

Le Mans Université
Licence Informatique 2ème année
Dragon Quest Like

Choux Alex, Beranger Anna, Fabre Arthur

5 avril 2022

Table des matières

1	Introduction	3
2	Conception	3
2.1	But du jeu	3
2.2	Analyse	3
2.3	Fonctionnalités	4
3	Organisation du travail	4
3.1	Répartition des tâches	4
3.2	Planning du projet	5
3.3	Outils utilisés	5
4	Développement du jeu	6
4.1	Architecture du projet	6
4.2	Makefile	6
4.3	Conception des ressources	7
4.3.1	Les entités	7
4.3.2	Le monde	8
4.3.3	Les images	8
4.4	Menus	8
4.5	Sauvegarde et chargement	9
4.6	Détection des collisions	10
4.7	Déplacements	11
4.8	Combats	12
4.9	Narration et boutique	13
4.10	Inventaire	13
5	Résultats	15
5.1	Attendus	15
5.2	Obtenus	15
6	Conclusion	16
6.1	Apports du projet	16
6.2	Piste d'améliorations	16
7	Annexes	17
7.1	Débogage	17
7.2	Autres	18

1 Introduction

Dans le cadre du projet du second semestre de la deuxième année de la licence informatique de l'université du Mans, nous devons réaliser un jeu. Ce jeu doit présenter des aspects techniques qui doivent être implémentés à partir des connaissances obtenues lors des trois semestres précédents. Notre jeu est un J-RPG (acronyme pour «japanese role playing game») que nous voulons semblable aux premiers opus de la saga Dragon Quest. C'est un type de jeu dans lequel nous explorons un monde, nous affrontons des monstres, nous équipons des objets et nous gagnons des niveaux.

2 Conception

Avant de commencer à programmer, il faut d'abord matérialiser une idée du jeu puis analyser les besoins techniques afin de réaliser correctement l'implémentation des fonctionnalités désirées.

2.1 But du jeu

Nous suivons l'histoire d'une jeune fille se baladant tranquillement dans la forêt. Tout à coup, un passant l'informe que le grand sorcier a profité de son absence pour attaquer ses amis qui sont restés se reposer à la maison. Inquiète, elle décide de faire marche arrière afin de sauver ses amis. Cependant, ce grand sorcier a semé des monstres sur son passage. Arrivera-t-elle à sauver ses amis ?

2.2 Analyse

Notre jeu, comme indiqué auparavant est un J-RPG, il a donc besoin des nombreuses fonctionnalités suivantes :

- un affichage
- un personnage jouable
- des monstres
- des caractéristiques de base : des points de vie, de la puissance d'attaque et des points de mana
- des marchands
- des coffres
- une histoire
- un inventaire
- un système permettant la simulation de combats se déroulant au tour par tour
- un système permettant l'équipement et le déséquipement d'objets
- un système permettant la sauvegarde et le chargement d'une partie

2.3 Fonctionnalités

Suite à l'analyse des besoins de notre jeu nous nous sommes mis d'accord sur une liste de fonctionnalités que nous souhaitions implémenter :

- le jeu doit démarrer avec l'affichage d'un **menu**, géré au clavier, dans lequel il est possible de charger une **partie sauvegardée** préalablement ou de commencer une nouvelle partie
- il doit être possible de **déplacer le joueur** grâce aux flèches directionnelles du clavier
- les monstres doivent **se déplacer** grâce à un algorithme simple avec un système d'agression qui doit leur permettre de **poursuivre** le joueur
- lorsqu'un joueur et un monstre entrent en collision un **combat** doit être lancé entre ces deux entités
- différents types de monstres doivent exister dont un boss (un monstre bien plus fort que les autres) à la fin du jeu
- la mort du **boss** doit conclure le jeu et elle doit être nécessaire pour le finir
- le joueur doit posséder un **inventaire** dans lequel il peut s'équiper d'une épée et d'un bouclier qui augmenteront ses statistiques
- la présence de marchands qui vendent des objets comme des pièces d'équipement ou des consommables (potion de points de vie ou de points de mana)
- la présence de **coffres** permettant au joueur d'obtenir de l'argent utile pour la **boutique** du marchand
- la présence de personnages non joueurs qui racontent l'histoire grâce à un système de **dialogues**

3 Organisation du travail

Définir les besoins du projet est une étape importante dans la gestion de celui-ci, cependant il est nécessaire suite à cette définition de répartir les tâches entre les différents contributeurs.

3.1 Répartition des tâches

Après l'analyse des fonctionnalités et des ressources nécessaires à la réalisation du jeu, une répartition des tâches est nécessaire. Le tableau ci-dessous 3.1 représente la répartition des tâches à réaliser ainsi que la contribution de chaque contributeur.

<i>Tâches</i>	<i>Contributeurs</i>	Alex	Anna	Arthur
Structures	1/3	1/3	1/3	
Makefile	1	0	0	
Menus début de partie	0	1	0	
Sauvegarde/Chargement	4/5	1/5	0	
Collisions	1	0	0	
Déplacements du monstre	0	0	1	
Combats	1	0	0	
Inventaire	1/5	0	4/5	
Dialogues et coffres	0	1	0	
Création et recherche d'image	0	1	0	
Boutique	0	1	0	
Aide et carte	0	1	0	

Répartition des tâches sous forme de tableau

3.2 Planning du projet

Pour développer notre projet, il nous fallait au minimum un planning à suivre. Nous en avons eu deux : un au début du projet au niveau global (le [Macroplanning]) et un autre (le microplanning) fait chaque semaine afin d'adapter notre travail en fonction du travail réalisé les semaines précédentes. Ces deux plannings sont très utiles pour nous permettre de voir l'évolution de notre travail au fil du temps dans le but d'adapter notre charge de travail.

Le microplanning (voir annexes 7.2) représente sous forme de tableau les tâches que nous avions prévu de suivre (le fichier texte original est disponible dans le dépôt git), cependant nous n'avons pas suivi à la lettre ce microplanning. Le microplanning (voir annexes 7.2) est le tableau qui représente le travail réalisé chaque semaine.

3.3 Outils utilisés

Dans le cadre du développement de notre jeu, différents outils sont utiles. Pour planifier à long termes notre projet, un diagramme de Gantt est intéressant.

