Rapport projet de programmation avancée

22/03/2021 - 06/05/2021

Evan JANNOT Corentin LUCAS

ENSC - 1A - 2020/2021

Sommaire

Programme

Classes

Au sein de notre projet, les classes sont regroupées dans différents dossiers. Ces dossiers permettent une meilleure organisation des fichiers au sein de notre solution, car ces derniers sont nombreux. Nous expliquerons ci-dessous les principales classes au sein de chaque dossier. Pour ce qui est du fonctionnement plus détaillé, le code est entièrement commenté et disponible sur GitHub.

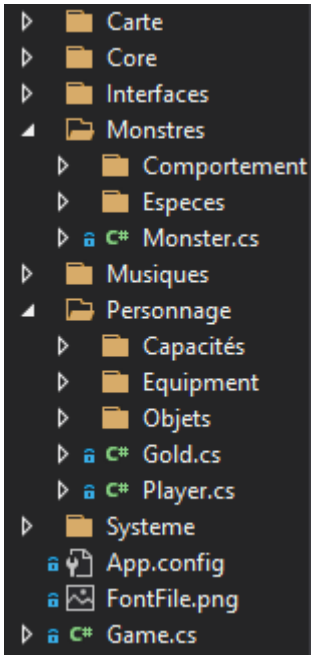

figure 1. Arborescence de nos fichiers

- *Game.cs* : C'est le fichier principal, celui contenant le main et qui permet le démarrage du jeu et l'initialisation de la plupart des paramètres. C'est à partir de ce fichier que les consoles sont créés et que le rendu visuel est effectué.
- **Carte**
  - *DungeonMap.cs* : Cette classe est l'une des plus importantes du jeu. Elle permet la gestion de la carte. En effet, elle contient des méthodes permettant de placer des monstres, placer le joueur, déplacer des acteurs, récupérer et placer des trésors, gérer l'état des portes et elle définit l'apparence des tuiles de jeu.
  - *MapGenerator.cs* : Cette classe va de pair avec la précédente. Celle-ci permet la création de la map en créant les différentes salles, les couloirs les reliant, les portes, les escaliers ainsi que les monstres et les trésors qui seront ensuite placés par DungeonMap.cs.
- **Core**
  - *Actor.cs* : Cette classe est la classe mère de celle du joueur et des monstres. Ainsi on y définit les différentes caractéristiques qui seront communes aux classes filles comme l'équipement ou les statistiques.
- **Interfaces** : Dossier contenant toutes les interfaces utilisées dans le jeu.

- **Monstres**
    - *Monster.cs* : Classe mère des différents types de monstres. On y définit la méthode virtuelle permettant d'afficher les monstres dans l'écran des statistiques et celle permettant aux monstres d'effectuer des actions. De plus, on retrouve la fonction permettant à certains types de monstres de se cloner.
    - **Comportement** : Dossier contenant toutes les classes correspondantes aux comportements des différents monstres.
    - **Especes** : Dossier contenant toutes les classes des monstres. Il y en a 12 au total. Chaque monstre possède ses propres caractéristiques et apparait dans un monde précis. Il existe différents types de monstres et chaque type a son comportement.
- **Musiques** : Dossier comportant les différentes musiques utilisées au sein de notre jeu. Chaque monde possède sa musique et les boss ont aussi une musique qui leur ait propre.
- **Personnage**
    - *Player.cs* : Permet la création du joueur. Cette classe est fille de la classe Actor.cs. De ce fait le joueur a de l'équipement, mais on lui définit aussi la possibilité de porter 4 objets et d'apprendre 4 capacités. En plus de cela il peut gagner de l'expérience et monter de niveau.
    - **Capacités** : Contient les deux classes ci-dessous ainsi que les classes de tous les sorts
        - *Ability.cs* : Classe définissant les sorts. Elle permet lancer les sorts et de ramasser ces derniers.
        - *AbilityGenerator.cs* : Cette classe permet de générer les sorts présents sur la map. Chaque sort a une probabilité plus ou moins élevée d'apparaître.
    - **Equipment** : Contient les deux classes ci-dessous ainsi que les classes des différentes pièces d'équipement
        - *Equipment.cs* : Permet de créer un équipement et de les ramasser.
        - *EquipmentGenerator.cs* : Cette classe génère l'équipement de la même manière qu'AbilityGenerator.cs génère les sorts à l'exception qu'ici, en fonction du monde où l'on se retrouve, le type d'équipement est différent.
    - **Objets** : Contient les deux classes ci-dessous ainsi que les classes de tous les objets
        - *Item.cs* : Classe qui définit les objets et permet de les utiliser et de les ramasser.
        - *ItemGenerator.cs* : Fonctionne de la même manière qu'AbilityGenerator.cs, mais pour les objets.
- **Systeme**
    - *ActorGenerator.cs* : Permet la création des acteurs. Cette classe va créer les monstres à chaque étage en fonction du monde et va en début de partie, créer le joueur. Elle permet aussi de réinitialiser ce dernier au début d'une nouvelle partie.
    - *CommandSystem.cs* : Classe qui gère les commandes du joueur. Elle permet de se déplacer, d'attaquer, de lancer les sorts et d'utiliser les objets. En plus de cela, elle déplace aussi les monstres et leur permet d'attaquer.

- *Pool.cs* : Classe permettant la création de listes pondérées. C'est à l'aide de cette dernière que nous pouvons générer des monstres, de l'équipement ou des objets ayant plus ou moins de probabilité d'être généré.
- *SchedulingSystem.cs* : Permet de gérer le système de temps et d'y ajouter ou retirer des acteurs.
- *TargetingSystem.cs* : Cette classe est utilisée par les sorts et permet de sélectionner une cible, une zone ou encore un segment afin de lancer un sort.

Ces classes sont les plus importantes au sein du projet et sont essentiels au fonctionnement de la majorité des fonctionnalités. Elles sont pour la plupart issues du tutoriel proposé par RogueSharp mais ont été commentées et certaines remaniées. De plus nous avons également ajoutées des classes qui n'étaient pas présentes à l'origine et modifié le fonctionnement de certaines classes qui présentaient des bugs. Mais nous verrons plus tard les différences entre notre jeu et celui proposé par le tutoriel.

Vous retrouverez à la page suivante les UML de nos différentes classes représentant les liens entre la classe mère et ses classes filles ainsi que, les méthodes et paramètres de chacune.

Si vous souhaitez avoir un aperçu encore plus large du lien entre toutes nos classes et les différentes interfaces, un diagramme des classes est présent au sein de notre solution sous le nom de "Diagramme.cd". Ce diagramme est aussi disponible au format .png sous le nom "Diagramme.png".

