Telecom Nancy

Université de Lorraine

Rapport de projet C

NOM DU JEU

Auteurs :
Nicolas Bédrine
Raphaël Moulet
Vincent Albert

25 mai 2015

Table des matières

1 Présentation du sujet

1.1 Un rapide résumé

Le jeu que nous nous sommes proposés de réaliser est un beat'em'all et tower defense s'inspirant du jeu BoxHead. Le principe général est de survivre sur une carte à plusieurs vagues successives d'ennemis. Pour se faire, le joueur incarne un personnage en vue à la troisième personne qui peut se déplacer, attaquer à distance avec des sorts et construire des tours.

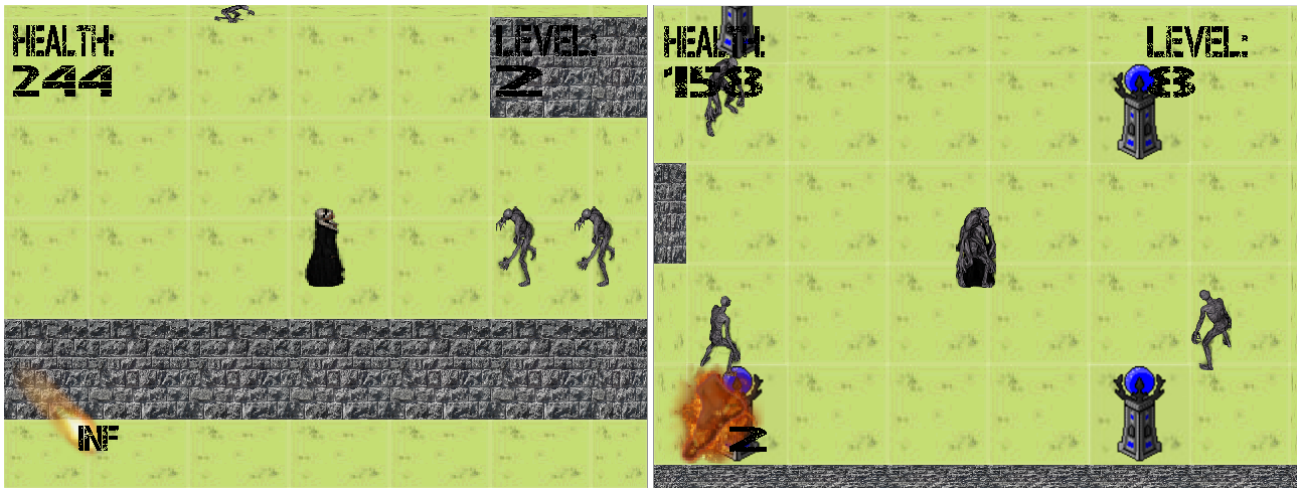

FIGURE 1 – Snapshots

1.2 Une licence pour les protéger tous

Le logiciel et l'ensemble des sources sont sous licence CC BY-NC-SA 3.0 FR.
Le partage et l'adaptation du logiciel sont permises à condition de ne pas l'utiliser commercialement, d'indiquer les changements effectués, de référencer l'œuvre originale et de conserver la même licence lors d'une diffusion ultérieure.

1.3 Le cahier des charges

Afin de gérer au mieux le projet, nous nous proposons d'établir un cahier des charges minimal dont les caractéristiques se retrouveront dans toute version finale du projet. En plus de cela, nous établirons une liste non ordonnée de fonctionnalités qu'il semble de prime abord intéressant d'implémenter. Leur mise en place découlera de leur difficulté ainsi que du temps disponible.
Cahier des charges minimal :

- Lancement d'une partie sur une carte unique
- Interaction avec le clavier pour se déplacer et attaquer
- Gestion de l'interaction avec l'environnement (collision, ...)
- Création d'un dispositif de défense
- Menu d'accueil

Fonctionnalités optionnelles :

- Capacités spéciales à usage limité
- Difficulté croissante des ennemis dans le temps (ennemis plus rapides, plus résistants, ...)
- Mode coopératif/deathmatch à deux en réseau ou sur le même clavier
- Mini mode aventure
- Menu pause
- Choix de la difficulté générale
- Éditeur de cartes et choix de carte

2 Organisation

2.1 Les outils utilisés

Le projet a été codé en langage C sous systèmes de type Unix. L'affichage graphique a été réalisé à l'aide de la bibliothèque SDL 2.0 qui est déjà utilisée dans de nombreux jeux. Les sprites sont libres de droits. Étant donné l'ampleur du projet et le nombre de participants, nous nous sommes convenus d'utiliser le gestionnaire de version git. Un dépôt public a été créé sur Github et est accessible à l'adresse suivante : `https://github.com/Neressea/projetC`

2.2 Le planning initial

Afin d'achever au mieux les objectifs que nous nous étions fixés, nous avons dressé un planning optimal des tâches à effectuer, en les répartissant par priorité. Voici le gantt des objectifs initiaux.