Pour programmer à plusieurs sur un même projet, un dépôt git est indispensable car il permet à chaque contributeur de modifier le code en utilisant différentes branches puis en les fusionnant. Un autre avantage du dépôt git

est qu'il garde en mémoire les précédentes versions au cas où un problème de conflit apparaîtrait lors de la fusion des branches.

4 Développement du jeu

Maintenant que les outils communs de travail sont définis et que les tâches analysées sont réparties, il est temps de s'attaquer au développement du projet.

4.1 Architecture du projet

Nous avons divisé notre projet en plusieurs modules :

- **SDL2_fonctions** regroupe l'ensemble des fonctions utilisant la bibliothèque SDL2
- **entite** comprend les structures relatives aux entités ainsi que leurs primitives.
- **monde** comprend les structures relatives au monde ainsi qu'à ses primitives
- **menu** contient toutes les fonctions liées à la gestion des menus
- **images** contient tout ce qui est relatif à la gestion des textures SDL
- **déplacements** est composé des fonctions de détection des collisions ainsi que des fonctions de déplacements (du joueur et du monstre)
- **affichage** contient tout ce qui est relatif à l'affichage
- **interactions** comprend les fonctions qui gèrent les interactions du joueur avec les autres personnages, les monstres et les coffres
- **jeu** regroupe les fonctions des autres modules dans des fonctions d'initialisation du jeu, de mise à jour des données et de mise à jour graphique
- **main** est le programme principal avec la boucle de jeu

Dans les annexes, se trouve le schéma des dépendances entre les différents modules (figure 1).

4.2 Makefile

Grâce à l'architecure de notre projet, il est désormais possible de réaliser un makefile. Le makefile est très utile afin de compiler les fichiers «sources» en fichiers «objets» puis en un fichier exécutable qui nous permet de jouer au jeu.

Le makefile est constitué de raccourcis ce qui permet de créer les fichiers dans les dossiers adaptés, par exemple les fichiers «objets» dans notre dossier «obj» ou bien le fichier exécutable dans le dossier des fichiers exécutables. L'avantage de cette façon de faire est qu'on ne se perd pas dans une masse de fichiers dans un unique répertoire. C'est un gain de temps et d'énergie. Le makefile possède aussi une option clean ce qui nous permet de directement supprimer tous les fichiers «objets» et exécutables créés lors d'une compilation précédente afin de rendre l'architecture du projet plus lisible. C'est un avantage notable lorsqu'on ajoute nos fichiers à la base commune du dépôt git. En effet, si les fichiers «objets» et exécutables sont ré-écrits il nous suffit de nettoyer notre dépôt personnel afin d'importer les modifications des autres personnes.

C'est un fichier dont l'utilisation est **fortement recommandée** afin qu'un projet se déroule correctement.

4.3 Conception des ressources

Avant de commencer le développement des fonctionnalités il nous faut définir les bases du projet.

4.3.1 Les entités

Pour notre jeu nous avons besoin de représenter le joueur, les monstres, les objets, les coffres et les personnages non joueur (deux types : les marchands et les personnages qui racontent l'histoire). Nous décidons de séparer en trois catégories ces entités : les entités combattantes, non combattantes et les objets.

Il y a deux types d'entités combattantes différentes : le joueur et les monstres. Ils ont des statistiques communes : les points de vie, la puissance d'attaque, la vitesse et les coordonnées nécessaires à leur affichage. Nous faisons donc le choix de créer une structure commune nommée «combattant» qui est utile pour regrouper ces informations. Cependant, le joueur et les monstres possèdent chacun des statistiques différentes : le joueur possède du mana (le monstre non) et le monstre possède un indicateur d'agressivité (le joueur non). C'est pour cela que nous faisons le choix de séparer en deux structures différentes le joueur (structure «joueur») et les monstres (structure «monstre») toutes les deux ayant en commun la sous-structure «combattant».

Les entités non combattantes correspondent aux personnages non joueurs et aux coffres. Un personnage non joueur peut-être un marchand ou un narrateur. La structure «non combattant» regroupe notamment les coordonnées

de l'entitée. Ces coordonnées sont nécessaires pour afficher les images, c'est pourquoi ces trois entités partagent la même structure.

Les objets peuvent augmenter certaines statistiques du joueur s'ils sont utilisés : les points de vie, les points de mana et l'attaque. Il est donc nécessaire de créer une structure «objet» regroupant toutes ses statistiques supplémentaires.

4.3.2 Le monde

Pour notre jeu nous avons besoin de faire évoluer notre joueur dans un monde. Nous faisons donc le choix de séparer celui-ci en plusieurs zones chacunes comprenant plusieurs salles. Il est donc nécessaire de définir trois structures : une structure «monde», une structure «zone» et une structure «salle».

La structure «monde» contient le joueur ainsi que les zones dans lesquelles celui-ci peut se déplacer. De plus, elle possède des variables permettant de retenir le choix de la partie et le choix des options sélectionnées à l'intérieur des menus. Afin de pouvoir traiter les différents états possibles du jeu (déroulement d'un combat, utilisation de l'inventaire...) il est nécessaire de lui ajouter une autre variable représentant l'état du jeu en cours.

La structure «zone» possède un tableau de salles. Chaque zone possède un numéro différent qui permet de les identifier. Enfin la structure «salle» comprend nécessairement un coffre et un monstre ainsi qu'un numéro permettant son identification. Elle peut aussi posséder des personnages non joueurs comme un marchand ou un narrateur. Les numéros de zone et de salle permettent d'afficher les graphismes correspondants.

4.3.3 Les images

L'affichage des images est quelque chose d'obligatoire dans le développement d'un jeu. Nous avons fait le choix de créer nous-mêmes nos images afin d'apporter **notre direction artistique**. De ce fait, nous ne dépendons pas de dimensions imposées par des images déjà existantes. Pour faciliter et rendre plus lisible notre code, nous avons fait le choix de définir une structure «images» qui contient toutes les textures SDL nécessaires à l'affichage de notre jeu.