UML

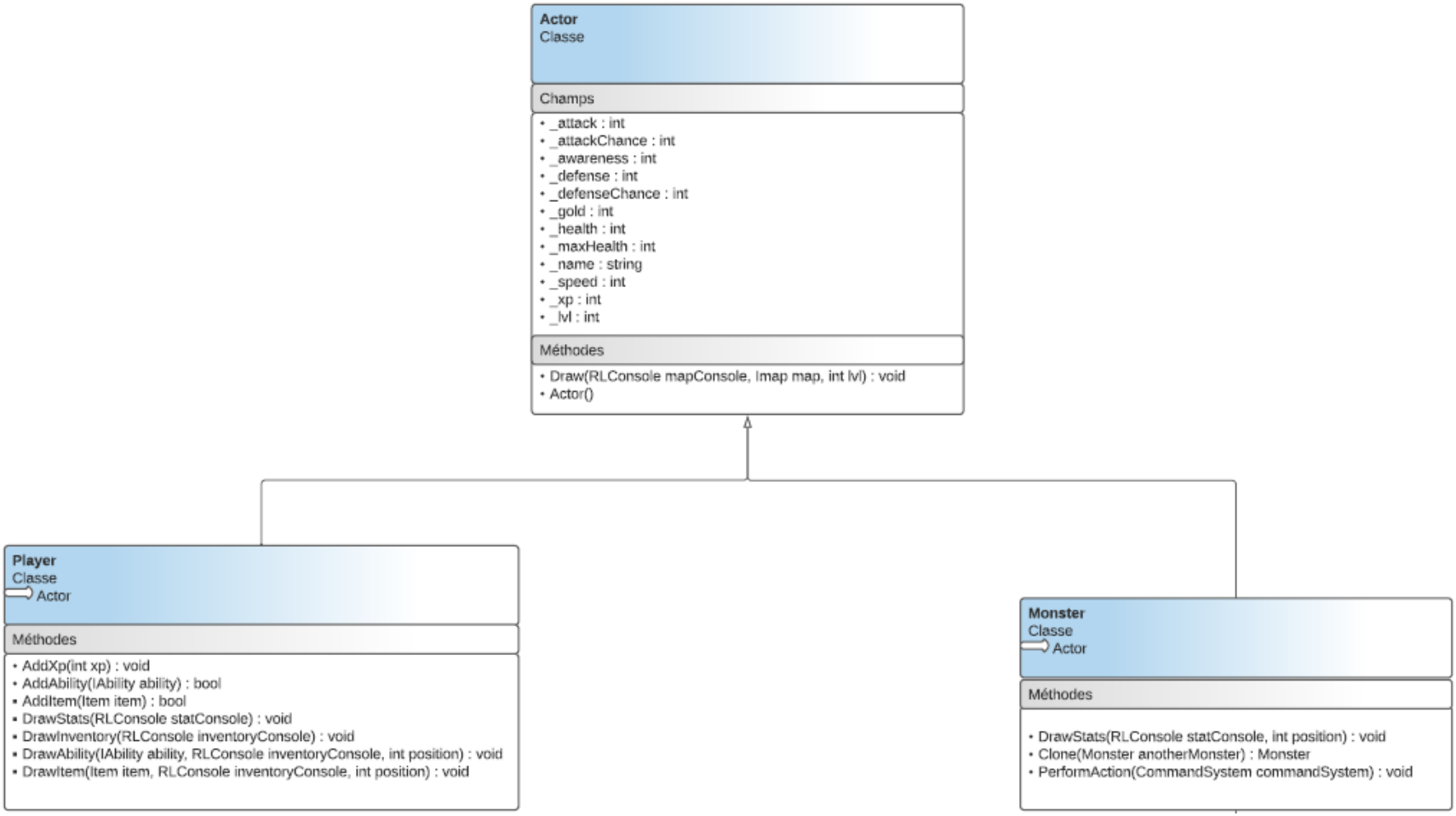

figure 2. UML de la classe mère Actor

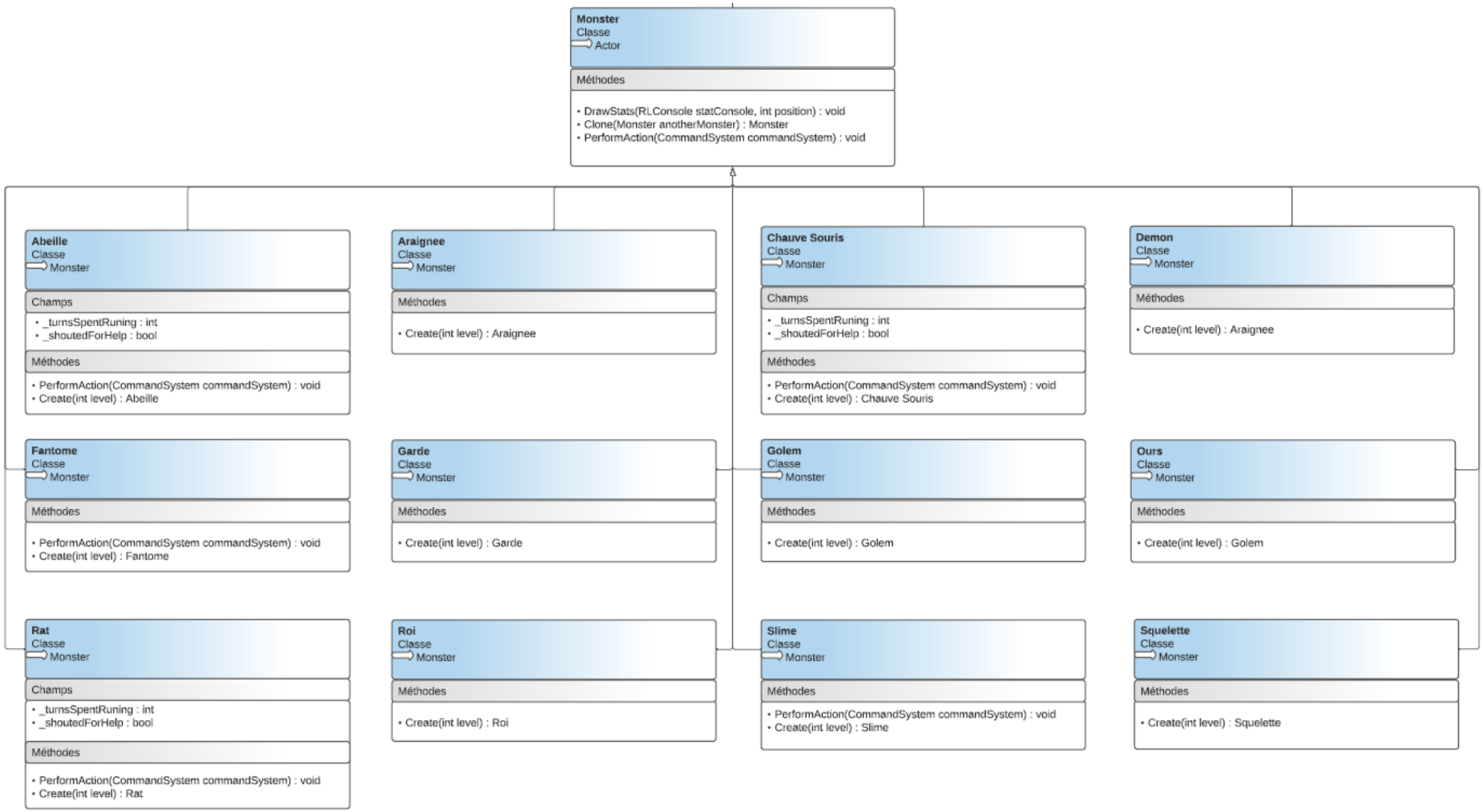

figure 3. UML de la classe mère Monster (classe fille de Actor)

