Valentin MARTIN

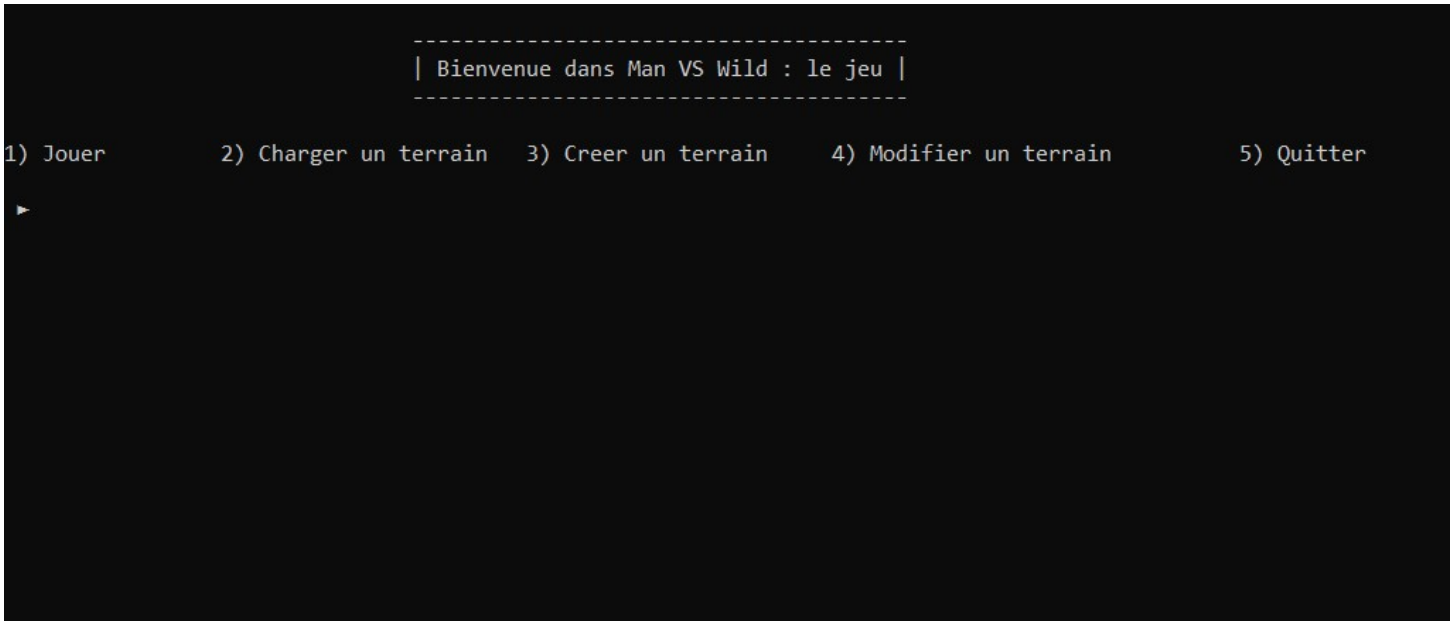
Antoine BOCH

Antoine FONTAINE

Téo BANNWARTH

ManVsWild

Projet de qualité de programmation



https://gitlab.com/DarkVex0s/manvswild

Rendu le vendredi 6 janvier 2023 à 14h

Table des matières

Introduction

Description du projet

Le but de ce projet est de créer un jeu de plateau (jeu au tour par tour jouable par 1 joueur pour le moment). Sur le plateau se trouve différents éléments de jeu (obstacles, fauves) ainsi que le joueur qui se déplace de case en case dans le but de survivre face aux fauves tentant de le manger. A chacun de ses déplacements, les fauves bougent eux aussi et le joueur doit essayer de faire tomber tous les fauves dans les pièges afin de gagner. A chaque tour que le joueur survit, son score augmente tout comme lorsqu'il arriver à piéger un fauve.

Il existe différents types de fauves qui ne se déplacent pas de la même façon et peuvent être la cible d'un autre fauve (proie et prédateur). Il existe également différents types d'obstacles bien que leurs effets soient similaires.

