Document de Conception – Jeu RPG en Java

Introduction :

Ce document fournit une vision détaillée de la conception du logiciel pour la création d'un jeu RPG en Java, intitulé "Jeu de rôle Java". Il couvre des aspects techniques tels que l'architecture du système, les stratégies de conception, les rôles et responsabilités. Mais aussi tous les aspects d'organisation d'un projet en groupe : la communication, la convergence de multiples idées, la résolution de problème, etc.

Aperçu du Projet :

Le jeu RPG en Java que nous avons nommé DEIZZAD propose une expérience immersive avec des niveaux variés, des ennemis, des objets et des mécaniques de plateforme.
Nous avons décidé de nous inspirer du jeu marios bros, à la fois car il s'agit d'un jeu de notre enfance sur lequel nous avons passé de longue heures étant petits, mais aussi car il représente pour nous l'équilibre parfait entre difficulté et réalisabilité : le joueur à toute les compétences en mains pour arriver à ses fins tout en étant challengé par une certaine difficulté, ouvrant de nombreuses possibilité d'amélioration et d'entraînement afin d'être le plus rapide possible. Le jeu est ainsi en deux dimension, vue de profil comme le montre l'image ci dessous :

Le jeu est développé en version graphique avec une interaction au clavier lorsque le jeu est lancé, et à la souris pour une navigation aisée dans les menus du jeu. Nous avons décidé d'implémenter un menu simple et efficace regroupant plusieurs informations afin de simplifier l'utilisation de DEIZZAD par l'utilisateur novice : des instructions concernant les touches de jeu ont été indiquées. De plus, 3 boutons cliquables permettent de lancer ou de relancer le jeu, de modifier les options du jeu ou de quitter tout simplement le jeu si vous avez suffisamment joué. Une image valant plus que mille mots, voici plus bas l'apparence concrète de notre menu :

Choix du personnage principal :

Pour le choix de notre personnage principal, nous avons opté pour un personnage représentant l'univer viking, à la fois car il est tout à fait adapté pour notre jeu de choisir une personnage de la sorte, mais aussi car la texture de Viking que nous nous somme procuré regroupait tous les critères d'animations que nous souhaitions utiliser : Idle, lorsque notre personnage est en repos, qu'il ne se déplace pas. Ainsi, nous simulons le mouvement vertical de vas-et-vient causé par sa simple respiration. L'animation Jump, l'animation de saut lorsque la touche Espace actionnant le saut du joueur est pressée. L'animation de marche lorsque les flèches gauche et droite du clavier sont actionnées.
De plus, notre texture de joueur possède encore quelques animations permettant d'expliciter une course, offrant à notre équipe de développeurs une marge de progression considérable nous permettant d'ajouter un mouvement supplémentaire et plus de dynamisme à notre jeu.

Type de Map :

Nous avons opté pour un choix de map en deux dimensions, vue de profil, à l'image du jeu mario. Cette map est d'une efficacité redoutable car elle permet une distinction très simple des différents éléments du jeu (ennemis, potions, décors, etc) tout en conservant une certaine élégance vintage que nous avons décidé de conserver : notre jeu se veut être un hommage au monde du jeu vidéo et hommage à ses grands grands classiques.
Notre implémentation de map, que nous trouvons suffisante et satisfaisante pour notre rendu, peut malgré tout être améliorée : nous nous sommes laissé la possibilité d'y ajouter

différents décors, statiques et animés pour rendre notre jeu encore plus complet, élégant et divertissant.

Monstres :
Discutons maintenant du choix d'un des éléments principaux du jeu, rendant notre chemin initial vers la victoire d'une simplicité enfantesque, à un véritable parcours du combattant : les ennemis. L'ennemi numéro 1 de notre jeu est une créature féroce qui patrouille dans les niveaux de DEIZZAD. Doté d'un instinct prédateur, il réagit de manière agressive dès qu'il détecte la présence du héros dans son champ de vision et fera tout pour l'empêcher d'arriver à son objectif final.
Cet ennemi est l'orque rouge, dit RedOrc. Son implémentation graphique résulte d'un choix similaire au sprite du héros : en plus de son élégant design simple et sobre, toutes les animations que nous souhaitions avoir lors de la prise de décision en groupe du début de projet étaient présentes : la marche, l'attaque avec ses doubles épées, ainsi que la mort.