figure 4. UML de la classe mère Item

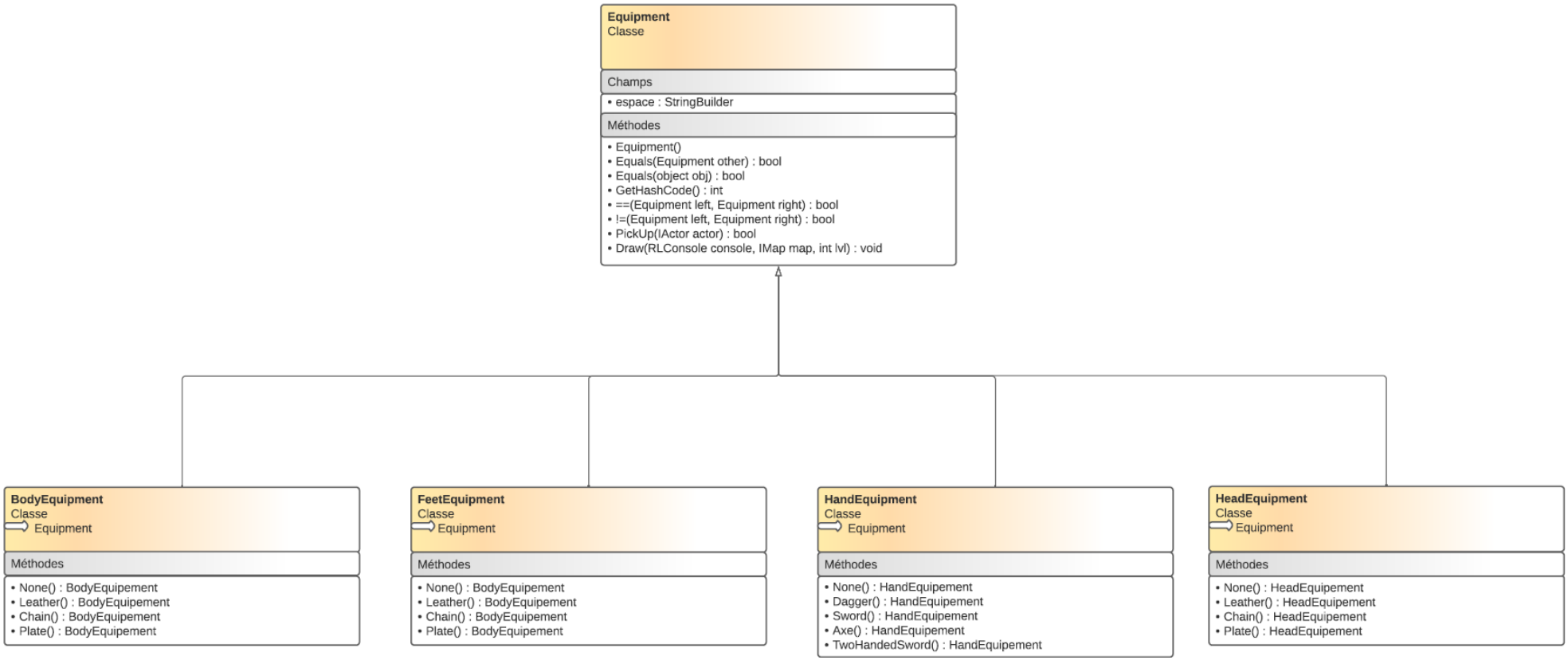

figure 5. UML de la classe mère Equipment

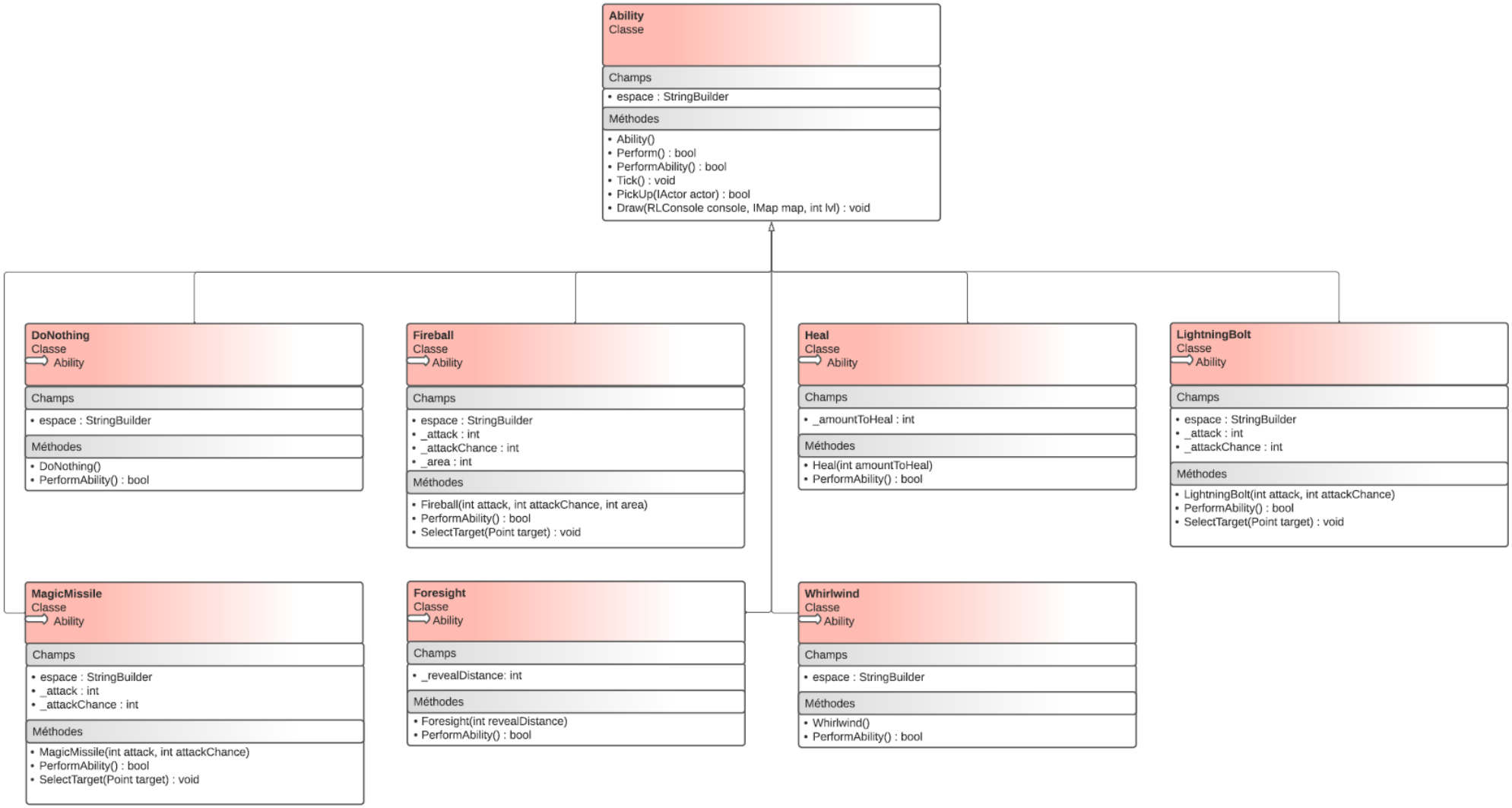

figure 6. UML de la classe mère Ability

Contenu ajouté par rapport au cahier des charges

Les principales caractéristiques à intégrer étaient :

- Exploration d'un univers de jeu au fur et à mesure dans le but d'atteindre un objectif donné
- Environnement textuel à 2 dimensions.
- Jeu au tour par tour
- Ressources pour survivre
- Permadeath

Ces fonctionnalités sont toutes présentes au sein de notre jeu.

Nous avons en plus intégré certaines des fonctionnalités bonus du sujet :

- Génération procédurale
- Interactions avec des PNJ et combats
- Système d'inventaire
- Interaction de complexité variée avec l'environnement

Et nous avons inclus des idées originales :

- Système d'expérience et de niveau
- Système de choix de classe
- Boss
- Aventure post-fin infinie

Différences avec le tutoriel

Le tutoriel nous a permis de comprendre le fonctionnement de la bibliothèque RogueSharp et nous a offert une base pour la création de notre jeu. Il nous laisse avec une génération de monde procédural infini possédant 3 monstres ayant chacun un comportement où le joueur évolue en utilisant des sorts, des objets et en portant de l'équipement. Nous avons alors apporté notre touche personnelle et nos fonctionnalités à cette base tout en l'assimilant (l'ensemble du code est commenté).

La première différence entre notre jeu et ce qui est proposé par le tutoriel est l'apparence. En effet, le système de base utilise une image de 128x128 pixels contenant l'alphabet ASCII. Nous avons souhaité changer cette apparence afin d'avoir quelque chose de plus plaisant à voir et de plus personnel. Pour se faire, il nous a fallu comprendre comment le jeu se servait de l'image donnée en entrée pour afficher les éléments.

L'image source doit être composée de 16 lignes et de 16 colonnes. Cela forme une sorte de tableau où chaque case est numérotée de 0 à 255 en partant du coin haut-gauche vers le coin bas-droit. Lorsque l'image est importée, il faut préciser le nombre de pixels d'une case (par défaut 8). Ainsi le programme va savoir que tous les 8 pixels, il change de case. Il suffit alors de remplacer sur l'image de base une case par l'image que l'on souhaite puis pour afficher cette dernière il faut soit entrer le caractère du code ASCII correspondant à l'emplacement où se situe l'image, soit entrer directement le numéro de l'emplacement précédé par "(char)". Pour plus de simplicité nous avons placé les lettres de l'alphabet à leurs emplacements dans le code ASCII pour facilement afficher du texte en jeu.

figure 7. Image du tutoriel (à gauche) et notre image (à droite)

Nos images font une taille de 16 pixels ce qui permet d'afficher plus de détails et d'avoir un jeu visuellement plus agréable. La majorité des icônes utilisées sont issues du site Kenney • Home qui propose des packs d'icônes pour différents types de jeux. Pour les autres nous les avons créées nous même. Nous avons d'ailleurs expliqué à plusieurs groupes comment procéder afin de changer les tuiles de jeu pour le personnaliser.

La deuxième différence majeure avec le jeu de base est que nous avons constitué des mondes avec un marchand et un boss. En effet, dans le tutoriel, chaque étage correspond à un niveau classique. Ici chaque étage correspond à un niveau au sein d'un monde, chaque monde étant composé de 6 étage, on y retrouve 4 étages classiques, un étage comportant un marchand et un étage comportant un boss.
Ainsi nous avons dû penser à un système d'achat qui n'est pas présent dans le tutoriel. De plus, aucun PNJ n'était créé au cours du tuto. Pour ajouter cette fonctionnalité, nous avons tout d'abord créé un étage composé d'une unique salle carré au sein de laquelle nous avons placé une tuile représentant un PNJ et nous affichons un message dans la boîte textuelle. Devant ce PNJ, nous plaçons un sort, une pièce d'équipement et un objet. Pour le système d'achat, lorsque le joueur marche sur le trésor qu'il souhaite récupérer le jeu vérifie s'il dispose de la quantité d'argent suffisante, si c'est le cas il lui retire et lui donne l'objet, sinon l'objet n'est pas ramassé.

