Le Mans Université

Institut d'Informatique Claude Chappe
Le Mans Université

Le Mans Université

Licence Informatique *2ème année*

Module Conduite de Projet

Battle Ground

Lazare Maclouf Geoffrey Pose Valentin Charretier
Matthieu Brière

5 avril 2022

Table des matières

1 Introduction

Nous avons développé ce jeu dans le cadre du module de projet en deuxième année de licence Informatique. Plutôt que de choisir un des sujets proposés (mots-mêlés, othello...), nous avons décidé de créer notre propre jeu (nommé "Battle Ground") avec nos propres règles en nous inspirant de jeux déjà existants.
En l'occurrence ici deux jeux nous ont servi de base de travail : Age of war et Plant vs zombies 2. ils partagent un principe en commun : le joueur doit défendre sa base soit en y installant de plantes (Plants vs zombies) ou en installant des tourelles de défenses ainsi qu'en créant des unités (Age of war). Age of war fonctionne avec un seul mode de jeu (1v1), où le joueur est face à une base ennemie, c'est-à-dire l'ordinateur, qui accède aux mêmes fonctionnalités que le joueur (possibilité de créer les mêmes entités et les mêmes tourelles). Plants vs zombies quant à lui fonctionnent avec un mode jeu de survie où le joueur doit survivre à des vagues de zombies qui arrivent sur sa base.
Le jeu que nous avons créé reprend les deux concepts (le concept de vague ainsi que le mode 1v1). Le joueur a donc une base à défendre avec la possibilité de créer une tourelle et des unités comme dans Age of War. Cependant, l'ordinateur en face de lui ne joue que des vagues comme dans Plants vs Zombies.
Quelques contraintes devaient être respectées telles que l'utilisation de certains outils comme Doxygen (générer la documentation du code) ou gdb (débuguer le code) le tout sur un dépôt GitHub. On trouve sur ce dernier, la totalité du code source, les librairies, l'exécutable ainsi que toute la documentation. Nous verrons comment le jeu a été conçu, c'est-à-dire les différentes étapes qu'il a fallu mettre en place pour faire le jeu "Battle Ground".
En premier lieu nous aborderons la conception du jeu, puis nous présenterons les différents outils utilisés ainsi la gestion du projet. Par la suite nous verrons toute la phase de développement et enfin nous terminerons par les résultats obtenus et nous dresserons un bilan de ce qui a fonctionné et ce qui n'a pas ou moins bien fonctionné.

2 Conception du jeu

2.1 Règles du jeu

Plusieurs modes de jeu ont été choisis :
- Le mode "survivant" : comme son nom l'indique, si le joueur lance une partie en mode survivant, il devra se défendre face à une suite ces vagues d'entités. Le joueur a perdu s'il reste des entités et que le joueur n'a plus de vie. À l'inverse, le joueur gagne s'il lui reste des points de vie et qu'il n'y a plus d'entités.
- Le mode "Classique" : initialement, il devait s'agir d'un mode ou le joueur joue contre l'ordinateur. Cependant, par manque de temps, on a décidé d'en faire un mode de 1 contre 1 sur le même ordinateur.

2.2 Options du joueur

FIGURE 1 – options du joueur en pleine partie

Durant une partie le joueur à 4 actions options :
-Créer des unités : en effet, le joueur a la possibilité de créer des unités 1 qui vont défendre la base et attaquer les entités ennemies. En contrepartie le joueur dépense une certaine somme d'argent pour pouvoir en créer.

-Créer une tourelle : le joueur peut non seulement créer des unités mais aussi une tourelle qui, une fois installée sur la base, se met à tirer sur les entités ennemies qui s'approcheraient un peu trop près. Lorsque le niveau est fini (en mode survivant), la tourelle, si elle a été installée, est supprimée.

-Mettre en pause : le joueur peut mettre en pause et reprendre quand il le souhaite. Il lui suffit d'appuyer sur le bouton pause en jaune en haut au milieu de l'écran.

-Quitter la partie : le joueur peut quitter la partie ce qui n'est pas une fonctionnalité en soit. Cependant, lorsqu'il souhaite quitter la partie, un message d'alerte s'affiche à l'écran et le joueur est informé que la partie ne sera pas sauvegardé.

2.3 Jouabilité et fonctionnalités du jeu

En premier lieu, lors du lancement du jeu, un menu s'affiche à l'écran avec comme options "jouer", "paramètres", "quitter".

FIGURE 2 – menu principal

Lorsque le joueur clique sur quitter, le jeu se ferme instantanément. Lorsqu'il clique sur jouer, un autre menu s'afficher alors avec les options suivantes : "survie", "classique", "multi" et "retour". Survie lance une partie en mode survie, retour fait revenir le joueur au menu principal, classique lance le mode de jeux 1vs1 et multi amène au paramètre du mode en ligne, (fonctionnalité entourée en rouge sur la photo du jeu).

Lorsqu'une partie est lancée, on a une palette de boutons en haut de l'écran correspondant aux unités que l'on peut développer sur le champ de bataille, à la tourelle que l'on peut installer, à la fonctionnalité retour au

FIGURE 3 – sous menu jouer

menu et enfin à la fonctionnalité mettre en pause (entouré en rouge sur la photo ci-dessous).

FIGURE 4 – photo du jeu en mode survivant

Il y a également en haut à gauche la somme totale d'argent du joueur durant la partie. Ce dernier commence la partie avec 1000 dollars et collecte 250 dollars par unités tuées. Il peut ensuite créer des unités en payant le prix

affiché en dessous de chaque case correspondant à une unité. On s'attend à avoir une fluidité plus ou moins correcte. La marge d'erreur dépendant du système d'exploitation et surtout du processeur (globalement un peu plus fluide sur Windows que sur Linux).

