METAL SHADOW WARS

IMAC 1 - S2

Programmation / Synthèse de l'image

RAPPORT DE PROJET

Nils **LANDRODIE**

Paul **ROZAIRE**

Vincent **SCAVINNER**

IMAC 1 - 2019 / 2020

LIEN DU PROJET

https://github.com/N0Ls/Metal-Shadow-Wars

Notes :

Touches de débug

M = Mettre en pause la musique

Q = Quitter le jeu

Flèche directionnelle de droite = passer au tour suivant

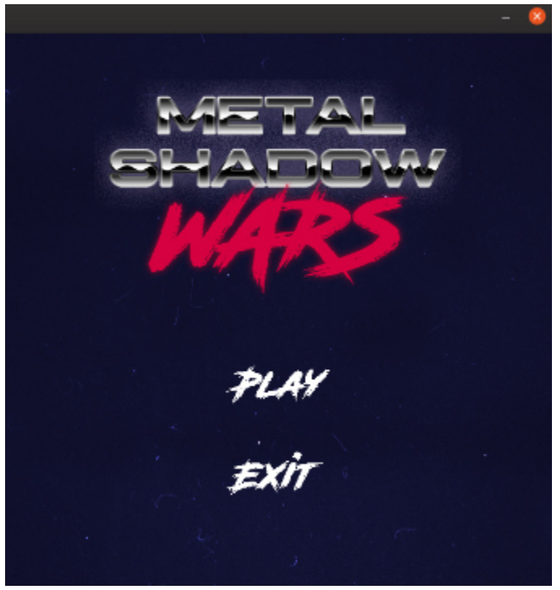
SYNOPSIS DU JEU / DIÉGÈSE

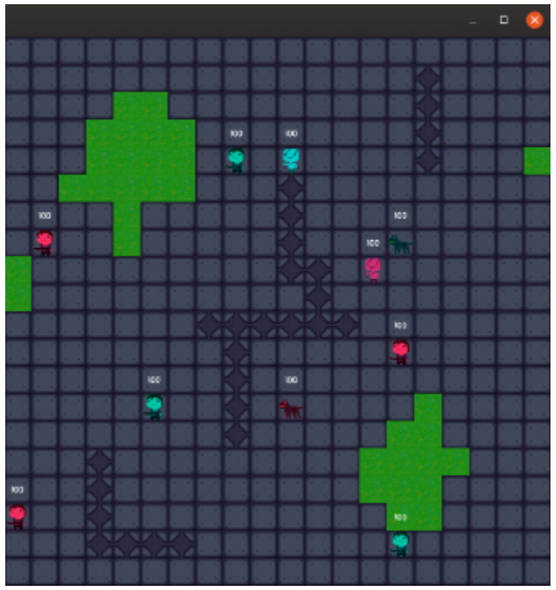
Vous vous retrouvez plongé en 2102, dans un monde sombre et dystopique où les derniers hommes survivants côtoient des êtres artificiels à l'apparence humaine créés par manipulation génétique.

La société s'est effondrée et la loi du plus fort a repris le dessus. Les humains ont recours à tour de bras à des améliorations cybernétiques pour tenter de concurrencer les robots et les êtres artificiels.

Des cendres de la société sont nés des gangs qui sont aujourd'hui les organisations dominantes et contrôlent de larges zones. Au quotidien ces gangs s'affrontent sans relâche pour savoir qui va prendre le contrôle de la ville.

Dans ***Metal Shadow Wars***, vous vous retrouvez justement au cœur d'une de ces rixes et vous devez lutter corps et âme pour terrasser l'adversaire et ainsi conserver votre territoire.





IMAC 1 - 2019 / 2020

MODE D'UTILISATION

Deux joueurs s'affrontent sur une carte de 20 cases par 20. Chaque joueur a un ensemble d'unités à contrôler dans le but d'anéantir son ennemi.