Outils utilisés

Pour l'outil de versionning, la première idée était GitHub, le plus évident pour tout le groupe. Cependant, Valentin a proposé de s'orienté sur GitLab car il l'avait déjà utilisé dans le cadre de son stage de DUT et que cet outil est très utilisé et demandé dans le milieu professionnel. Il est donc plus bénéfique pour le groupe d'avoir des connaissances avec GitLab.

L'environnement de développement intégré, nous avons communément choisi d'utiliser Code::Blocks car le compilateur est intégré dans l'IDE ce qui permet de gagner beaucoup de temps car il n'y a pas besoin de créer un Makefile et de le tenir à jour. De plus, il s'agit de l'IDE que chaque membre de l'équipe utilisait déjà.

Un autre avantage de Code::Blocks est qu'il intègre un outil de génération de documentation de code : DOXYGEN. Cet outil s'utilise de la même manière que la Javadoc, ce qui a permis à certains membres de l'équipe ayant déjà fait de la Javadoc de prendre en main rapidement cet outil et de l'expliquer aux autres.

Afin de réaliser des tests unitaires, nous nous sommes orientés vers DOCTEST étant l'outil que nous avons vu en cours et appris à utiliser. L'utilisation de DOCTEST à demander l'installation d'un autre compilateur car MINGW est difficilement compatible avec DOCTEST.

Draw.io est un site internet qui permet de réaliser des diagrammes UML. Ce site nous a permis de réaliser les diagrammes de classes pour notre projet. Concrètement, cela permet une visualisation des différentes classes nécessaires ainsi que de leurs liens entre elles.

Discord a permis à toute l'équipe d'échanger sur l'évolution du projet ou demander des conseils aux autres. Il était utilisé presque quotidiennement et permettait aussi à un membre de notifier les autres lorsqu'il avait « push » vers l'outil GitLab afin que nous puissions « pull » la nouvelle version sur nos machines pour avancer et éviter les conflits.

Répartitions des tâches

Les membres se sont réparti les tâches de la façon la plus équilibré possible entre quantité (de travail), volonté (si la personne préférait faire une autre tâche) et la difficulté (afin de prendre le niveau de chacun).

D'où la répartition des tâches suivante :

- Valentin MARTIN : réalisation du terrain de jeu représentant le plateau de jeu et participation à la réalisation du menu du jeu

- Antoine BOCH : réalisation des joueurs (1 type nécessaire mais ajout d'un 2ème type de joueur possible facilement), participation au développement de la boucle principale du jeu (la partie tour par tour) et participation à la réalisation des diagrammes UML

- Antoine FONTAINE : réalisation des fauves, participation au développement de la boucle principale du jeu et participation à la réalisation des diagrammes UML

- Téo BANNWARTH : réalisation des obstacles et participation à la réalisation du menu du jeu

Déroulement du projet

Le projet a débuté durant la semaine du 7 novembre 2022. Dès le début, on s'est concerté ensemble afin d'évoquer les différents aspects du jeu afin d'avoir déjà une idée vers laquelle s'orientée. Ces idées ont pris formes à travers d'une première version du diagramme de classe UML.

Durant la semaine du 28 novembre, Valentin a créé un premier « repository » sur GitHub pour le projet mais à la suite de diverses complications et après concertation commune, l'outil de versionning a été modifié pour partir sur GitLab. Il a fallu 1 journée pour paramétrer le nouveau dépôt sur GitLab afin que chacun des membres puissent réaliser les actions nécessaires de « clonage » du dépôt sur sa machine, de « push » c'est-à-dire d'envoyer les

fichiers vers le dépôt et de « pull » c'est-à-dire de récupérer les fichiers du dépôt dans le dossier cloné sur notre machine.