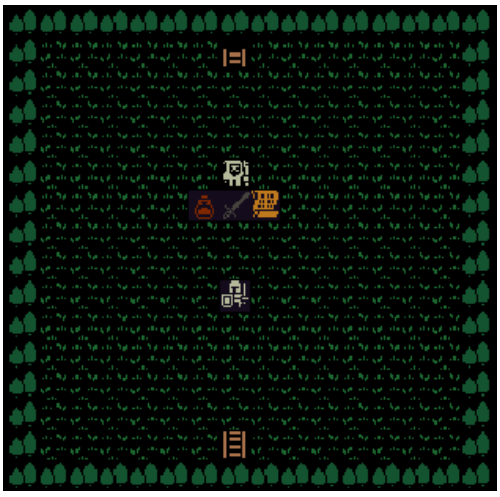

figure 8. Salle du marchand

Pour ce qui est des boss, nous avons mis en place un système détectant si le boss a été éliminé et lorsque c'est le cas, les portes de la salle s'ouvrent. Ainsi, l'étage comportant le boss est constitué de 2 salles, dans la première, nous plaçons le monstre et dans la dernière l'escalier pour changer de monde. Ainsi tant que le monstre n'est pas mort, il n'est pas possible d'avancer.
De plus, chaque monde possède sa propre apparence et son bestiaire. La difficulté évolue également entre les mondes et les armures présentes ont de meilleures statistiques.

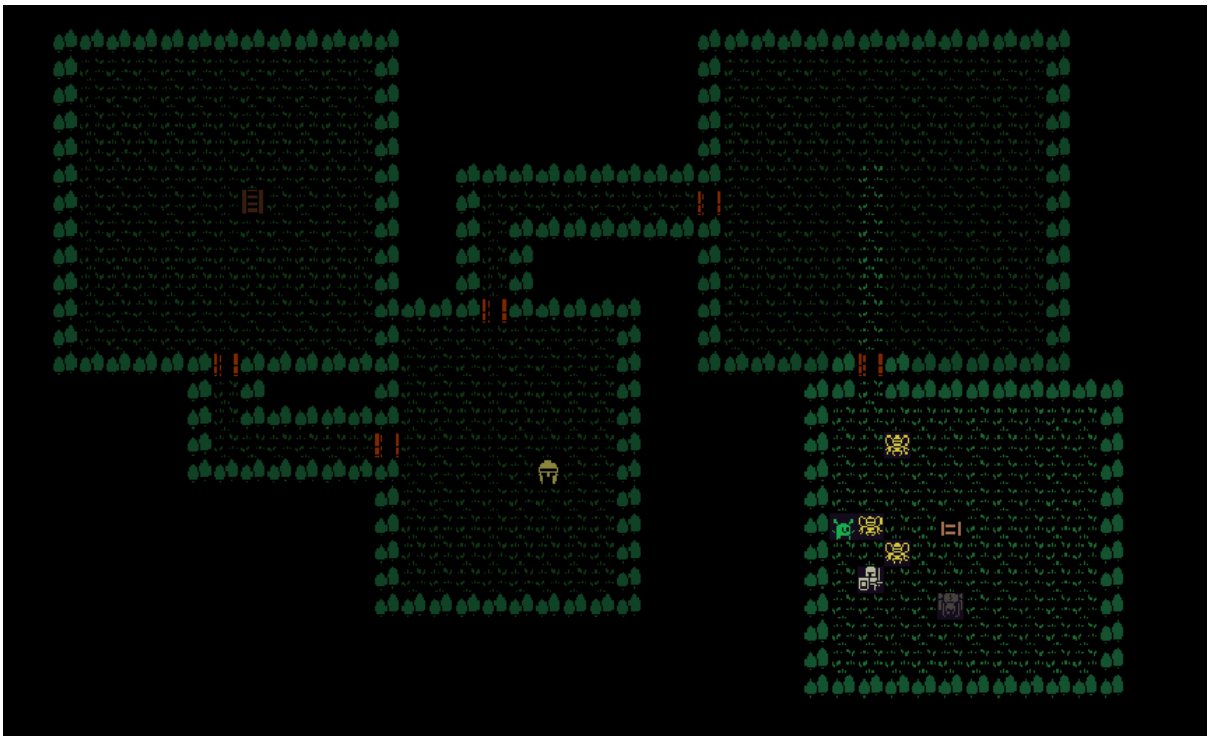

figure 9. Exemple d'étage du monde 1

figure 10. Exemple d’étage du monde 2

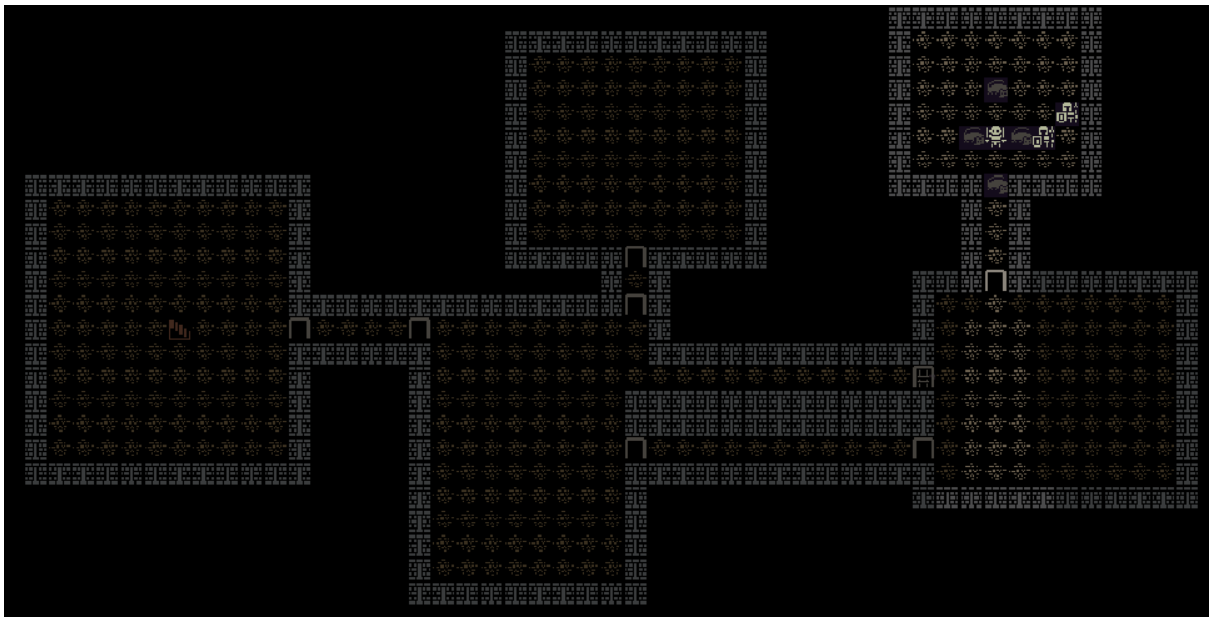

figure 11. Exemple d’étage du monde 3

Ensuite, nous avons mis en place un écran de pause ainsi qu'une possibilité de relancer une partie lorsque le joueur meurt. Pour l'écran de pause, si le joueur appuie sur ECHAP au cours de la partie, la carte est remplacée par le texte du menu pause. Il suffit d'appuyer sur une touche d'action du jeu pour reprendre la partie.
Pour ce qui est du relancement d'une partie à la mort, nous avons fait en sorte que lorsque le joueur perd et que son score est affiché, il puisse appuyer soit sur SUPPR pour quitter soit sur O pour relancer. En appuyant sur O, le perso est réinitialisé ainsi que les autres paramètres et variables évoluant au cours du jeu. Le joueur peut alors re entrer son nom et re choisir une classe au sein de la console classique puis au lancement de la nouvelle partie l'ancienne fenêtre de jeu se ferme et une nouvelle se lance.