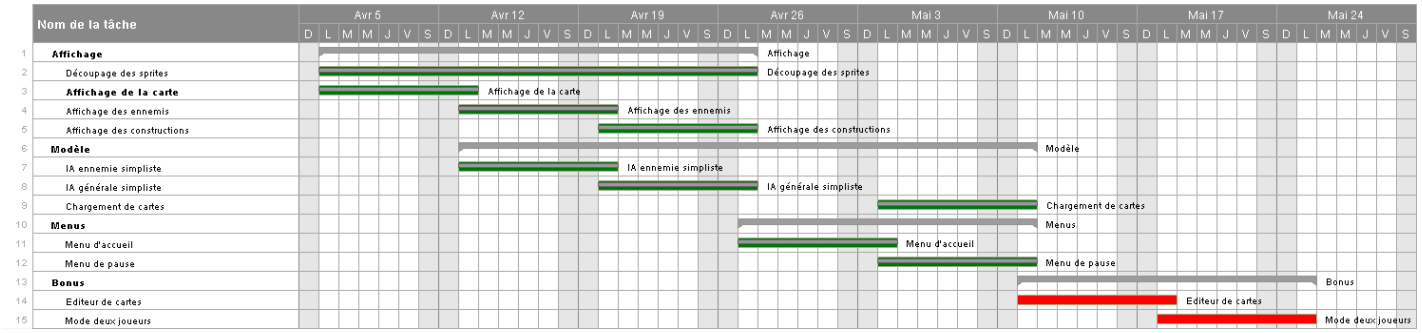

FIGURE 2 – Gantt du projet

Nous n'avons malheureusement pas eu le temps d'ajouter toutes les fonctionnalités optionnelles que nous avions prévu de faire. Cependant, nous en avons aussi ajouté de nouvelles en cours de développement .

2.3 UML des fichiers et fonctionnalités

Voici le schéma simplifié de la structure des fichiers. Seuls les structures et fonctions les plus importantes ont été représentées afin de conserver la lisibilité du schéma.

On peut remarquer l'organisation structurelle du code qui tend à se rapprocher d'un code objet. Les listes chaînées n'ont pas été représentées afin de ne pas encombrer l'UML.