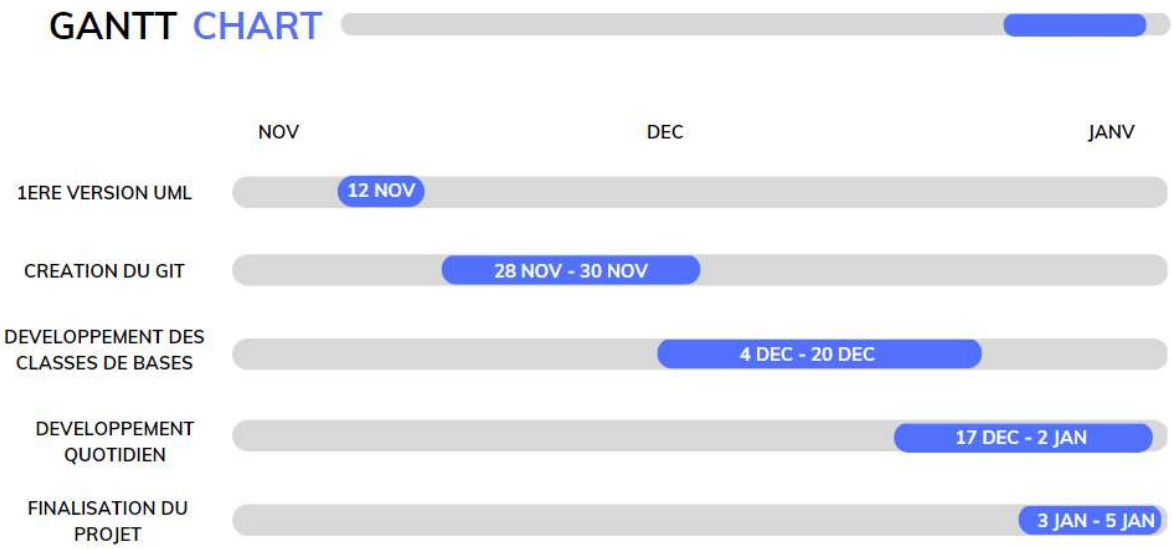
Ensuite, une période vide a pris place causée par de nombreuses évaluations.

Le développement a repris à partir du 4 décembre avec le début de développement des classes et de leurs méthodes par rapport au diagramme UML.

A partir de cette date, le développement avançait régulièrement afin de ne pas louper l'échéance du premier rendu le 20 décembre 2022. Ce premier rendu était un peu brouillon car l'équipe s'est rendu compte, la veille du rendu, qu'il fallait remanier le code afin de permettre une flexibilité totale et limiter la quantité de code à ajouter en cas de création d'un nouvel élément de jeu.

Enfin, durant les 2 semaines de vacances de noël, le développement avançait quotidiennement (à l'exception des jours de fêtes) comme en témoigne l'historique des « commits » sur GitLab. Cette période était assez difficile à gérer à cause des fêtes et du trajet pour certains afin de rejoindre leurs familles ce qui faisait perdre des journées entières d'avancement.

A la date du 4 janvier 2023, le projet en est au stade final, c'est-à-dire qu'il reste des tests à faire et quelques petites méthodes à développer mais tout sera finalisé d'ici la date de rendu, le vendredi 6 janvier à 14h.