Il existe pour le moment 3 types d'unités différents, chacun ayant ses spécificités. Leurs attributs comme la force, la dextérité, la portée diffèrent, mais il n'y a pas d'impact sur leur capacité à s'affronter.

Dans le jeu vous pourrez ainsi retrouver :

- L'humain / cyborg
- Le robot
- Le chien d'attaque

Metal Shadow Wars est basé sur le jeu *Advance Wars. Il* s'agit donc d'un jeu au tour par tour. À tour de rôle, chaque joueur va pouvoir déplacer ses unités et attaquer à la suite de ce déplacement.

Chaque action possible est matérialisée visuellement sur la carte. Une fois une unité sélectionnée, vous verrez donc s'afficher en orange la zone dans laquelle elle a la capacité de se déplacer.

Une fois ce déplacement réalisé vous n'aurez pas d'autre choix que d'attaquer. Si vous avez pu vous rapprocher suffisamment d'une unité ennemie et qu'elle se situe dans votre zone de tir (zone bleue) vous pourrez alors lui infliger des dégâts.

Attention, car le tir allié est activé et vous pourriez donc par inadvertance (ou pas) blesser vos camarades de gang.

Si vous n'avez pas eu la chance de vous rapprocher suffisamment de l'ennemi, vous aurez toujours la possibilité de détruire l'environnement pour vous défouler ou alors pour vous frayer un passage direct pour le prochain tour.

Quoi qu'il en soit, le but est de triompher de l'adversaire en réduisant à néant toutes ses unités. Cependant, il ne se laissera pas faire. Ne vous reposez donc pas sur vos lauriers et soyez stratégique dans votre approche.

METAL SHADOW WARS

MÉTHODES UTILISÉE DANS L'IMPLÉMENTATION

L'organisation du code était pour nous un point indispensable. Ayant rapidement eu conscience de la taille plus ou moins conséquente qu'allait prendre le projet, nous avons décidé de diviser notre code en plusieurs fichiers sources, chacun ayant son fichier d'en-tête. Nos fichiers sont séparés de sorte à former des groupes de fonctions étant de mêmes types ou servant à traiter des éléments similaires.

Bien entendu, nous avons automatisé la compilation avec un fichier makefile.

DIFFICULTÉS RENCONTRÉES

Une de nos principales difficultés concernait l'utilisation de SDL1.2. N'ayant vu que SDL1.2 en cours nous avons été effrayé à l'idée de passer à SDL2.0 et avons donc préféré rester sur un terrain connu. Cependant, l'utilisation de cette ancienne version a été assez problématique pour nous. En effet, nous avons trouvé qu'elle n'était pas adaptée à notre contexte, contexte dans lequel nous devions s'auto former et résoudre de manière autonome nos problématiques. Dès que nous tentions de chercher une solution en se renseignant sur Internet, nous tombions sur de sujets concernant la SDL2, qui est celle essentiellement utilisée aujourd'hui, et qui peut parfois être difficilement adaptable dans les situations que nous rencontrons.

Il nous est parfois arrivé de rencontrer des problèmes concernant nos environnements de travail. Nous utilisions des machines virtuelles et nous rencontrions parfois des problèmes de performance, qui pouvait s'avérer très handicapant. La résolution de ces problèmes annexes nous faisait perdre un temps considérable sur nos temps de développement.

Nous nous sommes aussi confronté à une autre difficulté qui a été l'utilisation de GIT. En effet, n'ayant pas d'expériences et son utilisation n'ayant été que brièvement évoqué en cours, nous nous sommes retrouvés à plusieurs reprises gênés par ignorance sur le sujet. Notre solution a donc été malheureusement de travailler avec GIT de la manière la plus basique possible en utilisant les branches le moins possible et utilisant seulement les fonctionnalités les plus simples.

Enfin le A* a été relativement long à mettre en place mais nous allons détailler cette partie par la suite.