Nous avons également implémenté des détails visuels améliorant l'expérience de jeu. En effet, initialement, l'équipement a la même apparence peu importe la partie d'armure ou le matériau. Nous avons alors fait en sorte que chaque pièce est une apparence différente et une couleur dépendant de sa rareté afin d'éviter à un joueur possédant un casque rare de le remplacer par un casque commun en pensant qu'il s'agit peut-être d'une autre pièce d'armure. Nous avons appliqué le même principe aux objets ce qui évite par exemple d'avoir en 4 exemplaires le même objet qu'on ne peut pas utiliser et ainsi bloquer l'inventaire (exemple : amélioration d'armure alors qu'on a atteint l'armure max).

Au niveau du joueur, nous avons changé plusieurs choses. Premièrement il est possible de définir un nom et ce dernier n'est plus imposé comme dans le tutoriel. Ensuite, il est possible de monter de niveau en accumulant de l'expérience. Chaque niveau demande 100 points de plus d'expérience et augmente la vie de 10. Enfin, en début de partie il est possible de choisir entre 4 classes. Chaque classe va apporter un bonus à certaines statistiques et un malus à d'autres afin de garder un jeu équilibré, mais proposant différentes approches. Afin de distinguer ces classes, l'apparence du personnage sera différente.

De plus, de la musique a été intégrée à notre jeu. En effet, ce qui fait le dynamisme et le charme de nombreux jeux est l'ambiance sonore. Ainsi, nous avons sélectionné des musiques issues de la bande originale du jeu *The Legend Of Zelda : A Link To The Past* pour accompagner le joueur au sein des différents mondes. Pour ce qui est des musiques de boss, nous avons opté pour des musiques issues de l'animé JoJo's Bizarre Adventure particulièrement dynamiques. Enfin, une fois le jeu terminé, le joueur pourra parcourir les niveaux bonus en écoutant des musiques issues des jeux *Celeste* et *Undertale*.

Pour la narration, il n'y avait aucune méthode proposée dans le tutoriel. Nous avons donc découpé cette dernière en deux parties. La première se situe dans la console classique juste avant d'entrer son nom et de choisir sa classe et, permet au joueur d'avoir l'intrigue principale ainsi que s'il le souhaite une explication des commandes. Ensuite nous avons créé deux mondes spéciaux dans lesquels nous ne générons pas de monstres et d'objets. Ces mondes n'ont qu'un étage composé que d'une pièce. Au centre de celle-ci se situe un vieux monsieur qui expose quelques éléments supplémentaires au joueur. Le premier monde est situé en tout début de jeu, c'est ici que le joueur commence et enfin le deuxième est situé après avoir terminé l'aventure de base et permet d'introduire le mode infini.

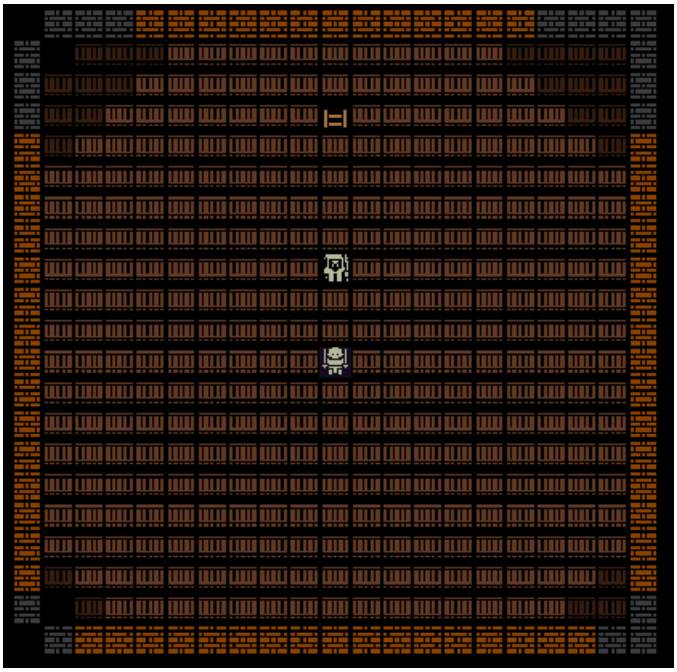

figure 12. Monde de début du jeu

Enfin, une fois les 3 mondes principaux terminés, nous avons mis en place la possibilité de continuer de jouer. Les mondes situés après la fin du jeu ont tous la même apparence et sont constitué selon le même schéma que les autres. Au sein de ces derniers, il est possible de retrouver tous les monstres que le joueur a pu affronter au cours de sa partie. De même, pour ce qui est des boss, le joueur affrontera dans chaque monde un des 3 boss rencontré de manière aléatoire.

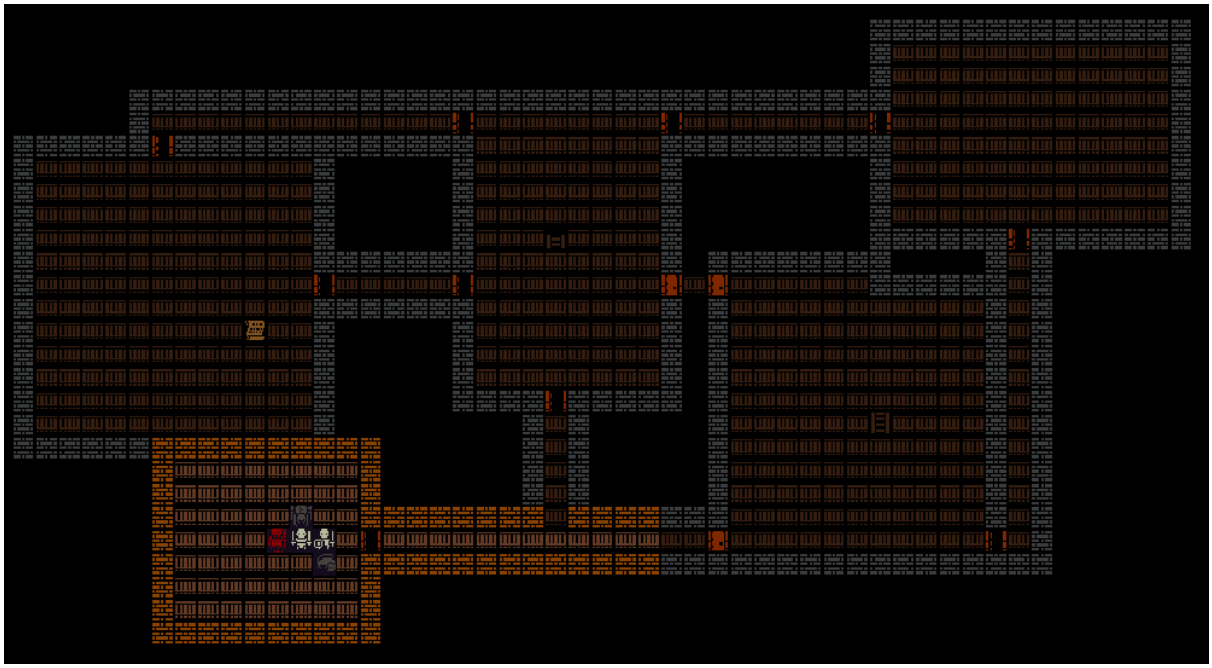

figure 13. Exemple d'étage du monde post-fin

Manuel

Afin de fournir à l'utilisateur ou à un individu souhaitant connaître notre jeu des informations sur l'univers et le gameplay, nous vous proposons un manuel de présentation du jeu.

Univers et scénario

Notre jeu prend place dans l'univers de la mythologie nordique. Le joueur incarne un guerrier viking ayant trouvé la mort en combat. La tradition veut que les plus valeureux des guerriers accèdent au Valhalla. Situé au sein même du royaume des dieux, c'est un véritable paradis. Malheureusement pour notre personnage, son accès a été bloqué par un ancien Roi déchu n'ayant pas pu y accéder. En effet, dans un dernier élan de rage, il effectua un sort interdit réveillant légions de monstres et bloquant l'accès à quiconque souhaiterait y accéder.