4.4 Menus

Au lancement du jeu deux menus sont disponibles. Ils sont présents afin de créer ou de charger une partie. Le premier possède deux options : «Jouer»

et «Quitter». Pour choisir l'option, notre variable monde (qui représente le monde du jeu) possède deux entiers nommés respectivement «option» (pour les menus principaux) et «option2» (pour les sous-menus). Pour faire un choix l'utilisateur doit sélectionner une option. Pour changer l'option sélectionnée il doit utiliser les flèches directionnelles du clavier, ce qui actualise la valeur actuelle de l'entier «option» (ou «option2» si le menu est un sous-menu) introduit précédemment. Pour valider l'option sélectionnée il doit enfonce la touche «entrée». S'il sélectionne l'option «Quitter» alors on quitte le jeu. Au contraire s'il choisit l'option «Jouer» un second menu s'affiche. Ce second menu propose de nouvelles options : «Partie 1», «Partie 2», «Nouvelle partie» et «Retour». En validant l'option «Retour», le joueur revient en arrière et affiche le menu précédent. En validant l'option «Partie 1» ou «Partie 2» l'utilisateur charge respectivement la première partie ou la seconde partie (parties préalablement sauvegardées). En validant l'option «Nouvelle partie» l'utilisateur affiche un nouveau sous-menu lui proposant d'écraser la sauvegarde de la première partie ou de la seconde partie. En choisissant d'écraser une des deux parties il commence une nouvelle partie en sauvegardant celle-ci à la place de la partie précédente. Le joueur peut se rétracter s'il ne souhaite pas écraser une partie précédemment sauvegardée. L'affichage des menus est disponible en annexe : figure 3, 4 et 5.

D'autres situations différentes du lancement du jeu nécessitent l'utilisation d'un menu : combattre un monstre, naviguer dans la boutique des marchands, utiliser l'inventaire. Les commandes utiles aux jeux sont rappelées dans un menu d'aide.

Il est important de noter qu'il est possible de **sortir du jeu** à tout instant en appuyant sur la **touche «echap»** du clavier.

4.5 Sauvegarde et chargement

Le chargement et la sauvegarde d'une partie se base sur l'utilisation de fichiers texte. Notre dépôt git est pourvu de trois fichiers texte : le fichier «init.txt» utilisé lors de l'initialisation d'une nouvelle partie, les fichiers texte «partie1.txt» et «partie2.txt» qui servent au chargement et à la sauvegarde de la première et de la seconde partie de l'utilisateur. Lors du chargement d'une partie précédemment sauvegardée ainsi que lors de la création d'une nouvelle partie, le fichier texte est écrasé par une nouvelle sauvegarde au début de la partie pour éviter des problèmes de chargement plus tard. Les sauvegardes sont automatiques lors d'un changement de zone. En effet, lorsque le personnage change de zone on sauvegarde les informations importantes dans le fichier texte de la sauvegarde. Pour se souvenir du fichier dans lequel la sauvegarde doit être réalisée, notre variable monde (introduite précédemment) contient la trace du choix de la partie sous forme d'entier. Cet entier est utilisé sous forme de condition afin de sauvegarder les informations sur

le joueur dans le bon fichier.

Le processus de sauvegarde écrit dans les fichiers texte uniquement les informations nécessaires afin que le chargement ultérieur soit réalisé sans problème. Nous sauvegardons le nombre d'objets différents présents dans l'inventaire, leurs indices dans le tableau de l'inventaire ainsi que le nombre d'objets possédés pour chaque objet. Nous sauvegardons aussi des informations sur les pièces d'équipements équipées lors de la sauvegarde par le joueur. Nous enregistrons d'abord le nombre d'objets équipés pour ensuite sauvegarder l'indice de la pièce d'équipement dans le tableau (l'indice zéro est utilisé pour équiper les épées et l'indice un est utilisé pour équiper les boucliers) et l'**«id»** de la pièce (entier respectivement différent pour chaque objet qui permet de les différencier). Le fichier de sauvegarde contient aussi le niveau du joueur, le numéro de la zone où il se trouve au moment de la sauvegarde, ses points de vie maximum et actuels, ses points de mana maximum et actuels, le nombre de ses pièces d'or, son attaque et sa puissance magique.

Le processus de chargement d'une partie utilise la lecture d'un fichier texte préalablement sauvegardé dans le cadre d'un chargement. Au contraire, si le joueur décide de créer une nouvelle partie celle-ci est initialisée à partir d'un fichier texte d'initialisation (le fichier «**init.txt**»). Pour charger ou créer une partie, le processus est le même. Nous commençons par regarder si des objets sont présents dans l'inventaire. Si c'est le cas nous ajoutons ces objets à l'inventaire du joueur grâce à l'indice et au nombre de possessions de cet objet. Nous regardons ensuite si des objets étaient équipés lors de la sauvegarde. Si c'est le cas alors on ajoute ces objets à son équipement. Ensuite, étant donné que chaque entier représente une statistique unique du joueur, nous affectons les entiers sauvegardés aux statistiques du joueur dans le bon ordre pour garder une cohérence.

4.6 Détection des collisions

Dans le cadre de notre jeu la détection des collisions est obligatoire afin que celui-ci puisse être joué sans problème. Il faut que celle-ci prenne en compte différents éléments :

- les barrières de l'environnement
- les monstres
- les personnages non joueurs
- les coffres

La détection des collisions se base sur un système de calcul reposant sur quatre variables :

- la coordonée latérale x à l'origine de l'image (en haut à gauche de l'image)
- la coordonée horizontale y à l'origine de l'image (en haut à gauche de l'image)
- la largeur w de l'image
- l'hauteur h de l'image

Pour vérifier les collisions quatre vérifications s'imposent :

- le point en haut à gauche d'une image ne doit pas être à l'intérieur d'une autre image
- le point en haut à droite d'une image ne doit pas être à l'intérieur d'une autre image
- le point en bas à gauche d'une image ne doit pas être à l'intérieur d'une autre image
- le point en bas à droite d'une image ne doit pas être à l'intérieur d'une autre image

Nous calculons les coordonées de chaque point de cette façon :

- le point en haut à gauche a pour coordonnées le couple ($x;y$)
- le point en haut à droite a pour coordonnées le couple ($x+w;y$)
- le point en bas à gauche a pour coordonnées le couple ($x;y+h$)
- le point en bas à droite a pour coordonnées le couple ($x+w;y+h$)

Si une collision est vérifiée des actions peuvent-être entraînées par celle-ci :

- annuler un déplacement
- lancer un combat
- changer de salle ou de zone

4.7 Déplacements

Nous avons deux types de déplacements :

- les déplacements simples
- les déplacements complexes

Les déplacements simples correspondent seulement à un déplacement d'une unité dans une des quatre directions : droite, gauche, haut, bas.