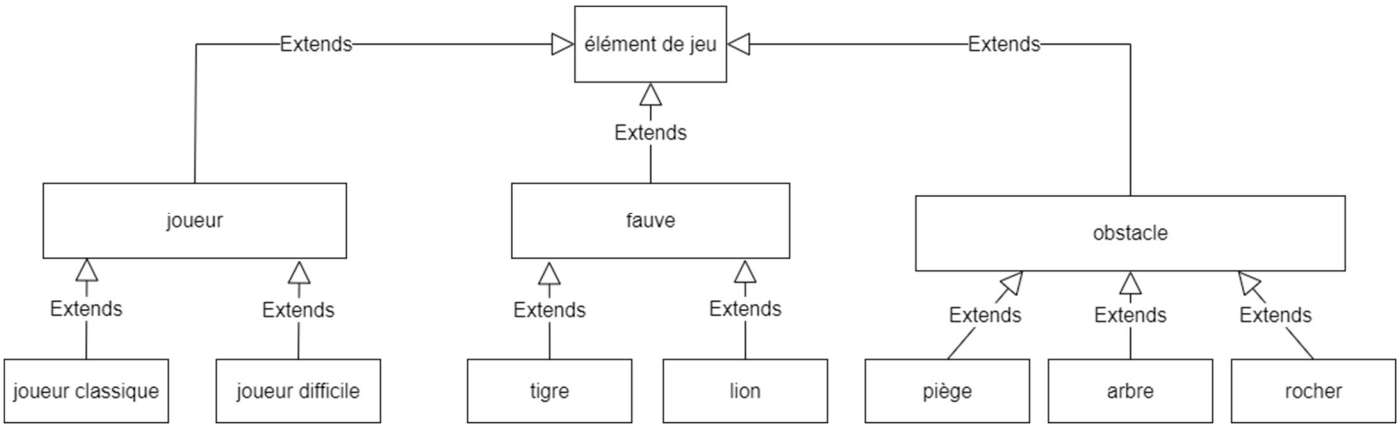


Le projet

Pertinence des outils enseignés

Tout au long de ce projet, de nombreuses notions vues durant les cours ont été utiles et réutilisées. La première notion utilisée est le principe d'héritage. En effet, nous avons remarqué qu'une certaine architecture se dégageait du côté des éléments de jeu. Les éléments peuvent être regroupés dans des types. Par exemple, les arbres, les pièges et les pierres sont des obstacles, c'est-à-dire que ce sont des éléments dont le but est de restreindre les mouvements du joueur et des fauves. De plus, un obstacle est fixe, c'est-à-dire qu'il ne bouge pas sur le terrain. De même, il existe différents types de fauves (tigres et lions) et différents types de joueurs (joueur classique et joueur difficile avec des mouvements restreint). Ce qui veut dire qu'il existe 3 types de bases : l'obstacle, le fauve et le joueur. Mais on remarque que ces 3 types sont en réalité des éléments de jeu, c'est-à-dire qu'ils sont des types dérivés d'un élément de jeu.

Cela donne l'architecture suivante :



Une notion élémentaire au bon fonctionnement de l'héritage est la notion de pointeurs dynamiques avec les unique_ptr. Le plateau de jeu est représenté par un tableau de 2 dimensions composé de pointeurs qui peuvent être nulles (nullptr) ou pointant vers un élément de jeu. En effet, avec l'héritage, afin de faciliter l'ajout de nouveaux types d'éléments de jeu, il était nécessaire de combiner ces 2 notions comme vues pendant les TD et TP (par exemple, lors de l'exercice de gestion des logements). Les pointeurs sont assez délicats à manipuler et étaient la source majeure des problèmes de compilations du code.

Pour compléter les 2 notions ci-dessus et faire que cela fonctionne, il faut utiliser le polymorphisme. Plus concrètement, pour pouvoir dire que le terrain peut contenir un élément de jeu, il faut qu'il comprenne qu'il puisse contenir n'importe quel type d'élément de jeu. C'est dans cette phase que le polymorphisme intervient. En utilisant la notion d'héritage et de pointeurs dynamiques, il est possible d'utiliser des pointeurs dynamiques de la façon

suivante : unique_ptr<gameElement>. Dans ce cas, le pointeur peut contenir n'importe quel type héritant d'un élément de jeu (gameElement). Tout ceci permet de résoudre le problème du plateau contenant différents types d'éléments de jeu en reprenant l'écriture précédente avec les pointeurs dynamiques.

Afin de faciliter la compréhension du projet et de sa répartition, il a été nécessaire de remanier le code en plusieurs classes traitant chacune d'une fonctionnalité principale du projet (par exemple, la classe « menu » qui s'occupe uniquement du menu, la classe « field » qui s'occupe uniquement de la création et gestion de terrain de jeu, …)

De plus, il est nécessaire de réaliser des tests unitaires pour vérifier le bon fonctionnement du code. Cela permet d'éviter de faire des tests manuellement qui prennent du temps et de remarquer rapidement et facilement si une fonction ne marche pas comme souhaitée. Ces tests permettent de s'assurer que le résultat obtenu par la fonction correspond au résultat attendu.

Enfin, l'utilisation de git a été vitale pour pouvoir travailler sainement sur ce projet en équipe. Il a servi à stocker les fichiers du projet permettant à chacun de pouvoir récupérer ces fichiers et de mettre à jour sa version du projet sur sa machine lorsque des modifications ont été réalisées. Cependant, il faut rester attentif avec cet outil car il peut rapidement y avoir des conflits entre les différentes versions du code.

Les fonctionnalités

Lors du lancement du jeu, l'utilisateur a le choix entre plusieurs actions (jouer, charger un terrain, créer un terrain, modifier un terrain). Toutes ces fonctionnalités fonctionnent parfaitement.