C'est donc dans ce contexte que le joueur évoluera. À son réveil après la mort, le joueur est accueilli par une voix apaisante. Cette dernière lui explique alors la situation et lui demande s'il sait se servir de son corps dans le monde des défunts. Ce prétexte nous permet d'expliquer les commandes aux joueurs novices. Après cela, la voix demande au joueur son nom et la classe qu'il fut durant sa vie sur terre.

figure 14. Classes disponibles (Guerrier, Berserk, Brute et éclaireur)

Une fois que le joueur a pris connaissance des commandes, a renseigné son nom et a sélectionné une classe, le jeu se lance dans une pièce en bois avec un personnage mystérieux au milieu (**cf. figure 11**). Ce dernier se trouve être la voix qui a précédemment parlé au joueur. Il lui détaille alors le parcours qu'il devra effectuer afin de surmonter l'épreuve qui l'attend.

figure 15. Le mystérieux personnage…

Au cours de son aventure le joueur devra traverser 3 lieux, ces derniers sont implémentés sous forme de mondes. Le premier est la forêt (**cf. figure 8**). Au sein de celle-ci se trouve des monstres tels que des araignées géantes, des nuées d'abeilles ou encore des slimes qui en grand nombre peuvent s'avérer des plus redoutables. Une fois celle-ci traverser, le joueur arrivera dans une clairière où se trouve l'homme qui l'a accueilli en début de partie. Ce dernier lui propose en échange d'argent amassé sur les ennemis d'acheter un objet, une pièce d'équipement ou un sort afin de se préparer à la suite. En quittant la clairière, il pourra accéder à une caverne rejoignant le château. Cependant, il devra abattre le terrible ours qui en garde l'entrée.

figure 16. Bestiaire de la forêt

Une fois le joueur arrivé dans la caverne (**cf. figure 9**), il devra affronter les spectres des guerriers ayant échoués à atteindre le château. Ces derniers sont présents sous la forme de fantômes et de squelettes et tentent d'empêcher le joueur d'avancer dans sa quête du fait de l'emprise du roi maudit. Il se trouve aussi que des hordes de chauves-souris habitent ses cavernes et peuvent s'avérer être un fléau lorsqu'elles sont en trop grand nombre. Mais une fois le bout de la caverne, le vieil homme lui propose encore une fois de lui fournir de l'équipement en échange d'argent. Il est cependant un peu radin et lui demande plus que la fois précédente… Une fois la pièce où se trouve le vieillard passée et l'accès au château à portée de main, le joueur devra affronter dans un duel plein de violence le Golem gardien de l'entrée.

figure 17. Bestiaire de la caverne

Enfin, une fois le joueur dans le château (**cf. figure 10**), ce dernier devra faire face à la garde personnelle du Roi fidèle à ce dernier même dans sa folie. À leurs côtés se trouve des démons invoqués par le Roi tout droit venu de Helheim l'enfer nordique. Ces deux types d'ennemis sont redoutables et donneront du fil à retordre au joueur. Et enfin le château se trouve infesté de rats étant donné que ce dernier n'est pas entretenu depuis des années et est rempli de monstres, qui serait assez fou pour y faire du ménage ? L'étranger aurait peut-être pu étant donné qu'il était encore là, à proposer de l'équipement à un prix encore plus élevé ! Si le joueur continue d'avancer et parvient au bout du château, il rencontrera le Roi et un affrontement épique s'engagera, qui en sortira vainqueur ?

figure 18. Bestiaire du château

Si le joueur arrive au bout de sa quête, l'homme mystérieux l'attendra dans une salle similaire à celle de départ. L'accès au Valhalla est maintenant rétabli et le joueur peut enfin y accéder. Cependant, son aventure est loin d'être terminée ! Afin de prouver à quel point ce dernier est valeureux, il a alors accès à un monde infini (**cf. figure 12**). Ce monde est redoutable et contient tous les monstres que le joueur a pu rencontrer au cours de son aventure. Jusqu'où parviendra-t-il à se rendre ? Mais peu importe à quel point il avancera dans ce monde, il aura son accès au Valhalla garanti et pourra être fier de son parcours !

Si le joueur meurt au cours de sa quête, l'aventure s'arrête ici pour lui. Mais l'histoire, elle, continue ! En effet, si ce n'est pas lui qui vaincra le Roi, un autre héro arrivera bien un jour pour tenter sa chance. C'est dans ce contexte que le joueur peut relancer une partie après sa mort avec un nouveau personnage. Cependant, bien des années peuvent être passées et le monde tel qu'il l'était n'est plus. La topologie des niveaux a évolué et le joueur devra faire face à tous les dangers.

Contrôles et ressources

Déplacements

Pour se déplacer, il suffit d'utiliser les flèches directionnelles du clavier. Il est possible de se déplacer suivant 4 directions (haut, bas, gauche, droite).
Les portes s'ouvrent automatiquement lorsque le joueur marche sur ces dernières à l'exception des portes des salles de boss qui s'ouvrent uniquement si ce dernier est mort.

Interactions avec l'environnement

Pour changer d'étage, il suffit d'appuyer sur la barre d'espace en étant placé sur la case d'un escalier ou d'une échelle. Il est cependant impossible de remonter vers un étage déjà effectué veillez donc à avoir bien exploré la pièce, car vous ne pourrez pas y retourner plus tard !
Pour acheter un objet auprès du marchand, il suffit au joueur d'avoir en sa position suffisamment d'argent et de se placer sur la case contenant l'objet qu'il souhaite acheter. Veuillez-être sûr de ce que vous souhaitez, car le marchand ne procède à aucun remboursement !

Combats

Pour attaquer, rien de plus simple. Il suffit d'avancer vers un ennemi lorsque l'on se situe à côté de lui pour le frapper. Nous vous conseillons de rester appuyer sur la touche directionnelle en direction de l'ennemi afin que les combats se déroulent de manière plus rapide, car en appuyant de manière répétée, en fonction du type d'ennemi, cela peut s'avérer long.

Utilisation des objets, capacités

Au sein de sa quête, le joueur pourra rencontrer différents objets et différentes capacités. Chaque capacité a la même apparence, un vieux parchemin composé de gros caractères. Mais, chaque objet a une icône différente. Le joueur peut connaître au maximum 4 capacités et ne peut pas en oublier. Il peut porter 4 objets et une fois leur nombre d'utilisations dépassé, ces derniers sont détruits.

figure 19. Icône des sorts et des objets

Voici le détail des effets des objets ainsi que des sorts :

- **Sorts :**
    - Boule de feu : Envoie une boule de feu sur une zone sélectionnée par le joueur, effectue des dégâts à tous les ennemis présents à l'intérieur.

- Clairvoyance : Permet d'avoir un aperçu des alentours sans avoir besoin de s'y rendre !
- Soin : Restaure une dizaine de points de vies au joueur.
- Éclair : Envoie un éclair dévastateur le long d'une ligne sélectionnée par le joueur, effecteur des dégâts à tous les ennemis s'y trouvant.
- Missile magique : Envoie une boule d'énergie magique à un ennemi ciblé par le joueur.

- **Objets :**
  - Armure+ : Augmente l'armure totale du joueur de 1 point.
  - Destruction : Baguette magique permettant de détruire tout ce qui se trouve dans une direction au hasard, même les murs !
  - Potion de soin : Potion restaurant une quinzaine de points de vies au joueur.
  - Carte magique : Carte de la zone permettant de savoir tout ce qu'il se trouve à cet étage. Les ennemis sont marqués comme des cases noires.
  - Téléportation : Permet de se téléporter aléatoirement dans une salle de l'étage.
  - Aiguisage : Permet d'aiguiser l'arme du joueur et d'augmenter son tranchant. Cela augmente le pourcentage de chances de toucher.

Afin d'utiliser des objets ou une capacité, il vous suffit d'appuyer sur la touche liée à l'emplacement sur lequel se trouve l'objet ou le sort que vous souhaitez utiliser. Certaines capacités peuvent demander de sélectionner une cible ou une zone à frapper, vous pouvez déplacer cette zone en vous servant des flèches directionnelles et appuyer sur entrée pour confirmer votre sélection.

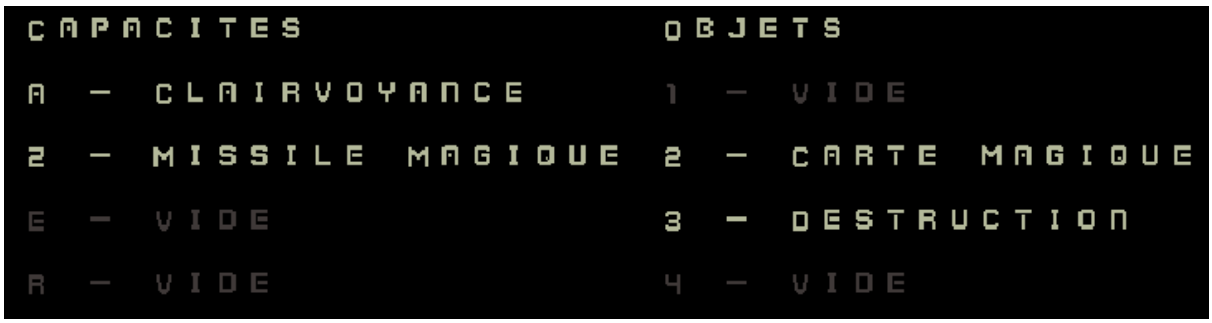

figure 20. Menu des capacités et objets

Équipement

L'équipement est ramassé par le joueur de manière automatique et se décompose en quatre parties. La première est le casque représenté par une icône de casque sur le sol, la deuxième le plastron qui est représentée par une icône de la partie d'armure en question, la troisième est l'arme qui va être représentée par une épée au sol et enfin les jambières représentées par une icône de bottes.