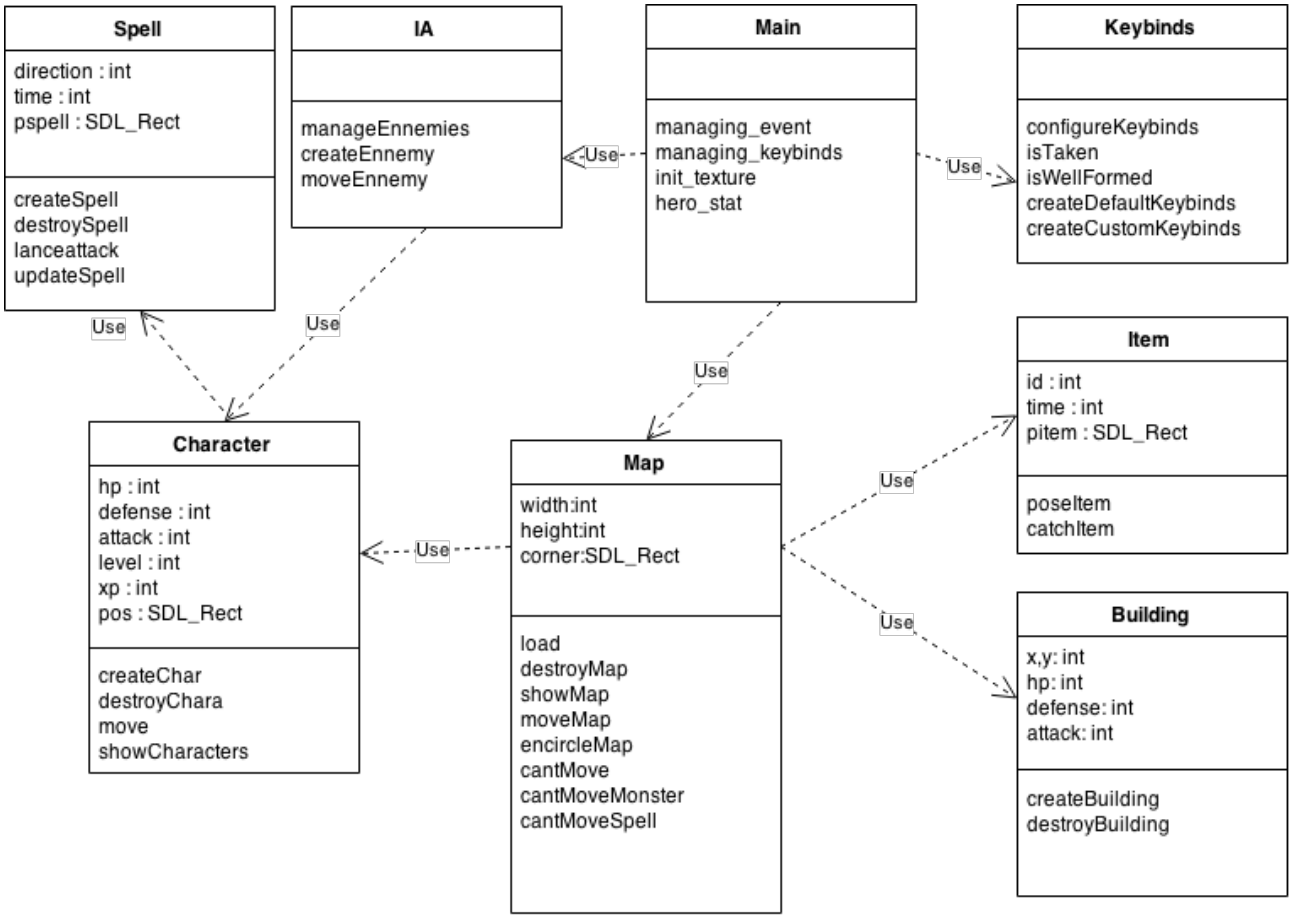
Spell
direction : int
time : int
pspell : SDL_Rect
createSpell
destroySpell
lanceattack
updateSpell
IA
manageEnnemies
createEnnemy
moveEnnemy
Main
managing_event
managing_keybinds
init_texture
hero_stat
Keybinds
configureKeybinds
isTaken
isWellFormed
createDefaultKeybinds
createCustomKeybinds
Use
Use
Use
Use
Use
Character
hp : int
defense : int
attack : int
level : int
xp : int
pos : SDL_Rect
createChar
destroyChara
move
showCharacters
Map
width:int
height:int
corner:SDL_Rect
load
destroyMap
showMap
moveMap
encircleMap
cantMove
cantMoveMonster
cantMoveSpell
Item
id : int
time : int
pitem : SDL_Rect
poseItem
catchItem
Building
x,y: int
hp: int
defense: int
attack: int
createBuilding
destroyBuilding
Use
Use
Use

FIGURE 3 – UML

3 La phase de développement

3.1 Les attentes atteintes

L'ensemble du cahier des charges minimal a été implémenté. De plus, la difficulté est proportionnelle au niveau du joueur. En effet, le nombre d'ennemis dépend du niveau du joueur, et donc des dégâts qu'il a infligés. Nous avons aussi ajouté une attaque à usage limitée. Des items sont laissés par les ennemis vaincus pour régénérer les points de vie du héros ou son sort limité. Des musiques et sons sont joués dans l'écran de sélection, durant la partie, et lorsque le joueur perd. Nous avons aussi implémenté la possibilité de changer les touches de contrôle. Cependant, les touches actuellement sélectionnées ne sont pas affichées et une touche déjà affectée doit être désaffectée avant d'être de nouveau assignée à une autre commande. Enfin, la vue a été adaptée pour que la carte soit redimensionnable en cours de partie.

3.2 Ce qui reste inachevé

Malheureusement, nous n'avons pas réussi à mettre en pratique nos autres idées. Nous avons tout de même préparé l'implémentation de certaines fonctionnalités. Ainsi, même si l'éditeur de cartes n'a pas été réalisé, le code a été adapté afin de pouvoir charger des fichiers textes formatés, ce qui facilite son ajout ultérieur.
Pour l'intelligence artificielle des ennemis, nous avions prévu d'utiliser l'algorithme de Dijkstra afin de créer des ennemis intelligents capables d'éviter les obstacles. Cependant nous n'en avons pas eu le temps ; nous avons donc décidé pour pimenter le jeu de jouer sur le nombre d'ennemis.