Les déplacements complexes sont ceux qu'utilisent le joueur et le monstre. Avant de déplacer le joueur on vérifie qu'il n'entre pas en collision avec l'environnement, un personnage non joueur ou un coffre. S'il entre en collision le déplacement n'est pas réalisé. Pour autant si le joueur entre en collision avec un monstre on déclenche un combat.

Le déplacement du monstre est plus complexe. Son déplacement prend en compte plusieurs éléments :

- la distance entre le monstre et le joueur
- un état représentant son caractère agressif : soit il rôde, soit il pourchasse le joueur
- une indication de temporalité

Pour déterminer son état agressif, c'est-à-dire s'il se déplace aléatoirement dans la salle ou s'il pourchasse le joueur, on regarde la distance entre lui et le joueur. En dessous d'une certaine distance, le monstre devient agressif et pourchasse le joueur. Pour autant, si la distance qui sépare le monstre et le joueur devient suffisamment grande le monstre se met à rôder. Cet état est utile afin de déterminer le comportement lors du déplacement du monstre. Cependant, pour gérer la vitesse du déplacement du monstre il est nécessaire de passer par un indicateur temporel. Cet indicateur temporel sert de condition pour déterminer si le monstre se déplace ou si au contraire il reste immobile.

Il est important de noter que le monstre ne se déplace pas si celui-ci est mort lors d'un précédent combat.

4.8 Combats

Une des principales fonctionnalités des jeux «RPG» est celle du combat. Notre jeu ne déroge pas à cette règle : il possède un système de combats qui prend en compte le joueur et le monstre engagé ainsi que leurs statistiques respectives notamment celles d'attaque, de points de vie et (pour le joueur) les points de mana.

Nos combats se déroulent au tour par tour, le joueur a le choix au début de réaliser une action ou de fuir. S'il décide de fuir alors le combat est terminé et le joueur s'échappe. Au contraire, s'il décide d'attaquer il a le choix entre attaquer simplement à l'épée, lancer un sort ou bien utiliser un consommable comme une potion de régénération de points de vie ou de points de mana.

Pour permettre l'affichage des éléments du combat, comme par exemple les points de vie du joueur ou les possibilités que le joueur possède en combat, une fonction a été créée affichant tous les éléments nécessaires à la bonne

compréhension et au bon déroulement du combat. En plus de celle-ci, une fonction permettant la saisie au clavier des actions est implémentée dans le jeu ce qui permet que le jeu soit complètement jouable au clavier. L'affichage des menus des combats est disponible en annexe : figure 6, 7 et 8.

La gestion des combats possède aussi deux conditions d'arrêt soit le joueur meurt (ses points de vie tombent à zéro), soit le monstre meurt (même principe). Si le monstre meurt le joueur gagne de l'or et peut continuer à explorer le monde. Au contraire, si le joueur meurt «Perdu» est affiché et le joueur retourne au dernier point de sauvegarde (figure 9 disponible en annexe). Étant donné que la fonction de sauvegarde enregistre toutes les informations nécessaires au bon chargement d'une partie **à chaque début de zone**, il est important de préciser que si le joueur meurt durant l'exploration de la zone il doit **recommencer l'exploration à zéro**.

4.9 Narration et boutique

La narration et les boutiques sont disponibles via des interactions avec des personnages non joueurs. Dans les deux cas il est nécessaire d'être suffisamment proche du personnage non joueur afin de pouvoir interagir avec lui. Une indication («Appuyer sur p pour parler») s'affiche lorsque la distance entre les deux personnages est suffisamment petite.

Dans le cas d'un narrateur, en enfonçant la touche «p» du clavier une partie de l'histoire est affichée à l'intérieur d'une barre blanche en bas de l'écran (figure 11 disponible en annexe). Pour enlever l'affichage de l'histoire et continuer le jeu il suffit d'enfonçant la touche «entrée».

Dans le cas d'un marchand, en enfonçant la touche «p» du clavier une boutique s'ouvre. Dans cette boutique il est possible d'acheter des pièces d'équipement ou des consommbales (figure 12 disponible en annexe). Les consommables peuvent-être achetés plusieurs fois ce qui n'est pas le cas des pièces d'équipement (on ne peut les acheter qu'une seule fois). L'or possédé par le joueur est affiché en continu ce qui lui facilite la gestion de ses ressources financières. Il est important de noter que la boutique est **évolutive** ce qui signifie que son contenu évolue en fonction de la progression.

4.10 Inventaire

L'inventaire est une des fonctionnalités les plus importantes d'un J-RPG, il permet d'utiliser des consommables ou d'équiper des pièces d'équipements par exemple. Il semble donc nécessaire de rendre son accès le plus facile possible. La touche «i» du clavier semble être la plus intuitive pour accéder à l'inventaire. C'est pourquoi pour accéder à l'inventaire il faut appuyer sur

cette touche. Il est important de noter que la sortie de l'inventaire se fait elle aussi en appuyant sur la touche «i» du clavier. Il est important de noter que les sélections des objets de l'inventaire sont rendues possibles grâce à la variable option précédemment introduite dans la sous-section «Menu».

L'inventaire possède différentes fonctionnalités :

- parcourir les cases de l'inventaire
- deux types d'équipements équipables différents : les boucliers et les épées
- l'équipement et le déséquipement des pièces d'équipement
- l'actualisation des statistiques du joueur lors d'un équipement et d'un déséquipement
- l'utilisation de consommables : potion de points de vie ou de points de mana
- affichage des équipements dans l'inventaire du joueur
- affichage en surbrillance des équipements équipés par le joueur

L'inventaire est divisé en six cases et 3 colonnes :

- on affiche les consommables dans la colonne de gauche
- on affiche les équipements en pierre (épée et bouclier) dans la colonne au centre
- on affiche les équipements en diamant (épée et bouclier) dans la colonne de droite

Chaque case est affectée à un seul et unique objet (consommable et pièce d'équipement). Si on ne possède pas une pièce d'équipement, celle-ci n'est pas affichée dans l'inventaire. Au contraire, les consommables sont toujours affichés ainsi que leur nombre.