3 Outils Utilisés

Pour développer le projet, nous avons dû faire appel à plusieurs outils dont certains indispensables.

3.1 GCC

Il s'agit du compilateur utilisé pour générer un éxecutable à partir du code source. De nombreuses options sont disponibles notamment pour le débuggage ou pour préciser une librairie spécifique à inclure pour la compilation. Il est compatible avec Windows, Mac-OS et Linux, Mais le projet fonctionne aussi avec le compilateur clang.

3.2 GDB

GDB est un débuggeur pour le langage c. Il a permis de détailler l'origine de nombreuses erreurs de segmentations (en grande partie liées aux listes chaînées).

3.3 Doxygen

Il s'agit de l'outil de documentation de code qui a été utilisé pour générer des schémas et graphes pour mieux comprendre les dépendances et interaction des fonctions grâce à un balisage minutieux du code source au préalable.

3.4 Discord

Cela a été notre moyen de communication durant le projet. En effet nous avons fait un groupe discord ce qui a été très pratique pour échanger sur diverses choses concernant le projet, notamment les avancés de chacun et partager des bouts de code.

3.5 GitHub

GitHub a été l'hébergeur en ligne de notre projet. c'est une plateforme gratuite et regroupant beaucoup d'outils et de programmes codés sous divers

langages. C'est dessus que l'ensemble des fichiers du projet sont stockés (du code source jusqu'à la documentation sans oublier les librairies, le makefile et les fichiers de ressources du jeu).

3.6 Atom et Visual studio code

Ces deux outils sont plus ou moins similaires et servent tous deux d'éditeurs de texte spécifique à la programmation. très pratique et gratuit, c'est dessus que nous avons codé le jeu. Visual studio code a la particularité d'avoir un environnement très complet avec des outils facilitant la vie du développeur et permettant des raccourcis (reconnaître une fonction et donner la doc en rapport avec, réorganiser et indenter le code pour qu'il soit lisible etc..).

3.7 Valgrind

Valgrind a permi de vérifier tout au long du développement du projet que le jeu n'avait pas ou peu de fuites de mémoires (finir avec 0 fuite de mémoires est compliqué). Et ainsi nous assurer que le jeu était correctement optimisé et n'allouait pas de la mémoire inutilement et surtout sans la libérer derrière.

3.8 Paint

Pour réaliser des les décors, les entités ainsi que tous les autres éléments à afficher, nous avons utilisé paint. Il s'agit d'un logiciel qui permet de créer ou modifier une photo avec un immense panel d'option. On peut ajouter du texte, suppersposer, découper des images, modifier les couleurs, dessiner avec diverses formes géométriques possibles. Enfin pour ce qui est du format on a la possibilité d'enregistrer la photo sous diverses format (bmp, png, jepg, gif). Cela a été pratique pour toute la partie graphique de Battle Ground.

3.9 Paint.net

Il d'agit d'une version améliorée et poussée de paint classique. En effet, on peut faire plus de choses comme modifier retirer toute une couleur d'une image d'un seul coup. Cette option a été très pratique pour supprimer l'arrière plan des entités et ainsi ne pas avoir de traces blanches sur l'écran.

3.10 Gimp

Gimp est un outil plus ou moins similaire aux deux précédents avec quelques particularités différentes. Ces trois logiciels ont été utiles car dans certains cas, en se retrouvant coincé, on peut réussir à modifier, créer l'image de la bonne façon en variant les logiciels. L'utilisation de trois logiciels différents pour la création des graphismes a été un atout puissant dans la création du jeu.

3.11 Cyotek spriter

C'est un outil servant à créer un unique sprite à partir de plusieurs images. En effet, avec des images séparées sur lesquelles on a une position unique à chaque image, cyotek spriter a permis quand nous le voulions de créer un sprite complet sur une seule et même image.

4 Gestion du projet

4.1 Organisation des tâches

Geoffrey Posé s'est occupé de générer le Doxygen final à l'aide du l'outil Doxygen ainsi qu'à organiser et créer une arborescence lisible, claire et structurée du code. Il s'est occupé de corriger certains warnings dans le code au fur et à mesure de l'avancement du projet et a créé la page internet sur laquelle nous avons pu faire notre diagramme de Gantt et la Documentation. Il nous a aidé à configurer entièrement le GitHub et à bien définir les environnements de travail (création de branches). Il a également produit un mode de jeu en 1v1 sur le même ordinateur.

Matthieu Brière a créé le git sur lequel le jeu a été déposé avec tous les fichiers qu'il comprend. Il s'est occupé de trouver et modifier certains sprites comme pour l'entité nommée "fighter". Il a modifié les boutons du menu afin de les rendre plus jolis. Il a aussi codé le fichier audio.c grâce auquel la musique du jeu peut fonctionner à l'aide de la librairie SDL mixer.

Lazare Maclouf a codé les fichiers c excepté le fichier audio.c et classique.c. , incluant les primitives grâces auxquelles le jeu fonctionne, l'interface graphique, la gestion des menus, les animations des entités, le mode survivant, les structures (joueur, entite, wave, msg, defense etc..), la gestion de l'environnement, du comportement des entités, de leur cohérence de déplacement etc..Il s'est occupé de rédiger le rapport final en latex. Il a trouvé, modifié et créé les images des décors, des entités bandit, mumma et voisin ainsi que pour l'argent, les cadres, les menus et les boutons (avec l'aide de

Matthieu Brière). Il a également créé les images contenant des polices d'écritures 3D (les images comme survivant, vous n'avez pas assez d'argent etc..).

