires font les bons amis.  
Lors d'un travail en groupe, et particulièrement lorsque chacun travaille sur un IDE différent, il est important de bien commenter ses fonctions, voire de rédiger un document annexe présentant leurs objectifs.
- Un warning ne vient jamais seul.  
Ne pas se précipiter pour tester son code et prendre le temps de résoudre tous les warning peuvent éviter des accidents fâcheux.
- Il faut se méfier du code qui dort.  
En quelques mois, nous avons plusieurs fois perdu le fil de nos travaux respectifs. En plus de l'importance des annotations, nous avons finalement appris à ne pas laisser des bugs non résolus pendant plusieurs jours, de façon à ne pas nous perdre dans nos lignes de code.