Il est possible de se déplacer de case en case via les touches directionnelles du clavier. Pour équiper une pièce d'équipements, il faut **la posséder** (sinon on ne peut pas l'équiper), être sur la case de la pièce et appuyer sur la touche «entrée». La case de la pièce équipée va alors s'afficher en surbrillance et les statistiques du joueur vont être mises à jour en prenant en considération les statistiques de la pièce d'équipement. Pour déséquiper une pièce il faut réitérer le même processus que pour l'équiper. Les statistiques du joueur sont aussi mises à jour et la case n'est plus en surbrillance.

Pour consommer un consommable (potion de points de vie ou de points de mana), il faut que le curseur de l'inventaire soit sur la case d'un consommable et que la touche «entrée» soit pressée. Le joueur regagne des points de vie (ou des points de mana) équivalents à un certain montant. Il ne peut pas

regagner plus de points de vie (ou de points de mana) qu'il ne lui en manque. Si il a tout ses points de vie (ou ses points de mana) il ne peut pas consommer le consommable. Lors d'une utilisation d'un consommable le nombre restant de ce consommable est décrémenté de un. Le joueur ne peut évidemment pas utiliser un consommable s'il n'en possède pas.

5 Résultats

Suite à l'explication des différentes fonctionnalités implémentées dans notre jeu, il est temps de faire bilan afin de prendre du recul sur ce qui a été réalisé.

5.1 Attendus

Lors de l'analyse conceptuelle au début du projet nous avons identifié une liste de fonctionnalités voulues et attendues :

- une carte qui s'actualise lorsqu'on change de salles
- une histoire inédite
- un système de combats
- différents monstres qui bougent de manière interactive
- un personnage qui bouge grâce aux touches du clavier
- un système d'inventaire
- un système de sauvegarde et de chargement
- des animations lors des déplacements
- des animations lors des combats

5.2 Obtenus

Toutes les fonctionnalités souhaitées au début du projet sont implémentées le résultat est donc très satisfaisant de notre point de vue. Pour autant, durant le développement du projet nous avons voulu développer de nouvelles fonctionnalités comme par exemple les animations, de la musique et des effets sonores. Celles-ci ne sont pas malheureusement implémentées dans notre projet.

En ayant commencé le projet avec une meilleure expérience concernant la gestion d'un projet, nous aurions pu faire bien plus de choses notamment implémenter ces trois fonctionnalités par exemple.

6 Conclusion

Nous arrivons au bout du dossier, il est temps pour nous de dégager ce que le projet nous a apporté humainement, techniquement mais aussi intellectuellement.

6.1 Apports du projet

Ce projet de groupe permet de faire évoluer notre vision du travail en groupe, notre capacité à écouter et à s'exprimer, notre capacité à respecter des délais et notre capacité à résoudre des problèmes techniques seuls.

De notre point de vue le travail en groupe n'est pas une chose aisée car elle nécessite une très bonne communication ce qui n'est pas toujours présent (ou suffisamment présent) dans un groupe. Effectivement, la communication est la clé (avec le respect des délais) pour que le projet avance dans une dynamique commune et que les contributeurs ne s'éparpillent pas dans différentes directions.

En conclusion, notre jeu est loin d'être parfait par ses graphismes (limitation de la qualité des graphismes via l'utilisation des fonctions d'affichage graphique de la librairie sdl2) ou encore par le fait que certaines fonctionnalités de notre jeu pourraient être perfectionnées. Cependant nous sommes très contents du résultat produit, car le jeu possède toutes les fonctionnalités que nous voulions lui implémenter.

6.2 Piste d'améliorations

Bien que nous soyons plus que satisfaits du résultat de notre jeu nous avons énormément d'idées de nouvelles fonctionnalités qui renforceraient la qualité de celui-ci :

- développement approfondi des fonctionnalités déjà existantes
- implémentation de la musique ainsi que d'effets sonores
- implémentation des animations
- développement d'un système de points de compétences acquis en montant de niveau qui permettraient de développer des compétences spéciales
- ajout d'un bestiaire permettant de consulter les fiches des monstres défaites

7 Annexes

7.1 Débogage

Contexte Lors du lancement de notre jeu il doit être possible de le quitter à tout moment. L'affichage du jeu et la prise en compte des événements se répètent dans une boucle dans le main. Pour sortir de cette boucle il faut soit : appuyer sur la touche «echap», soit fermer la fenêtre en cliquant sur la croix de la fenêtre.

Problème, en appuyant sur la touche «echap» le jeu ne se ferme pas. On suppose donc que la condition de sortie de la boucle n'est pas vérifiée lorsqu'on presse la touche «echap». La condition d'arrêt du jeu correspond à un entier qui prend la valeur «-1». Il faut donc regarder quelle valeur prend la variable pour savoir si on actualise bien sa valeur lorsqu'on enfonce la touche «echap».

GDB Notre réalisation du débogage de ce problème est illustré par la figure 2. On commence par poser deux points d'arrêt afin de regarder la valeur dde la variable que nous sommes censés actualiser. Le premier point d'arrêt se situe après avoir appuyé sur la touche «echap» et actualisé la valeur de la variable : «break src/jeu.c :928». Le second point d'arrêt se situe au début du main pour regarder la valeur à l'initialisation : «break main».

Au début du «run» on remarque en faisant «print monde->etat_jeu» que la valeur est initialisée à 0 (monde->etat_jeu est la variable qui sert notamment de condition d'arrêt, donc la variable que l'on doit actualiser). On continue l'exécution du débogage jusqu'au point d'arrêt suivant. Théoriquement, la valeur de monde->etat_jeu doit être «-1». Cependant la variable vaut la valeur 0. Il y a donc un problème d'actualisation de la valeur. On ajoute donc la ligne «monde->etat_jeu = -1» à la ligne 928 du fichier «src/jeu.c». En relançant l'exécution du programme on remarque qu'il est possible de quitter le jeu en enfonçant la touche «echap». Le problème est donc **résolu**.