IMAC 1 - 2019 / 2020

STYLE GRAPHIQUE

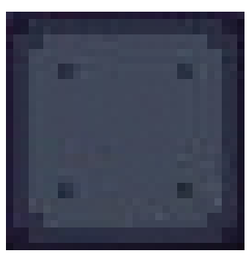
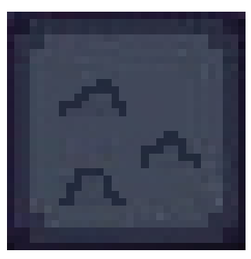
Étant pour plusieurs de grands amateurs de jeux vidéo, nous avons dès les prémices du jeu réfléchis au style graphique et diégétique dans lequel nous voulions que notre jeu s'inscrive. Nous avons donc pour cela essayé d'apporter un concept et un environnement unique. Pour cela, nous nous sommes tournées vers un univers rétro-futuriste, avec des références au cyberpunk, à la vaporwave et à la retrowave. Nous avons conçu chacun des éléments visibles à l'écran nous-mêmes.

Nous avons fais le choix du pixel art, car il permet de n'avoir à créer des assets graphiques que très peu détaillés tout en donnant un caractère au jeu. Nous trouvons qu'il s'applique notamment très bien au thème de notre jeu.

Pour ce qui est des assets en eux-mêmes, ils ont tous été réalisés par l'équipe.

Les personnages sont inspirés de Tiny Tower, mais ont été customisés spécialement pour notre style.




METAL SHADOW WARS

POINTS INTÉRESSANTS

A Star

Étant conscients de la difficulté et appréhendant un peu l'algorithme A Star, nous avons décidé de nous y atteler très tôt dans la phase de développement. La première partie de ce travail a été très longue, car il s'agissait de comprendre son fonctionnement. En effet les membres de l'équipe étaient très peu familiers avec les algorithmes de plus court chemin, nous avons du prendre notre temps afin d'appréhender sereinement le fonctionnement cet algorithme en particulier. Pour cela, nous avons donc croisé différentes ressources disponibles sur Internet et rédigé des notes de pseudo-codes.

Lors de la phase de réalisation, nous nous sommes confrontés à plusieurs difficultés. Bien évidemment, notre compréhension n'ayant pas été parfaite, nous avons dû à plusieurs reprises revenir sur les ressources vues précédemment pour affiner notre code et y détecter les erreurs.

La conception en elle-même de l'algorithme est aussi problématique, car les concepts utilisés, comme la boucle while, sont très rapidement source de bugs. De même, nous devions accéder à beaucoup de ressources et devions travailler beaucoup avec la gestion d'adresses mémoires. Nous nous retrouvions donc souvent avec des segfaults. Ce genre d'erreur est compliqué à gérer et chronophage à débugger, car l'erreur n'est pas explicite et force à relancer à chaque fois l'instance de notre jeu.

Enfin nous avons aussi perdu beaucoup de temps en créant des fonctions pour notre utilisation de liste chaînées alors que nous aurions pu simplement utiliser les librairies existantes (std:list). Malheureusement, nous n'en connaissions pas l'existence sur le moment.

Après un temps considérable nous avons fini par obtenir un algorithme A star fonctionnel bien que sûrement mal optimisé.

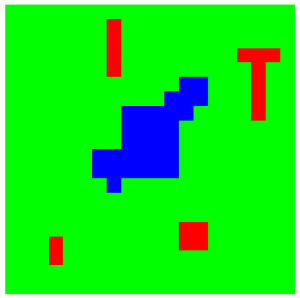
Par la suite, la promotion a reçu de l'aide de la part de Steeve Vincent pour cet algorithme. Par peur de modifier (donc possiblement de le faire dysfonctionner) et par satisfaction d'un travail de longue haleine, nous avons préféré conserver notre version personnelle de A Star. Ainsi, notre algorithme comporte sûrement quelques erreurs persistantes mais très rares ainsi que des soucis d'optimisation, mais il illustre notre travail et notre compréhension de l'algorithme.