Attention, il existe différents niveaux de protection. En effet, les pièces d'armures peuvent être jaunes, bleus ou rouges. Une armure jaune correspond à une armure en cuir et offre une protection faible. Une armure bleue est une armure en maille et offre une meilleure protection. Cependant, la meilleure armure est la rouge et représente une armure constituée de plaques.

figure 21. Icônes d'équipement

Mise en pause et quitter

Il est possible de mettre en pause le jeu à tout moment en appuyant sur ECHAP. Il suffit alors d'appuyer sur l'une des directions pour reprendre. Il est aussi possible de quitter le jeu à n'importe quel moment en appuyant sur la touche SUPPR. Cela fermera la console.

Plan de tests

Afin de vérifier le bon fonctionnement de notre programme, nous avons mis en œuvre tout au long du développement des tests ponctuels à l'ajoute des fonctionnalités pour vérifier leur bon fonctionnement. Malgré ce suivi régulier, nous avons mené une série de tests plus poussés permettant de valider le bon fonctionnement de l'ensemble du jeu et des fonctionnalités présentes sur ce dernier.

- **Test 1 :**
  Test au sein de la console lors du démarrage du jeu afin de vérifier si le programme réagit correctement lors d'une erreur de saisie de texte.

- **Test 2 :**
  Vérification du bon fonctionnement du choix des classes en début de partie.

- **Test 3 :**
  Test de la mort du joueur afin de vérifier la fonctionnalité permettant de rejouer.

- **Test 4 :**
  Test de toutes les capacités disponibles au sein du jeu sur les monstres. Pour cela on donne directement au joueur les capacités à tester.

- **Test 5 :**
  Test de tous les objets disponibles au sein du jeu. Pour cela on donne directement au joueur les objets à tester.

- **Test 6 :**
  Vérification des visuels et du bon affichage des éléments de jeu. Pour ce faire on parcourt les différents étages en vérifiant tous les éléments présents afin de vérifier qu'ils s'affichent de la bonne façon.

- **Test 7 :**
  Vérification du bon fonctionnement du système d'expérience. On élimine le maximum de monstres en vérifiant que la montée de niveau s'effectue et que la vie est bien modifiée.

- **Test 8 :**
  Test sur le système d'achat au sein des différents mondes afin de vérifier si le prix augmente bien selon le monde où se situe le marchand et si l'argent est bien déduit de l'inventaire du personnage.

- **Test 9 :**
  Test sur les différents boss afin de vérifier la fermeture des portes et leur ouverture lors de la mort de ce dernier.

- **Test 10 :**
  Vérification de la détection de la victoire lorsque le joueur meurt après avoir atteint le monde final.

- **Test 11 :**
  Enfin le dernier test a permis de vérifier la difficulté de notre jeu en enchainant un grand nombre de parties afin d'ajuster au mieux les statistiques des monstres ainsi que celles du joueur.

Gestion de projet

Planning

Afin de nous fixer des objectifs et pour notre organisation, nous avons élaboré un planning prévisionnel en nous fixant des dates butoirs réalisables pour ne pas accumuler de retard. Nous avons ainsi obtenu le planning ci-dessous (**cf. figure 22 et 23**).
Entre notre planning prévisionnel et notre planning réel, nous pouvons constater des différences plus ou moins importantes.

La première différence que nous pouvons constater se situe au niveau du choix de l'univers. En effet, nous avons mis plus de temps que prévu à définir l'univers autour de notre jeu ainsi que le scénario que suivra le joueur. En effet, le jeu devant être une application console et n'ayant tout les deux que peu d'expérience en programmation orientée objet, nous avons dû penser à quelque chose de simple, mais qui possède tout de même une trame scénaristique justifiant les éléments de notre jeu.

La deuxième différence majeure se situe dans le changement de l'apparence du jeu. Cela vient du fait qu'au cours de la réalisation du tutoriel, nous avons voulu au plus vite apporter une identité à notre jeu et avoir quelque chose qui nous ressemble. De plus, nous avons pensé que le plus tôt serait le mieux étant donné que peu de fonctionnalités étaient encore implémentées.

Enfin nous pouvons observer un léger décalage au niveau de la réalisation du tutoriel. Ce dernier étant bien expliqué et assez concis, nous avons eu besoin de moins de temps que prévu pour le réaliser. Ainsi nous avons pu passer plus de temps sur l'ajout de nos fonctionnalités ainsi que celles bonus.

De manière générale, le planning initialement établi a été globalement respecté et nous n'avons pas accumulé de retard.

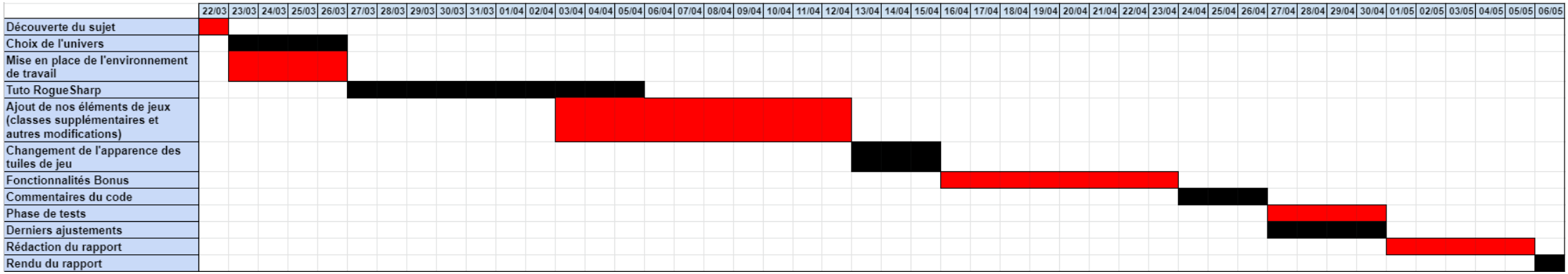

figure 22. Planning prévisionnel

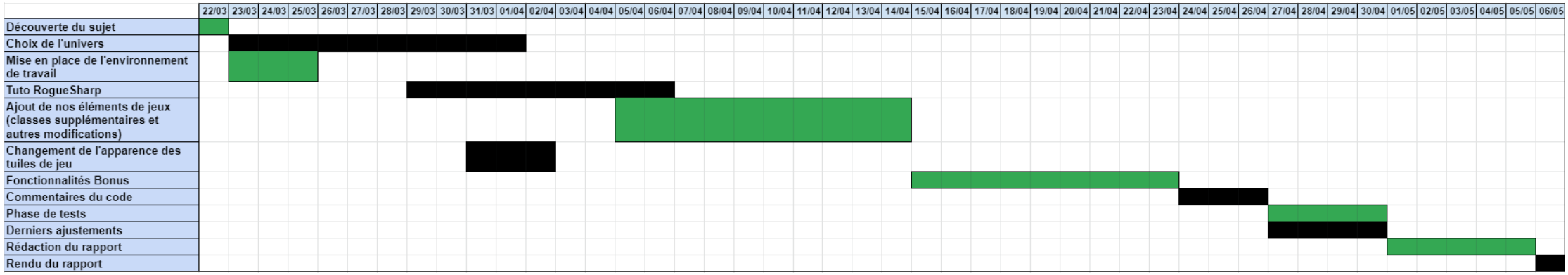

figure 23. Planning réel

Entre notre planning prévisionnel et notre planning réel, nous pouvons constater des différences plus ou moins importantes.

La première différence que nous pouvons constater se situe au niveau du choix de l'univers. En effet, nous avons mis plus de temps que prévu à définir l'univers autour de notre jeu ainsi que le scénario que suivra le joueur. En effet, le jeu devant être une application console et n'ayant tout les deux que peu d'expérience en programmation orientée objet, nous avons dû penser à quelque chose de simple, mais qui possède tout de même une trame scénaristique justifiant les éléments de notre jeu.