7.2 Autres

<i>Contributeurs</i> <i>Semaine</i>	Alex	Anna	Arthur	Tout le monde
24 janvier - 30 janvier	Fonctions des fichiers «monde.h» et «monde.c»	Fonctions des fichiers «monde.h» et «monde.c»	Fonctions des fichiers «joueur.h» et «joueur.c»	Définition des structures
31 janvier - 6 février	Création du makefile + Aide Arthur et Anna	Céation et destruction du monde, des zones et des salles	Création et initialisation joueur et combattant	
7 février - 13 février	Définition de la librairie «jeu.h»	Initialisation de la carte + Recherche d'images	Fin des fonctions sur les entités	Prototype graphique du jeu
14 février - 20 février	Écriture de l'histoire + Création du fichier interaction	Gestion des collisions/interactions	Menu affichage dans le terminal	Prévisions du microplanning
21 février - 27 février	Gestion des combats	Initialisation du jeu via un fichier + installation de TTF	Gestion de l'inventaire du joueur	Prévisions du microplanning
28 février - 6 mars	Gestion des combats et des collisions	Menu de base et plein écran	Déplacement du monstre	Prévisions du microplanning
7 mars - 13 mars	Gestion des collisions	Initialisation du jeu et sauvegarde à partir de fichiers texte	Déplacements du monstre	
14 mars - 20 mars	Gestion des collisions + changement de zones et salles + réorganisation des fichiers	Initialisation/sauvegarde à partir de fichiers textes, affichage menu choix de partie et affichage de la salle	Détection du joueur par le monstre	
21 mars - 27 mars	Combats + Correction des fuites de mémoire	Placement et interactions avec entités non combattantes	Inventaire	

Microplanning attendu sous la forme d'un tableau

<i>Contributeurs</i> <i>Semaine</i>	Alex	Anna	Arthur	Tout le monde
24 janvier - 30 janvier	Fonctions des fichiers «monde.h» et «monde.c»	Fonctions des fichiers «monde.h» et «monde.c»	Fonctions des fichiers «joueur.h» et «joueur.c»	Définition des structures
31 janvier - 6 février	Création du makefile + Aide Arthur et Anna	Céation et destruction du monde, des zones et des salles	Création et initialisation joueur et combattant	
7 février - 13 février	Définition de la librairie «jeu.h»	Initialisation de la carte + Recherche d'images	Fin des fonctions sur les entités	Prototype graphique du jeu
14 février - 20 février	Écriture de l'histoire + gestion des collisions	Affichage sdl	Déplacement monstre	Prévisions du microplanning
21 février - 27 février	Collisions	Affichage sdl	Déplacement monstre	Prévisions du microplanning
28 février - 6 mars	Collisions	Affichage sdl	Déplacement monstre	
7 mars - 13 mars	Collisions	Affichage sdl	Déplacements monstre + Inventaire	
14 mars - 20 mars	Combats	Création des images	Inventaire	
21 mars - 27 mars	Combats + Correction des fuites de mémoire	Création des images	Inventaire	
28 mars - 1 avril	Combats + Sauvegarde	Dialogues	Inventaire	

Microplanning réel sous la forme d'un tableau

Références

[Macroplanning] https://docs.google.com/spreadsheets/d/1ef4WqBX_j6s_ss7xId5KBBCj4vFqk0Iw-Li9mVlAPKU/edit#gid=1115838130