Malheureusement nous n'avons pas eu le temps d'implémenter un arbre binaire de recherche et donc, à ce jour, notre algorithme fonctionne toujours avec des listes.

IMAC 1 - 2019 / 2020

Chargement de la carte par image

Nous avons fait le choix de pouvoir charger dynamiquement la carte à partir d'une image png. Il s'agit d'une fonctionnalité intéressante car elle permet de créer et mettre en place rapidement des nouvelles cartes de jeu.



Le fonctionnement est simple, nous créons une image de la taille de notre carte, dans notre cas du 20 par 20 pixels, et nous assignons différentes couleurs en fonctions des types de cases qu'elle représentent. Le programme lit ensuite chaque pixel et créer une case avec des propriétés en conséquence.

La réalisation a été un tout petit plus compliqué que ce que nous pensions car il réside une certaine difficulté dans la lecture des couleurs des pixels en SDL. Après différents essais en se basant sur la documentation de la librairie SDL croisés avec des exemples issus d'internet, nous avons finalement réussi à implémenter cette fonctionnalité. Elle représente un ajout que nous avons trouvé intéressant à la fois du point de vue de la programmation avec l'utilisation approfondie de SDL mais aussi d'un point de vue pratique pour la mise en place de nos cartes de jeu.

Joueur Artificiel

La consigne du projet mettait en avant la possibilité de pouvoir affronter un joueur dirigé par l'ordinateur. Une fois, toutes les fonctions du joueur « réel » misent en place, nous avons réutilisé ces dernières afin que l'ordinateur s'en serve pour avoir sa chance face à nous humains.

Nous avons commencé par faire un joueur « bête » qui se déplaçait de manière aléatoire sur la carte. Il était alors relativement aisé de le vaincre.

Nous avons ensuite opéré plusieurs changements itératifs pour rendre ce joueur artificiel meilleur. Nous avons donc programmé un algorithme détectant l'unité ennemi la plus proche et faisant se déplacer l'unité dans cette direction. Nous avons essayé, dans la mesure de nos compétences de rendre ce joueur artificiel, le plus « intelligent » possible et les résultats ce sont faits sentir. En effet à présent, le joueur artificiel est beaucoup plus efficace et ajoute une difficulté au jeu.

Nous en sommes donc assez satisfaits.

METAL SHADOW WARS

MÉTHODE DE TRAVAIL

Pour ce projet, nous avons mis en place un github, nous permettant ainsi de versionner et d'avoir une bonne visualisation des changements réalisés par les membres du groupe.

Durant la phase de développement, nous avons élaboré des scénarios afin d'effectuer des tests.

En fin de phase de développement, nous avons effectué du refactoring, notamment afin de supprimer toutes les étapes, fonctions et variables nous ayant servis lors de différents tests.

Enfin, la dernière étape a consisté à réaliser la documentation du code et à le commenter de la manière la plus claire possible.

POINTS À AMÉLIORER

Nous sommes dans l'ensemble satisfaits de notre projet. Cependant, nous pouvons relever quelques points d'améliorations du jeu :

Utilisation d'un arbre binaire de recherche pour le A Star : Comme évoqué auparavant, nous n'avons pas eu le temps d'ajouter l'utilisation d'arbres binaires pour dans l'algorithme A Star. Il serait donc judicieux pour accélérer la recherche, notamment dans le parcours des nœuds, d'ajouter cette fonctionnalité.

Optimisation du framerate : Afin d'optimiser les performances, il serait peut-être intéressant de ne pas appeler certaines fonctions à toutes les frames du jeu.

Plus grande utilisation de la POO : Nous avons commencé à mettre en place des classes dans notre programme. Cependant, leur utilisation est un peu sous-exploitée et il serait donc peu être intéressant, à l'avenir, de creuser plus dans ce domaine.