La deuxième différence majeure se situe dans le changement de l'apparence du jeu. Cela vient du fait qu'au cours de la réalisation du tutoriel, nous avons voulu au plus vite apporter une identité à notre jeu et avoir quelque chose qui nous ressemble. De plus, nous avons pensé que le plus tôt serait le mieux étant donné que peu de fonctionnalités étaient encore implémentées.

Enfin nous pouvons observer un léger décalage au niveau de la réalisation du tutoriel. Ce dernier étant bien expliqué et assez concis, nous avons eu besoin de moins de temps que prévu pour le réaliser. Ainsi nous avons pu passer plus de temps sur l'ajout de nos fonctionnalités ainsi que celles bonus.

De manière générale, le planning initialement établi a été globalement respecté et nous n'avons pas accumulé de retard.

Répartition des tâches

Les tâches ont été réparties de la manière suivante au sein du binôme :

	Evan	**Corentin**	**Evan et Corentin**
Conception	-Scénario/trame -Création des tuiles de jeu	-Dialogues/textes -UML des classes	-Choix de l'univers
Programmation	-Système d'achat/marchand -Système de boss -Écran de mort et rejouer -Système d'expérience -Musique -Jeu post-fin	-Système de mondes -Écran de pause -Création des différents monstres -Implémentation du système de classe -Détails visuels (équipement, objets)	-Tutoriel -Commentaires du code -Tests
Autre	-Dépôt sur GitHub		-Découverte du sujet -Planning -Rapport

figure 24. Répartition des tâches

Outils

Au cours de notre projet, nous avons eu recours à des outils de différents types afin d'effectuer des tâches, nous organiser et communiquer.

Outils de travail

Tout d'abord, nous avons utilisé Visual Studio 2019 sur nos postes personnels afin de développer notre jeu. Nous avons par ailleurs dû installer les packages OpenTK ainsi que RLNET afin de pouvoir utiliser la bibliothèque RogueSharp et réaliser ce projet.

Afin de réaliser notre image source, nous avons utilisé l'outil Paint 3D facile à prendre en main et rapide d'utilisation, car peu gourmand en ressource et présent de base sur nos ordinateurs.

Les UML ont été réalisé à l'aide de l'outil collaboratif en ligne LucidChart qui permet d'effectuer des MindMap simplement tout en travaillant à plusieurs sur le même fichier.

Outils de communication

Au niveau de la communication, nous avons communiqué via Discord pour effectuer des appels réguliers de suivi afin de faire le point sur l'avancement du projet couplet à des messages ponctuels sur Messenger afin d'échanger rapidement et plus simplement. Nous avons aussi profité des créneaux en présentiels pour travailler sur des points plus précis et plus simples à aborder de vive voix ainsi que pour échanger et demander des conseils à notre enseignant de TD.

Outils de gestion de projet

Ensuite pour nous organiser et répartir les tâches au sein du binôme, nous avons créé un tableau Trello qui nous a permis d'avoir une vision globale de ce qui était en cours de réalisation, ce qui était terminé, non débuté et sur quoi il fallait revenir.

Pour mettre en commun nos différents fichiers et rédiger ce rapport, nous nous sommes servis de Google Drive ainsi que de Google Docs, des outils auxquels nous sommes habitués et qui nous permettent de réaliser des travaux en collaborations tout en étant à distance.

Enfin pour déposer les livrables, nous avons utilisé le Dépôt GitHub créé pour cette occasion ainsi que l'application GitHub Desktop plus simple à prendre en main que la console pour des novices de GitHub.

Bilan

Au cours de ce projet, nous avons fait face à plusieurs difficultés. Ces dernières nous ont permis de progresser dans le domaine de la programmation orientée objet et nous ont demandé une certaine capacité d'adaptation et de raisonnement logique. Le sujet étant assez ouvert, mis à part un cahier des charges et une consigne générale, nous pouvions laisser libre cours à notre imagination ce qui est assez stimulant en particulier pour un projet de ce type.

La première difficulté rencontrée est liée à la base même d'un jeu vidéo, c'est l'histoire. En effet, imaginer un scénario qui tient la route en ayant un univers cohérent tout en pouvant proposer des éléments de gameplay intéressant n'est pas chose aisée. Surtout que nous nous trouvions limité d'un point de vue graphique comme fonctionnel. Nous ne pouvions pas utiliser de moteur de jeu comme Unity et nous n'avons jamais réalisé de jeu vidéo par le passé. Il nous a donc fallut un certain temps afin de déterminer et concevoir l'univers dans lequel le joueur allait évoluer.

Ensuite, nous avons fait le choix d'utiliser la bibliothèque RogueSharp. L'utilisation de cette dernière nous a conduit à faire face à deux difficultés majeures.

Tout d'abord il a fallu installer la librairie pour l'utiliser. Après une première installation qui fut un échec, nous nous sommes rendu compte après des recherches sur internet qu'il fallait créer non pas une application console de type "core" mais de type "framework" ce qui n'était pas précisé dans le tutoriel d'installation. Il a alors fallu tout réinstaller. Nous avons pu communiquer cette information aux autres groupes au sein de notre TD afin qu'ils ne fassent pas la même erreur.

Une fois que l'installation était effectuée, nous avons dû nous familiariser avec cette dernière et effectuer le tutoriel fourni. Cette tâche est une étape non négligeable à prendre en compte dans le planning de l'exécution de ce projet et nous a demandé un certain investissement afin de saisir l'ensemble des concepts pour commenter au mieux notre code.

Une autre difficulté que nous avons rencontrée concerne le changement de l'apparence de notre jeu. En effet, afin d'avoir un résultat plus esthétique, nous avons décidé de changer l'apparence des tuiles du jeu. Cependant, rien n'était expliqué au sein du tutoriel en ce qui concerne la lecture de l'image contenant les caractères. Il a alors fallu réfléchir à une manière de procéder. Nous avons d'abord tenté différentes idées qui furent des échecs, puis nous avons remplacé l'un des caractères textuels par une image. Lorsque nous entrions la touche remplacée, l'image apparaissait à la place du caractère. Nous avons alors remplacé l'ensemble des caractères et avons découvert que le programme pour lire l'image utiliser l'emplacement du caractère. Nous avions alors la méthode pour utiliser n'importe quel caractère simplement à partir de son emplacement sur la grille.

De plus, nous avons eu quelques difficultés lors de l'ajout de certaines fonctionnalités comme le système de marchand, car les objets ne voulaient pas s'afficher au bon endroit où ne s'affichait pas parfois ou encore le système de verrouillage de porte pendant un boss.

Il y a une chose que nous n'avons pas mis en place, les fenêtres textuelles et les dialogues avec les PNJ. En effet, cela ne semble pas être possible avec RLNET d'afficher des bulles textuelles lorsqu'un personnage parle ainsi, l'interaction avec les PNJ se serait faite via la console de message en bas de l'écran et n'aurait pas été agréable à utiliser. Nous avons donc abandonné l'idée. Ainsi, nous avons préféré afficher un texte simple avec les informations nécessaires lorsque le joueur arrive dans la salle d'un PNJ.

Malgré cela, nous avons réussi à résoudre nos problèmes et à disposer d'un jeu comportant toutes les fonctionnalités que nous désirions au départ et qui s'approche au maximum de ce que nous souhaitions.

Crédits

Musiques

Monde 0 et monde final :
Beginning of the Journey - The Legend of Zelda: A Link to the Past

Monde 1 :
Legend of Zelda: A link to The Past music - overworld theme

Monde 2 :
Hyrule Castle - The Legend of Zelda: A Link to the Past

Monde 3 :
The Dark World - The Legend of Zelda: A Link to the Past

Modes mode infini :
Celeste Original Soundtrack - Resurrections

Boss 1 :
Jotaro's Theme - Stardust Crusaders (ft. Fabulous Reindeer)

Boss 2 :
Josuke's Theme - Diamond is Unbreakable - JoJo's Bizarre Adventure

Boss 3 :
Giorno's Theme ~ "il vento d'oro" - JoJo's Bizarre Adventure: Golden Wind

Boss mode infini :
Undertale - Megalovania

Images

Kenney • 1-Bit Pack
Anvil | OpenGameArt.org

Tutoriel

[Creating a Roguelike Game in C# | Game programming in .NET with RogueSharp]