Valentin Charretier a travaillé sur les sockets, pour tenter de créer un mode multijoueur. Ce qui aurait ainsi offert la possibilité de jouer en 1v1 avec deux ordinateurs distincts.

4.2 ajouts de tâches au fur et à mesure

Nous avons remarqué que le Gantt prévisionnel n'était pas parfait. En effet, de nombreuses tâches pour développer le jeu étaient manquantes. C'est pourquoi sur le Gantt effectif nous avons rajouté les différentes tâches qui manquaient (Tests, étapes supplémentaires pour coder le jeu...) au fur et à mesure de l'avancée du projet. On ne se rendait compte du besoin de certaines tâches qu'une fois en train de développer concrètement le jeu et qu'avant cela il était difficile de répertorier et lister tout ce qu'il fallait faire pour créer le jeu.

5 Développement

Le jeu a été codé avec les librairies SDL, SDL image ainsi que SDL mixer pour l'audio.

5.1 Structures utilisées

Plusieurs structures on été codées pour le bon fonctionnement du jeu. Les structures entite, wave et joueur par exemple sont essentielles lors du déroulement d'une partie. En effet, une wave 5 est une liste chaînée d'entité comportant chacune de nombreux paramètres. Parmi eux, on peut retrouver la valeur de l'abscisse de la barre, de l'ordonnée de la barre, de l'abscisse de l'entité, de l'ordonnée de l'entité, ses points de vie, son nombre de dégâts, une chaîne de caractère correspondant à l'image de l'entité pour l'animer etc..

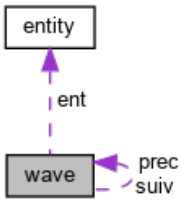


FIGURE 5 – liste chaînée

Ces structures sont gérées grâces à des primitives, contenues dans le fichier vague.c Toutes les fonctions de création d'une structure entité ou vague, de déplacement dans la liste de chaînée ou encore de suppression se trouve dans ce fichier. La structure joueur par exemple, contient les informations relatives au joueur comme son argent, ses points de vie etc.. Certaines structures peuvent paraître très longues (avec beaucoup de paramètres). Mais cela va se justifier avec les points suivant.

```
typedef struct entity
{
    void (*met_a_jour)(struct entity *);
    void (*charger_img)(struct entity *, SDL_Renderer *);
    int attaque;        /*valeur qui correspond à l'attaque si l'entité doit attaquer ou non*/
    int pv;
    int type;           /*attaque à distance ou de près*/
    int degat;
    int x;
    int y;
    int x_barre;        /*valeur qui correspond à la position de la barre de vie*/
    int y_barre;
    int w;              /*dimensions de l'image affichée à l'écran (largeur)*/
    int h;              /*dimensions de l'image affichée à l'écran (hauteur)*/
    int x_image;        /*absisse du début sur l'image*/
    int y_image;        /*ordonnée du début sur l'image*/
    int w_image;        /*dimensions de l'image dans le fichier png (largeur)*/
    int h_image;        /*dimensions de l'image dans le fichier png (hauteur)*/
    int temps;          /*valeur qui correspond au nombre d'itérations de boucles à attendre avant qu'il commence à avancer*/
    int nb_pos;         /*correspond au nombre de position dans le fichier en mode déplacement*/
    int nb_pos_attaque; /*correspond au nombre de position dans le fichier en mode attaque*/
    char nom_fichier[100];
    char nom_fichier_attaque[100];
    int montant;        /*indicateur permettant de savoir si les images sont en phases montantes ou ascendantes*/
} entite;


typedef struct wave
{
    entite *ent;
    struct wave *suiv;
    struct wave *prec;
} t_wave;
```

FIGURE 6 – structure vague et entitée

5.2 L'interface graphique avec SDL et SDL image

La bibliothèque SDL ne peut, par défaut que charger sur le rendu et afficher sur la fenêtre des images au format bmp.
Pour remédier à cela et afficher des images avec tout type d'extension, il a fallu utiliser la librairie SDL image. Toutes les fonctions relatives à l'interface graphique (affichage des menus, des images etc..) se trouvent dans le fichier interface.c Elles sont utilisées dans le reste du code lors de la gestion d'une partie par exemple.
Plutôt que de charger un tableau de textures en variable globale dans tout le jeu. Des fonctions indépendantes se chargent de toutes les étapes pour afficher une image à l'écran. De la création de la texture, jusqu'à la copie sur le rendu.

5.3 Animations des unités et des tirs

5.3.1 Technique d'animation des entités

Deux techniques d'animation ont été utilisées pour animer les entités.
La première consiste à charger successivement plusieurs images différentes sur lesquelles on a une position différente à chaque fois.
La deuxième consiste à charger une partie d'une image comportant toutes les positions successives d'une entité ou d'un objet (sprite) et à se déplacer dans le fichier pour charger les différentes positions de l'entité.
Étant tombé sur des entités avec des images différentes, le plus simple a été d'incorporer les deux méthodes d'animations au jeu.