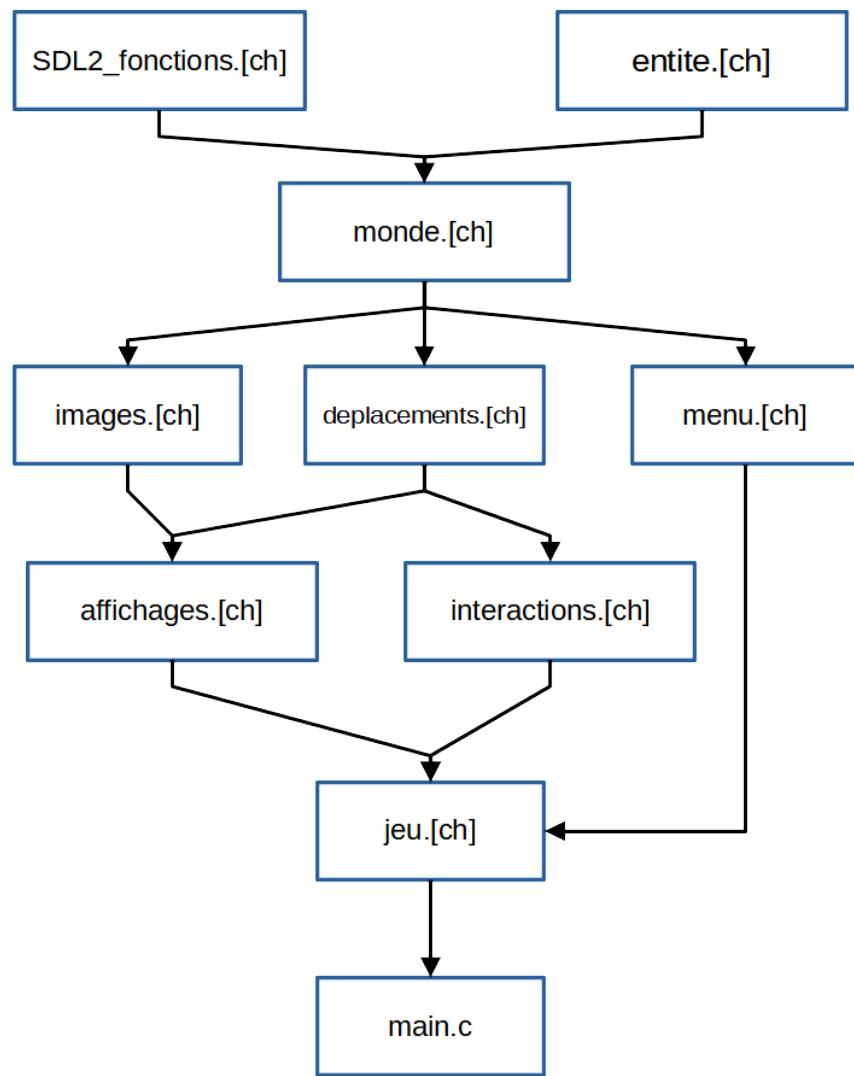


FIGURE 1 – Architecture de notre projet

```

(gdb) break src/jeu.c:928
Breakpoint 1 at 0x8653: file ./src/jeu.c, line 928.
(gdb) run
Starting program: /info/etu/l2info/s196370/DragonQuestLike/bin/DragonQuest
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
id :1
Nom obj: Epee en pierre

[New Thread 0x7fffee90b700 (LWP 3085)]
[New Thread 0x7ffffe10a700 (LWP 3086)]
[New Thread 0x7ffffd909700 (LWP 3087)]
[New Thread 0x7ffffd108700 (LWP 3088)]
print monde->etat_jeu
quit
^Z
Thread 1 "DragonQuest" received signal SIGTSTP, Stopped (user).
0x00007ffff79fe50b in ioctl () at ../sysdeps/unix/syscall-template.S:78
78     ..../sysdeps/unix/syscall-template.S: Aucun fichier ou dossier de ce type.
(gdb) break main:57
Breakpoint 2 at 0x5555555566e9: file ./src/main.c, line 15.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /info/etu/l2info/s196370/DragonQuestLike/bin/DragonQuest
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 2, main () at ./src/main.c:15
15     int main(){
(gdb) print monde->etat_jeu
$1 = 0
(gdb) next
22     TTF_Font * police = NULL;
(gdb) continue
Continuing.
id :1
Nom obj: Epee en pierre

[New Thread 0x7fffee90b700 (LWP 3166)]
[New Thread 0x7ffffe10a700 (LWP 3167)]
[New Thread 0x7fffee10a700 (LWP 3168)]
[New Thread 0x7ffffd909700 (LWP 3169)]

Thread 1 "DragonQuest" hit Breakpoint 1, evenements (event=0xfffffffffe3b0, monde=0x55555555663f0)
at ./src/jeu.c:928
928         printf("fin du jeu");
(gdb) print monde->etat_jeu
$2 = 0