Meilleur découpage du code : Bien que nous soyons très à cheval quant à la qualité et la clarté de notre code, nous nous sommes parfois trop empressé de coder et n'avons pas assez réfléchi en amont à l'utilisation et la répartition de notre code. Cela nous laisse donc une impression de redondance de certains morceaux du code que nous aurions pu optimiser. À l'avenir, nous reverrions donc ces points et découperions plus notre code.

IMAC 1 - 2019 / 2020

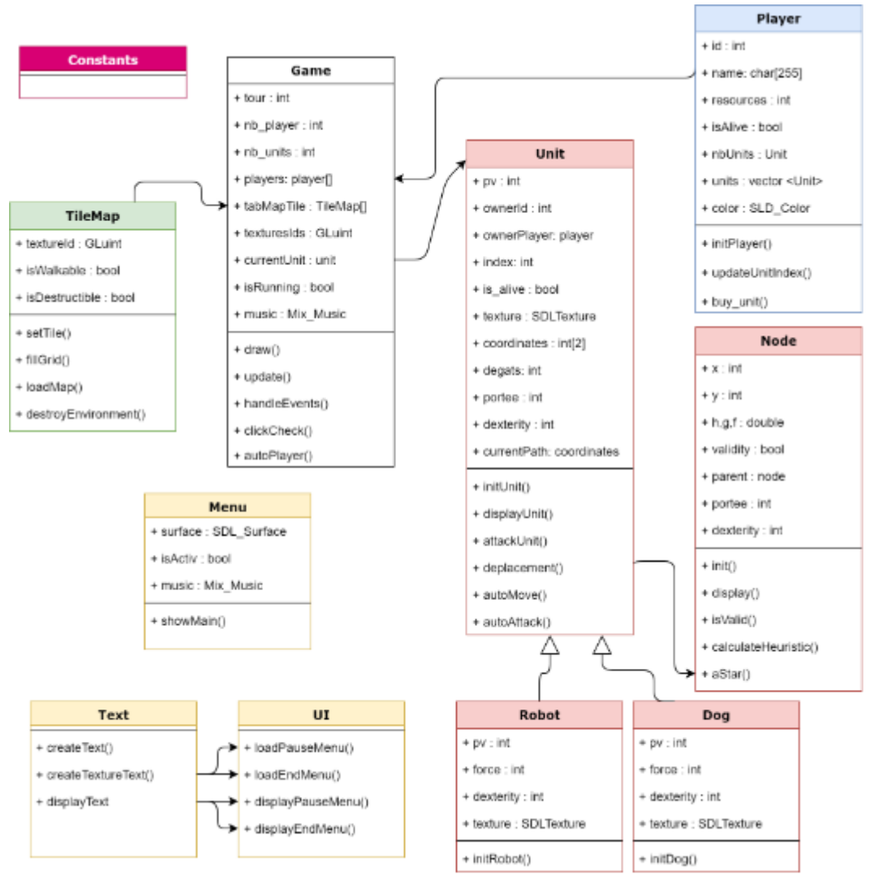
FONCTIONNALITÉS À IMPLÉMENTER

Le jeu est fonctionnel. Et si une seconde phase de développement se mettait en place, il pourrait être judicieux d'implémenter de nouvelles fonctionnalités :

- Système d'achat
- Animation des personnages
- Animation plus poussée de l'environnement et de l'univers
- Plus d'informations utilisateurs (informations liées aux actions par exemple)

SCHÉMA GLOBAL

Ci-dessous, vous trouverez un diagramme de classes représentant notre programme. Notez qu'il s'agit d'un diagramme résumé, qui a pour but de vous décrire brièvement le choix d'architecture adopté pour notre jeu. Il permet avant tout de décrire les relations adoptés par nos différents composants.



IMAC 1 - 2019 / 2020