5.3.2 Fonctionnement des animations globales

La SDL fonctionne avec un seul et unique rendu pour une fenêtre sur lequel on affiche ce qu'on veut. Ainsi, si l'on souhaite afficher un décors avec une unité bougeant dessus, il suffit simplement de superposer le décors et l'unité et donc les charger dans le bon ordre. Pour la faire bouger, il faut à chaque fois tout supprimer et tout recharger.
Plusieurs problèmes on été rencontrés :
-Un clignotement de certaines parties des images une fois affichées à l'écran (du à une mauvaise gestion de l'ordre d'affichage)
-Une synchronisation dans un premier temps de l'animation des entités lorsqu'elles sont plusieurs à se déplacer en même temps sur l'écran

Pour régler ces problèmes, les structures dont on a parlé plus haut on étés mises en place.
Concrètement avant d'afficher le rendu final à l'écran, dans la fonction gérant le partie (pour le mode survivant par exemple), on parcours toute la liste chaînée d'entité, pour laquelle chaque position est enregistrée et mise à jour. si une entité est créée après une autre, son animation ne sera pas la même que la précédente. Une fois parcouru toute les listes chaînées et charger les images successivement sur le rendu, puis en dernier on charge les images relatives au bouton, à l'affichage en haut (cadre dans lequel il y a les unités, le bouton pause, retour etc..). Cela permet ainsi une indépendance complète entre les entités

5.3.3 Animation des tirs

Comme pour l'animation des entités, une structure tir a été définie pour pouvoir animer indépendamment le tir des autres éléments du décors avec le même procédé que pour les entités, à la différence qu'il n'y a qu'un seul tir à la fois.

5.4 Gestion de l'environnement et des unités

De nombreux éléments pour avoir un jeu cohérent et fonctionnel ont été à prendre en considération et à faire. La vitesse de déplacement des entités, des tirs, l'arrêt d'une entité lorsqu'elle est devant un obstacle ou une autre entité... Pour se faire, il a fallu faire des fonctions vérifiant pour chaque entité (en parcourant à chaque fois la liste chaînée d'entité complètement) leur position relative aux autres entités ainsi que leur position absolue afin d'éviter qu'une entité ne continue à avancer et par conséquent sortir de l'écran ou passer à travers une autre entité. Toujours dans les structures des indicateurs de type int ont été mis en place afin de pouvoir ordonner l'arrêt d'une entité ou son mouvement lors de l'utilisation des fonctions d'affichages. Il existe deux types d'entités au sein de Battle Ground : les entités courtes et longues portées.

FIGURE 7 – niveau 2 partie survivant

Lors de la gestion des lignes d'entités, il faut prendre en considération la distance relative aux autres unités pour savoir si elle peut avancer ou si elle doit s'arrêter mais aussi pour savoir quand déclencher les attaques. Pour pouvoir ainsi faire jouer ces deux types d'entité en harmonie, il a fallu que les fonctions de gestion de l'environnement prennent en compte le type pour chaque entité et ainsi agir en conséquence (toujours avec un paramètre spécifique relatif dans la structure). Ainsi, Une bonne jouabilité et un jeu va avec un bon respect des lois physiques du jeu, un dynamisme des objets animés ainsi qu'une cohérence dans leur comportement (arrêt des entités face à un obstacle, reprise de déplacement lorsque la voie est libre, attaque lorsqu'une entité ou une base est à portée de main...).

5.5 Structure des fichiers et des fonctions principales

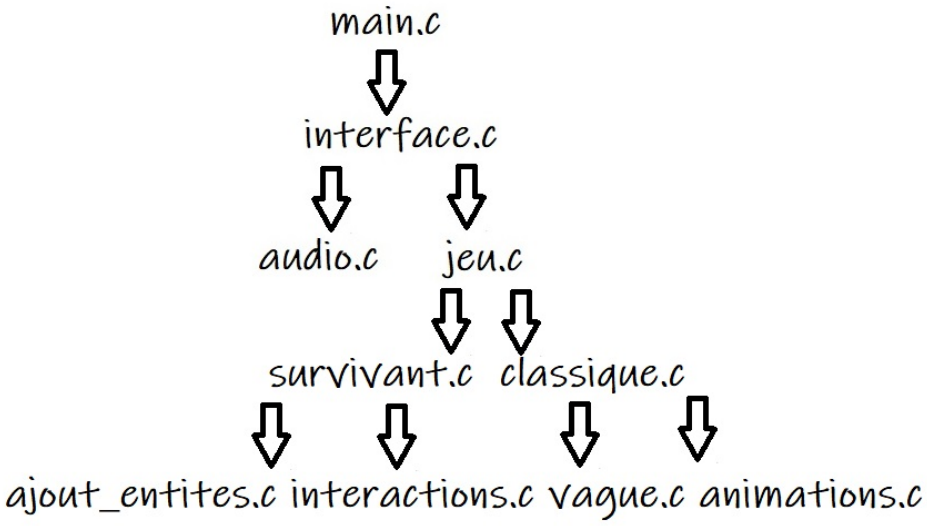


FIGURE 8 – schéma sur la hiérarchie des fichiers entre eux

Le code source du jeu Battle Ground est composé de 9 fichiers C ainsi que 9 fichiers H.

5.6 Tests

Lors de la création du jeu, de nombreuses fonctions pour manipuler les structures de données on été faites. Comme ces structures sont manipulées avec des pointeurs, si les fonctions ne sont pas correctement réalisées, cela peut donner lieu à des erreurs de segmentation.

C'est pourquoi des tests ont été fait pour l'utilisation de chacune des ces fonctions jusqu'à ce qu'elles fonctionnent parfaitement dans des fichiers à part, séparés du code source du jeu. D'autres tests pour la SDL ainsi que pour les fichiers audio ont également été faits.