Lorsque l'utilisateur veut joueur, il a le choix entre les différentes cartes présentes dans le jeu. A l'origine, seule une carte par défaut est fournie avec le jeu mais le joueur à la possibilité de créer autant de carte qu'il ne le souhaite.

Lorsqu'il créé une carte, il peut choisir, pour chacun des éléments de jeu, la quantité de cet élément qu'il souhaite disposer sur le terrain. Une fois tous les éléments correctement placés, le joueur revient sur le menu.

Lorsque l'utilisateur lance une partie, il peut se mouvoir partout autour de lui à condition qu'il ne se déplace pas de plus d'une case et qu'il ne se déplace pas sur un élément de jeu. Les fauves essayent de se rapprocher tour par tour du joueur en fonction de sa position. La partie se termine lorsque le joueur a été mangé ou s'il est parvenu à faire tomber tous les fauves dans les pièges. L'utilisateur a alors accès à quelques statistiques sur sa partie.

Lorsque l'utilisateur modifie un terrain, il doit premièrement choisir le terrain à modifier. Ensuite, il peut modifier autant qu'il le souhaite le terrain. Pour cela, il lui est demandé de choisir la case qu'il souhaite modifier. Si la case est vide, il lui est proposé de rajouter un élément de son choix, sinon, si la case contient déjà un élément de jeu, il lui est proposé de le supprimer. Enfin, lorsque l'utilisateur est satisfait de ses modifications, il peut quitter afin de revenir au menu. En quittant les modifications, le terrain est automatiquement enregistré par-dessus l'ancienne version de celui-ci.

Les difficultés

La première difficulté était tout simplement de se mettre d'accord tous ensemble sur la structure finale du projet, la décomposition en plusieurs et la représentation du terrain de jeu. La solution était de discuté ensemble et de voir en profondeur comment résoudre les différents problèmes avant de se lancer dans le développement. Par exemple, au départ nous sommes partis sur l'idée de se servir d'une classe « box » qui représente une case du jeu mais nous avons finalement préféré un tableau 2 dimensions contenant des pointeurs dynamiques au lieu d'un tableau 2 dimensions contenant des « box » qui elles-mêmes contiendraient chacune un pointeur dynamique.

Les principales difficultés résidaient dans l'utilisation du terrain de jeu représenté par un tableau de 2 dimensions contenant des pointeurs dynamiques. La manipulation de ce tableau est assez complexe notamment avec le polymorphisme avec gameElement. Une erreur récurrente était « use of deleted function » notre terrain est déclaré de la façon suivante :

vector<vector<unique_ptr<gameElement>>> d_field;

Il était assez difficile de récupérer les éléments du terrain ou de les modifier.

Enfin, nous avons eu un problème avec les mouvements des fauves. En effet, lorsqu'un fauve se trouvait en bordure de terrain, le code plantait retournant un code d'erreur et s'arrêtant. Après avoir passé plusieurs dizaines de minutes à comprendre la raison de ce problème, c'est finalement un des tests unitaires qui a permis de se rendre compte de l'oubli d'un « -1 » après un terrain.size(). En effet, le terrain faisait une taille de 8 par 8, la méthode size() renvoyait 8 mais pour parcourir le terrain, il faut commencer à 0 et aller jusqu'à 7 soit size() – 1.

Conclusion

Pour chacun des membres ayant participé à la réalisation de ce projet, cela leur a permis de développer leurs connaissances en tests unitaires et en qualité de programmation. Ce projet a permis de démontrer qu'il est nécessaire de penser à son code avant de se lancer dedans et devoir tout modifier au premier problème. Les tests unitaires ont permis de débugger le code à un moment crucial. Il a également permis d'accroître considérablement nos connaissances avec le langage C++ et surtout avec les pointeurs dynamiques. En effet, le fonctionnement et l'intérêt des pointeurs nous étaient encore flous.

Enfin, ce projet nous a permis de mettre en place une vraie répartition des tâches ce qui a grandement facilité le travail en équipe. Cela se démontre par le fait d'avoir réussi à terminer le projet et d'avoir pu rapidement rebondir suite à l'abandon du projet d'un membre de l'équipe. Le travail d'équipe peut paraitre simple mais il doit être bien coordonnée pour pouvoir être réellement efficace et permettre un gain de temps sur un projet.