```

FIGURE 2 – Exemple de débogage



FIGURE 3 – Premier menu



FIGURE 4 – Second menu

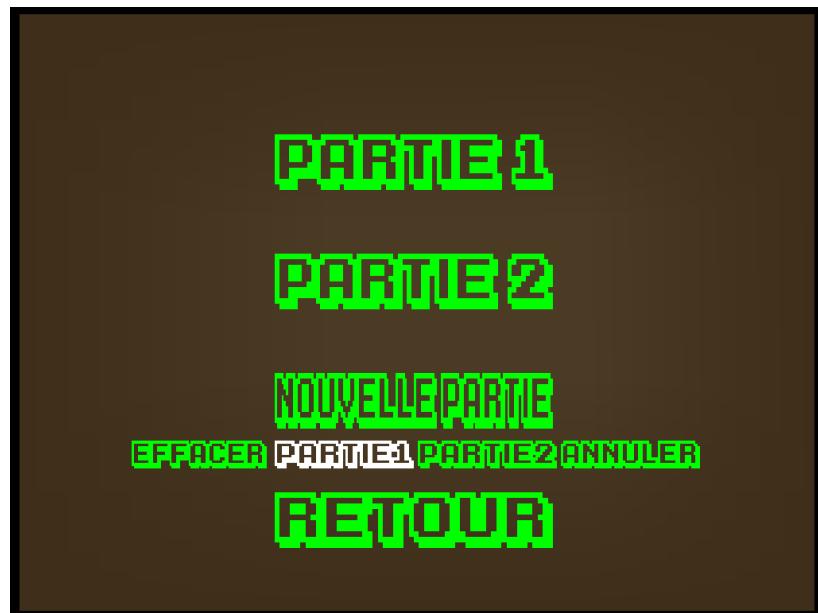


FIGURE 5 – Sous-menu des nouvelles parties

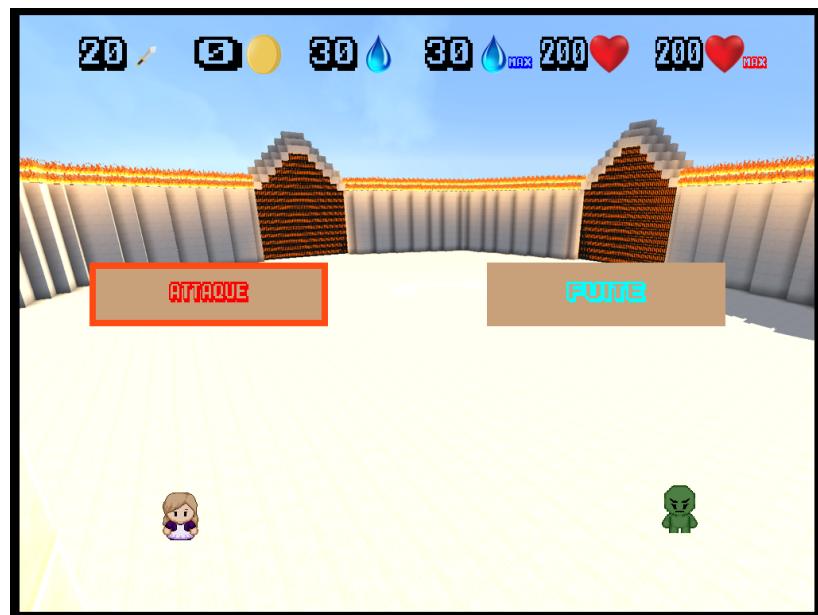


FIGURE 6 – Premier menu lors d'un combat

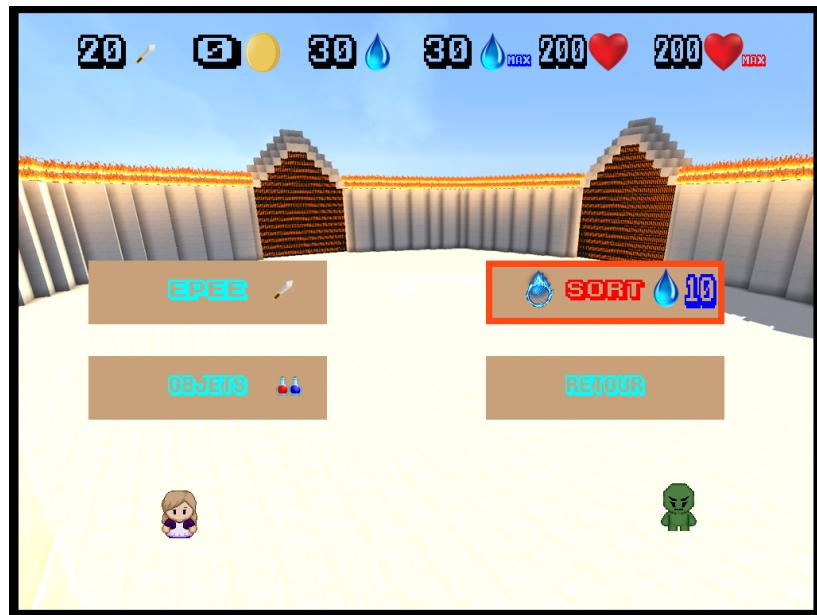


FIGURE 7 – Deuxième menu lors d'un combat

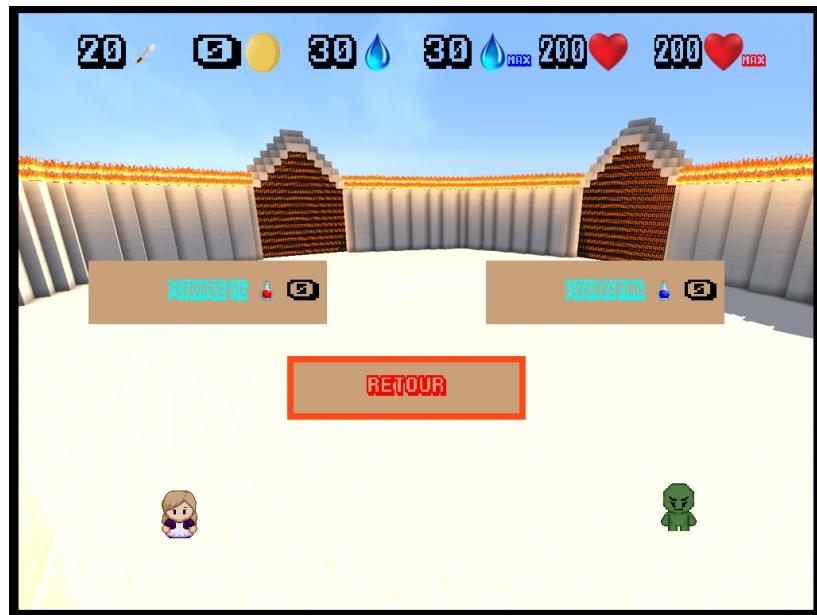


FIGURE 8 – Troisième menu lors d'un combat



FIGURE 9 – Écran de défaite



FIGURE 10 – Victoire à la fin du jeu

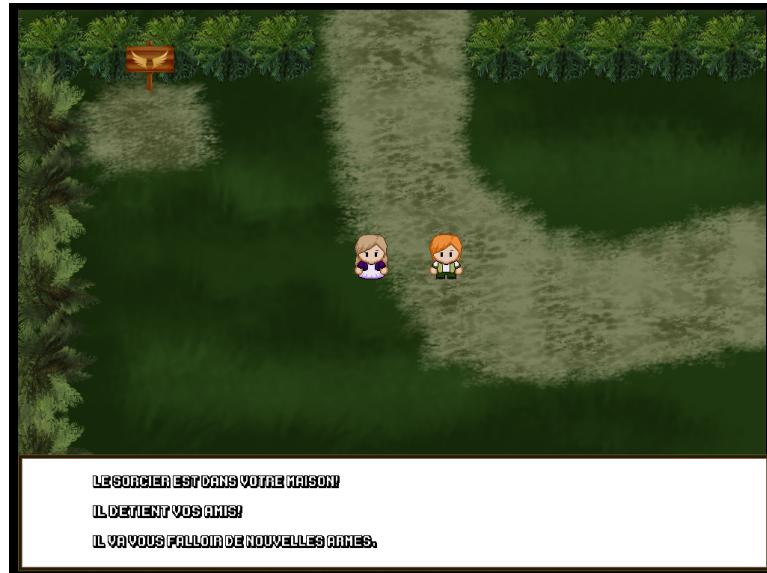


FIGURE 11 – Exemple de narration



FIGURE 12 – Boutique



FIGURE 13 – Inventaire sans équipement



FIGURE 14 – Inventaire avec équipement



FIGURE 15 – Exemple de carte d'une zone

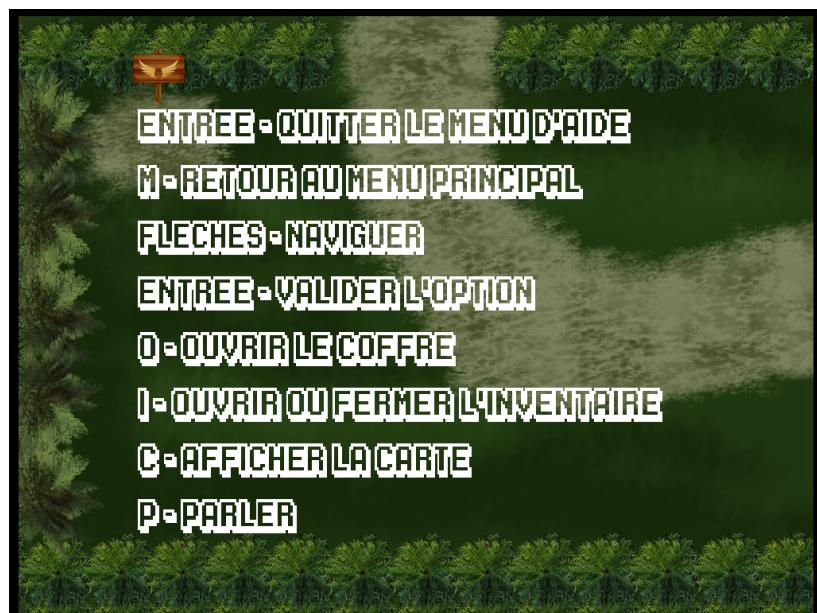


FIGURE 16 – Menu des commandes utiles dans le jeu