6 Bilan

6.1 Résultats

Le jeu fonctionne globalement correctement. Nous avons dans l'ensemble fait différemment du diagramme de Gantt de départ.
En effet, on a fait face à plusieurs imprévus et plusieurs aspects que nous n'avions pas anticipé. Pour ce qui est de l'affichage, La SDL ne pouvant que simplement charger à l'écran ce qu'on lui demande et superposer les images,

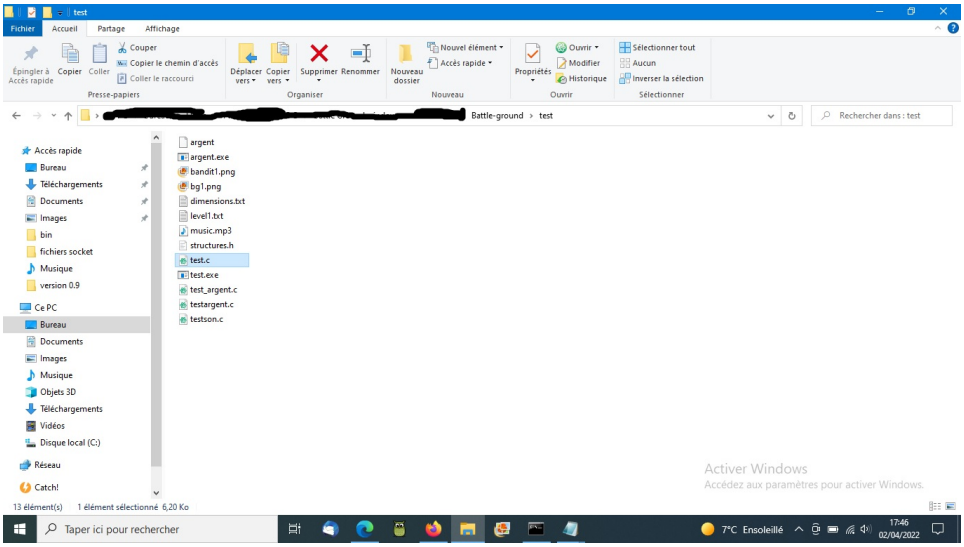


FIGURE 9 – dossier de test

il a fallu élaborer une méthode d'animation et d'affichage comme vu plus haut. On s'était notamment trompé sur le temps que chaque tâche allait nous prendre. Les phases de test ont été beaucoup plus longues que prévues. Avant d'avoir des structures de données et des primitives qui manipulent ces structures fonctionnelles, il a fallu du temps et beaucoup de tests ainsi que de débuggage avec gdb. Ne plus avoir d'erreur de segmentation n'a pas été chose facile. De plus, la coordination des éléments dans le jeu a été plus compliquée que prévue elle aussi. En effet, il a fallu voir et revoir les fonctions qui se chargent de l'affichage des différents éléments, afin de les placer dans le bon ordre, au bon moment et au bon emplacement sur l'écran. Pour ce qui est des boutons durant une partie, réussir à bien détecter au bon moment le passage de la souris au bon endroit et au bon moment a été assez difficile aussi car les autres fonctions (celles qui gèrent le joueur, les entités et autres) sont appelées en permanence. Lorsque le programme se trouve dans une fonction cela le bloque temporairement dans ce qu'il est en train de faire, ce qui peut empêcher l'exécution de choses en parallèles (l'utilisation de threads pour y remédier peut être utile comme nous le verrons dans les améliorations potentielles ci-dessous). Il a fallu donc calibrer et placer les instructions dans le bon ordre. Ces défauts ont ainsi pu être corrigés en parti avec une bonne structure du code.
L'équilibrage du jeu n'est pas optimal. Le jeu est en parti déséquilibré et on peut gagner à tous les coups avec la même stratégie. Le potentiel d'action de l'utilisateur est limité ce qui n'offre pas une très grande expérience de jeu.

En comparant le Gantt prévisionnel et le Gantt effectif, on peut voir que de nombreuses tâches n'avaient pas été anticipées et le nombre d'heures prévues pour les effectuées étaient souvent mal jugées (sous-estimées ou sur-estimées).