3.3 Les problèmes rencontrés

3.3.1 Quelques problèmes de communication

En nous confrontant pour la première fois à un travail de groupe de cette ampleur, nous avons rencontré quelques difficultés, dont certaines communicationnelles. En effet, lors de la recherche de sprites, nous avons commencé par les boules de feu ; puis nous nous sommes concertés pour la recherche des boules de glace. Cependant à cause d'un quiproquo, nous avons eu temporairement des sprites de cornets de glace.

3.3.2 Git et les machines virtuelles

L'un des membres utilisant une machine virtuelle, nous avons passé un certain temps à essayer de configurer la VM pour pouvoir pusher les fichiers sur github, en vain. Nous avons donc été contraint de transférer des fichiers sur un autre ordinateur et d'utiliser un autre compte github, ce qui explique les commits de seulement deux membres.

3.3.3 De la mémoire et des fuites

Après plusieurs longues séances de codage consécutives, nous nous sommes rendus compte que le programme provoquait un ralentissement général, voire l'instabilité du système d'exploitation. Nous avons rapidement repéré l'origine du problème. En effet, la mémoire vive consommée augmentait à chaque tour de boucle jusqu'à être saturée. En utilisant valgrind, nous avons pu mettre en évidence des textures chargées qui n'étaient pas détruites au sein de fonctions appelées dans la boucle principale.

3.3.4 Gestion événementielle

Nous n'arrivions pas dans un premier temps à gérer l'appui de plusieurs touches en même temps. Nous avons alors changé notre Wait Event par un Poll Event. Ce dernier a comme avantage de garder en mémoire plusieurs événements, contrairement à Wait Event qui les gère un par un. Les événements étant stockés dès le premier appel de la fonction Poll Event, un switch(Poll Event) n'est pas nécessaire. Nous avons ensuite décidé de créer un tableau de booléens contenant les états de chaque touche nécessaire au jeu. Il a donc fallu contrôler à la fois les pressions exercées sur les touches et leurs relâchements. Ce tableau nous a permis de gérer les événements de manière plus lisible, mais aussi de gérer grâce à une condition l'état de plusieurs touches à la fois.

3.3.5 La vue et le modèle

Nous avons essayé de respecter au mieux possible un pattern vue-modèle, en les dissociant au maximum. Ceci a entraîné de nombreuses phases de calcul pour l'affichage, étant donné que le modèle est un quadrillage alors que la vue peut "glisser" sur la carte pixel par pixel. De la même manière, un calcul a été nécessaire dans la gestion des collisions pour corriger cette différence bornée par la taille des cases (dans notre cas 100px). De plus, à cause des collisions avec les murs nous avons dû redécouper les sprites des monstres et des héros afin de les rendre plus réalistes. En ce qui concerne la collision entre les personnages, nous l'avions implémentée à l'origine, mais vu le nombre d'ennemis, le héros finissait rapidement bloqué par une horde de monstres, ce qui rendait les parties infaisables. Nous l'avons donc retirée, en privilégiant le gameplay au réalisme.

Références et autres sources d'inspiration

`http://blog.hikoweb.net/index.php?post/2011/11/06/Exemple-de-rapport-en-LaTeX`
Source pour le template LaTeXayant servi à la rédaction de ce rapport.

`http://www.crazymonkeygames.com/Boxhead-2Play-Rooms.html`
Source d'inspiration du jeu.

`https://wiki.libsdl.org/`
Wiki de la SDL 2.0.

`http://gnurou.org/`
Nous nous sommes inspirés des algorithmes de ce site pour la gestion des événements.

`http://jeux.developpez.com/tutoriels/sdl-2/guide-migration/`
Guide de migration de passage de la SDL 1.2 à la SDL 2.0. Ce site nous a été utile étant donné que nous avions déjà programmé avec la SDL 1.2 auparavant.

`https://creativecommons.org/licenses/by-nc-sa/3.0/fr/`
Licence s'appliquant au logiciel.

`http://www.pioneervalleygames.com/free-resources.html`
`http://opengameart.org/`
Sources des sprites libres utilisées dans le jeu.

`https://www.draw.io/`
Site qui a servi à la création de l'UML

`https://app.smartsheet.com` *Site qui a servi à la création du Gantt*

`http://soundbible.com/`
`http://www.audiomicro.com/`
`http://incompetech.com/music/royalty-free/`
Sources des fichiers audio utilisés dans le jeu.