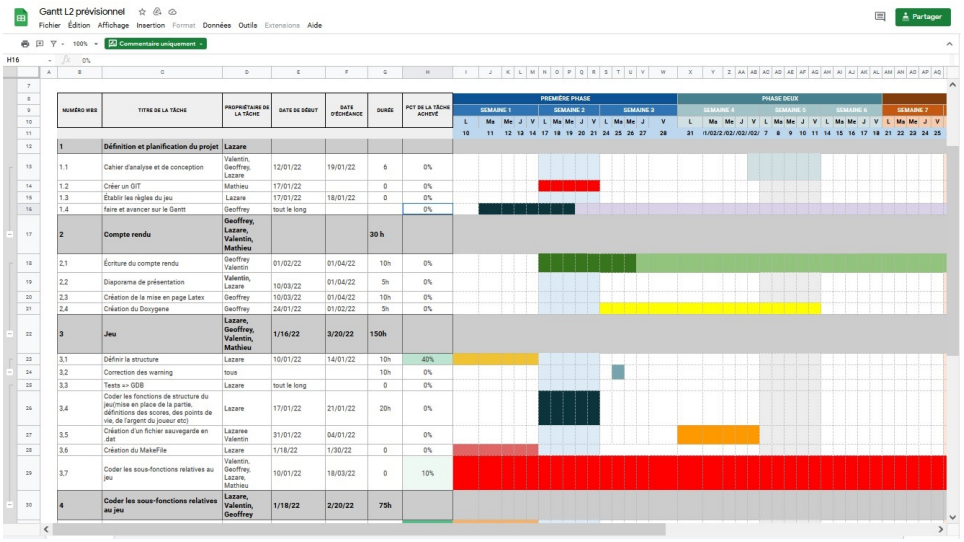


FIGURE 10 – Gantt Prévisionnel partie 1

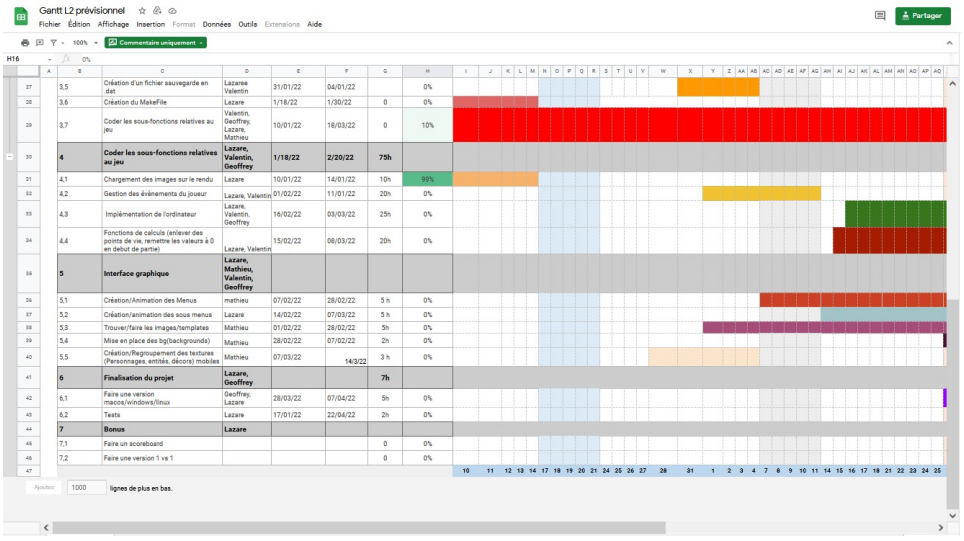
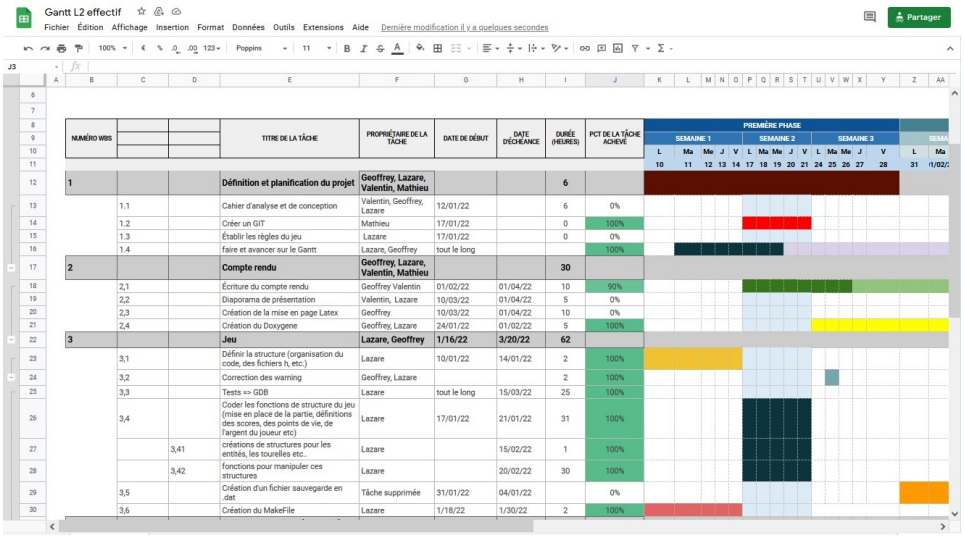
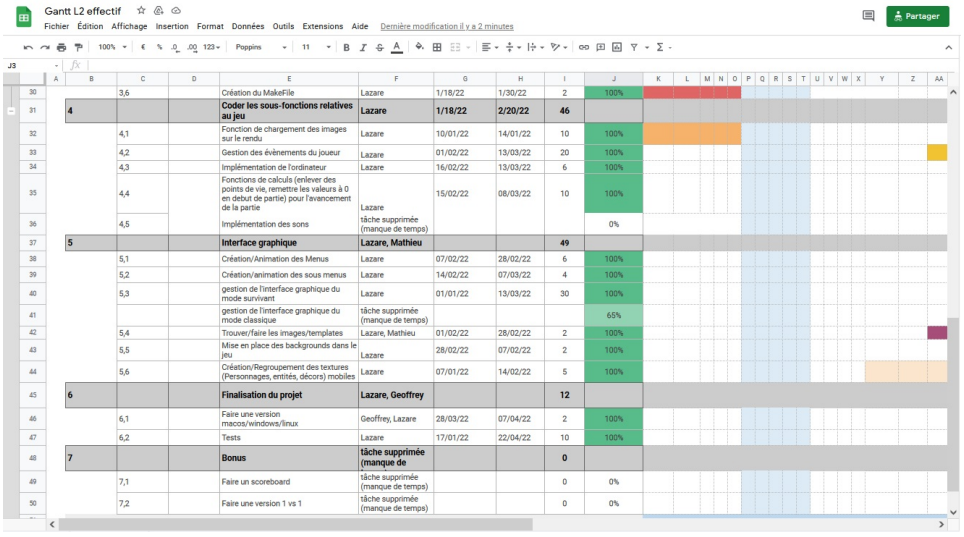


FIGURE 11 – Gantt Prévisionnel partie 2



NUMÉRO WBS			TITRE DE LA TÂCHE	PROPRIÉTAIRE DE LA TÂCHE	DATE DE DÉBUT	DATE D'ÉCHÉANCE	DURÉE (HEURES)	PCT DE LA TÂCHE ACHEVÉ
1			**Définition et planification du projet**	**Geoffrey, Lazare, Valentin, Mathieu**			**6**	
	1,1		Cahier d'analyse et de conception	Valentin, Geoffrey, Lazare	12/01/22		6	0%
	1,2		Créer un GIT	Mathieu	17/01/22		0	100%
	1,3		Établir les règles du jeu	Lazare	17/01/22		0	0%
	1,4		faire et avancer sur le Gantt	Lazare, Geoffrey	tout le long			100%
2			**Compte rendu**	**Geoffrey, Lazare, Valentin, Mathieu**			**30**	
	2,1		Écriture du compte rendu	Geoffrey Valentin	01/02/22	01/04/22	10	90%
	2,2		Diaporama de présentation	Valentin, Lazare	10/03/22	01/04/22	5	0%
	2,3		Création de la mise en page Latex	Geoffrey	10/03/22	01/04/22	10	0%
	2,4		Création du Doxygene	Geoffrey, Lazare	24/01/22	01/02/22	5	100%
3			**Jeu**	**Lazare, Geoffrey**	**1/16/22**	**3/20/22**	**62**	
	3,1		Définir la structure (organisation du code, des fichiers h, etc.)	Lazare	10/01/22	14/01/22	2	100%
	3,2		Correction des warning	Geoffrey, Lazare			2	100%
	3,3		Tests => GDB	Lazare	tout le long	15/03/22	25	100%
	3,4		Coder les fonctions de structure du jeu (mise en place de la partie, définitions des scores, des points de vie, de l'argent du joueur etc)	Lazare	17/01/22	21/01/22	31	100%
		3,41	créations de structures pour les entités, les tourelles etc..	Lazare		15/02/22	1	100%
		3,42	fonctions pour manipuler ces structures	Lazare		20/02/22	30	100%
	3,5		Création d'un fichier sauvegarde en .dat	Tâche supprimée	31/01/22	04/01/22		0%
	3,6		Création du MakeFile	Lazare	1/18/22	1/30/22	2	100%

FIGURE 12 – Gantt effectif partie 1



	3,6		Création du MakeFile	Lazare	1/18/22	1/30/22	2	100%
4			**Coder les sous-fonctions relatives au jeu**	**Lazare**	**1/18/22**	**2/20/22**	**46**	
	4,1		Fonction de chargement des images sur le rendu	Lazare	10/01/22	14/01/22	10	100%
	4,2		Gestion des évènements du joueur	Lazare	01/02/22	13/03/22	20	100%
	4,3		Implémentation de l'ordinateur	Lazare	16/02/22	13/03/22	6	100%
	4,4		Fonctions de calculs (enlever des points de vie, remettre les valeurs à 0 en debut de partie) pour l'avancement de la partie	Lazare	15/02/22	08/03/22	10	100%
	4,5		Implémentation des sons	tâche supprimée (manque de temps)				0%
5			**Interface graphique**	**Lazare, Mathieu**			**49**	
	5,1		Création/Animation des Menus	Lazare	07/02/22	28/02/22	6	100%
	5,2		Création/animation des sous menus	Lazare	14/02/22	07/03/22	4	100%
	5,3		gestion de l'interface graphique du mode survivant	Lazare	01/01/22	13/03/22	30	100%
			gestion de l'interface graphique du mode classique	tâche supprimée (manque de temps)				65%
	5,4		Trouver/faire les images/templates	Lazare, Mathieu	01/02/22	28/02/22	2	100%
	5,5		Mise en place des backgrounds dans le jeu	Lazare	28/02/22	07/02/22	2	100%
	5,6		Création/Regroupement des textures (Personnages, entités, décors) mobiles	Lazare	07/01/22	14/02/22	5	100%
6			**Finalisation du projet**	**Lazare, Geoffrey**			**12**	
	6,1		Faire une version macos/windows/linux	Geoffrey, Lazare	28/03/22	07/04/22	2	100%
	6,2		Tests	Lazare	17/01/22	22/04/22	10	100%
7			**Bonus**	**tâche supprimée (manque de**			**0**	
	7,1		Faire un scoreboard	tâche supprimée (manque de temps)			0	0%
	7,2		Faire une version 1 vs 1	tâche supprimée (manque de temps)			0	0%

FIGURE 13 – Gantt effectif partie 2

comme on peut le voir sur les deux photos représentant le Gantt effectif, de nombreuses tâches ont été réalisées à cent pour cent, ce qui est une bonne nouvelle du point de vue de l'organisation. Cependant on peut aussi voir des tâches inachevées telles que l'interface graphique du mode classique. et enfin, des tâches jamais effectuées soit par manque de temps, soit parce qu'après coup elles ont été inutiles au projet.

6.2 Améliorations potentielles

Comme on a pu le constater le jeu n'est pas fonctionnel à cent pour cent. En effet, il y a des petits bugs d'affichage, des petits problèmes de fluidité par moment. Et pour ce qui est de la jouabilité, elle peut être améliorée. C'est-à-dire que tout ce qui concerne l'équilibrage du jeu, incluant la difficulté, les possibilités du joueur, les évènements imprévus (avalanches de météorites qui détruisent les unités) pourrait être grandement peaufiné. Cela donnerait une expérience beaucoup plus variée et riche à l'utilisateur. La plupart des jeux de stratégies comme Age of war ou Plants vs zombies présentent un panel d'options et de possibilités pour se défendre et attaquer ce qui fait que l'utilisateur n'a aucune garantie de gagner selon juste une méthode particulière et doit rester attentif du début jusqu'à la fin de la partie. Concrètement, pour ce qui est de la défense, on pourrait faire en sorte qu'à chaque niveau il y ait beaucoup plus de types d'unités différentes et avec une fréquence variable. Le manque de diversité d'entités qui attaquent le joueur est notable au niveau de l'expérience utilisateur. De plus, il suffit de suivre une seule stratégie pour gagner. En effet, il suffit de placer la même unité au bon moment régulièrement et continuer jusqu'à ce que l'on gagne.
On pourrait également faire en sorte que le jeu soit beaucoup plus optimisé et opérationnel en chargeant un tableaux de textures directement au début du jeu plutôt que de tout recharger à chaque fois que l'on affiche une image. De plus on pourrait scinder toutes les différentes tâches à l'aide de threads pour rendre le tout plus fonctionnel et opérationnel. Pour ce qui est de l'affichage et de la gestion de la vie par exemple, dans une fonction, tout est traité et effectué en décalé. On parcours les listes chaînées d'entité et on met leur position, leur pv, leur barre de vie, leurs statuts etc à jour puis on regarde si l'utilisateur clique quelque part et si le joueur a encore de la vie et enfin on affiche tout dans le bon ordre à l'écran. Le plus judicieux aurait été de faire un thread pour l'animation des entités, un thread pour la gestion du statuts des entités, un thread qui s'occupe d'écouter les évènements etc.. Pour ce qui est des sons, on pourrait faire en sorte d'ajouter tous les bruitages nécessaires (percussion d'une unité, de tir, de défaite, de douleur, d'effort, de victoire etc..) et ainsi obtenir un programme complet en termes d'expérience de jeu. De plus, nous n'avons pas eu le temps de développer un mode classique fonctionnel ni un mode multijoueurs. Dans l'idéal, on pourrait compléter avec un mode multijoueurs en local ou avec la possibilité de jouer contre un autre joueur avec deux ordinateurs distincts ainsi qu'un mode classique avec une Intelligence Artificielle. L'option "paramètres" du menu n'a finalement pas été utile et a servi de décoration. On pourrait à l'avenir faire en sorte qu'on puisse régler le volume du jeu ainsi que la taille et la résolution de l'écran.

6.3 Conclusion

Ce projet a été une expérience très enrichissante pour de nombreuses raisons. La première est que pour la première fois il a été possible de mettre en pratique les compétences accumulées durant les cours car les TP, les TD et les examens sont des exercices mais ne sont pas une mise en situation réelle. L'intérêt ici a été de concevoir du début à la fin un jeu, et de coder tous les aspects du jeu avec les savoir faire acquis jusqu'à présent. Ce projet a été une occasion concrète de sortir de sa zone de confort et de se confronter à l'inconnu et par la même occasion de tirer des leçons sur ce qui fonctionne et ce qui fonctionne moins que ça soit dans l'organisation du travail ou dans les techniques de programmation. La nouveauté a été de gérer entièrement un projet et pas se contenter de coder c'est-à-dire concevoir les règles du jeu, l'environnement de travail, préparer les différents outils, la documentation, le git, coder avec plusieurs fichiers c séparés ainsi que produire une grande quantité de code pour obtenir un seul exécutable, s'occuper des autres aspects comme les graphismes et ainsi gérer les décors, les objets, les sprites pour les animations du jeu, les sons, les menus etc.. Ce projet a demandé plus d'investissement, une réelle implication ainsi qu'une organisation bien spécifique.
A l'avenir cela nous permettra d'être capable de coordonner et répartir les tâches en fonction des disponibilités et des envies des autres, se plier à un cadre avec certaines contraintes et trouver des solutions lorsqu'on se retrouve face à une impasse ou un problème. Cela nous permettra également d'être réaliste quant à la faisabilité d'un projet et d'être capable d'adapter en fonction des moyens à notre disposition pour réaliser ce projet.
Pour réaliser un jeu avec une interface graphique qui fonctionne correctement, il a fallu se documenter de notre côté, s'exercer nous-mêmes, faire des tests jusqu'à que cela marche, trouver les informations correspondant à ce que l'on veut faire etc.. Cela a donc fait également appel à notre capacité à travailler en autonomie et en autodidacte. Nous avons également appris à respecter un planning de travail en dehors des cours à l'université, et continuer jusqu'à ce que le résultat souhaité soit obtenu.
Pour ce qui est de l'aspect informatique et de la technique, ce projet nous a permis aussi d'apprendre beaucoup sur la librairie SDL et sur le codage d'une interface graphique en général. Dans ce module, nous avons pu apprendre sur la SDL mais aussi davantage sur le C en lui même (au niveau des structures et des pointeurs) car nous avons fait face à des erreurs auxquelles nous n'avons jamais fait face auparavant. Enfin, nous avons appris à configurer tout l'environnement de travail notamment pour réussir à compiler en utilisant les librairies supplémentaires (SDL, SDL image et SDL mixer) de sorte à ce que le jeu fonctionne sur n'importe quel environnement de travail et pas exclusivement sur celui avec lequel on a développé le jeu.

7 Sources

Références

[1] https ://pixabay.com/fr/

[2] https ://www.gamedevmarket.net/asset/2d-field-parallax-background/

[3] https ://images.google.com/

[4] https ://lf2.net/

[5] https ://www.youtube.com/c/Formationvid

[6] https ://wiki.libSDL.org/

[7] https ://mat7813.github.io/Battle-ground/doc/html/index.html