Outils Utilisés :
- **Java**: Langage de programmation imposé mais très efficace dans ce genre de projet : un véritable soulagement de ne pas avoir à effectuer toute la gestion de la mémoire comme nous l'avons fait en langage C depuis des années.
- **Git**: Outil de gestion de version pour le suivi des modifications et la répartition des tâches. Nous nous sommes servi de GIT de différentes façons. Premièrement, il s'agissait d'un moyen simple et efficace de stocker notre code, d'uploader les mise à jour effectuées par un membre du groupe et d'envoyer ses modifications aux autres. Secondement, en utilisant sa fonctionnalité de merge pour fusionner les différentes parties.
- **Discord**: Plateforme de communication pour une collaboration efficace permettant le travail et l'avancement du projet à distance. Ce logiciel permet aussi le partage de dossier et d'informations ainsi que la communication vocale à distance. Il nous à été très pratique pour communiquer, notamment pendant les vacances où chacun était chez soi et les jours où nous n'étions pas sur le campus.
- **Doxygen**: Outil de documentation du code source, pour référencer chaque partie du code et ainsi comprendre à tout moment ce que chacune des fonctions fait. Nous utilisons aussi doxygen car nous l'avons utilisé durant les années précédentes dans nos études. Cet outil de documentation nous est familier et nous gagnons ainsi du temps à l'utiliser.
- **VSCode**: Environnement de développement intégré avec une syntaxe vérifiée par le linter. Outils de développement très efficace et largement suffisant pour les besoins de développement de DEIZZAD.
- **XMind**: Outil pour les diagrammes de classes, utilisé pour collaborer à plusieurs sur un même fichier. Très pratique d'utilisations, permet de définir simplement les différents diagrammes pour notre projet.

Un outils que nous nous laissons libre d'utiliser pour de possible amélioration est l'utilisation des énumérations : en effet, bien que l'intérêt des énumérations pour notre projet est claire (permet de détecter une erreur dès la compilation et nous avertira très tôt dans l'exécution du code), elle implique de changements massif dans le code et, malheureusement, faute de temps, cette caractéristique entre dans la catégorie des améliorations possible pour notre jeu.

Contraintes et impératifs à suivre pour le développement du jeu :

Nous cherchons à mettre en place une gestion précise des mouvements du personnage et des collisions, garantissant une expérience fluide. De plus, nous souhaitons instaurer un système de niveaux évolutifs et diversifiés pour maintenir l'intérêt du joueur tout au long du jeu. L'intégration harmonieuse des ennemis, objets et mécaniques de saut est également cruciale pour offrir une expérience de jeu immersive. Enfin, il est essentiel d'assurer une adaptation cohérente au style visuel et sonore du jeu, garantissant une expérience homogène et attrayante pour les joueurs.

Stratégies Architecturales :

Dans le cadre des stratégies architecturales, nous avons opté pour l'utilisation du modèle MVC (Modèle-Vue-Contrôleur). Cette approche vise à établir une séparation claire entre trois composants essentiels du jeu : le modèle du jeu, la vue graphique et le contrôleur utilisateur.

1. Modèle du jeu (Model) : Cette partie gérera la logique métier du jeu, incluant la gestion des niveaux, des personnages, des ennemis, et d'autres éléments cruciaux. Il s'occupera également de la manipulation des données du jeu, telles que la progression du joueur, les statistiques, etc.

2. Vue graphique (View) : La vue graphique sera responsable de l'affichage visuel du jeu. Cela inclut la gestion des sprites, des animations et de tout élément visuel nécessaire à l'expérience du joueur. La séparation nette entre le modèle et la vue permet une flexibilité dans la présentation, facilitant ainsi d'éventuelles modifications esthétiques.

3. Contrôleur utilisateur (Controller) : Le contrôleur utilisateur prend en charge les interactions entre le joueur et le jeu. Il traduit les actions de l'utilisateur en commandes compréhensibles par le modèle du jeu. Cela inclut la gestion des entrées du clavier, de la souris ou d'autres dispositifs de contrôle, permettant une interactivité fluide.

En outre, pour la gestion des sprites, animations et interactions, nous adopterons une approche orientée objet. Cela signifie que chaque entité du jeu, qu'il s'agisse d'un personnage, d'un ennemi ou d'un objet, sera représentée comme un objet avec des caractéristiques et des comportements spécifiques. Cette approche favorise la modularité, la réutilisation du code et une maintenance plus aisée.

L'utilisation conjointe du modèle MVC et de l'approche orientée objet permettra une structure robuste, favorisant le développement, la maintenance et l'extension du jeu de manière efficace et structurée.

Architecture du Système :

1. Moteur de Jeu :

- Le moteur de jeu constitue le cœur du système, assurant la gestion de la logique du jeu, la progression des niveaux et l'état global. Il coordonne les interactions entre les différents composants du jeu et maintient une vue d'ensemble de l'évolution de la partie.

2. Gestion des Niveaux :

- Ce module est dédié à la manipulation des niveaux du jeu. Il se charge du chargement, de l'affichage et de la gestion des différents environnements de jeu. Il offre également la possibilité de passer d'un niveau à un autre en fonction de la progression du joueur. La gestion des objets spécifiques à chaque niveau est également prise en compte.

3. Système de Personnages :

- La gestion des personnages, en particulier du personnage principal, est confiée à ce composant. Il supervise les mouvements, les sauts et les collisions du personnage, assurant ainsi une expérience de jeu fluide et précise. L'affichage du personnage est également géré grâce à l'utilisation de sprites spécifiques, permettant une représentation visuelle cohérente et attrayante.

4. Ennemis et Objets :

- Cette partie du système intègre de manière prédéfinie les ennemis et les objets sur la carte du jeu. Les assets (sprites) spécifiques sont utilisés pour l'affichage, garantissant une représentation visuelle distinctive et informative. Des explications détaillées sur les caractéristiques et comportements des ennemis et objets sont également prises en compte, facilitant ainsi la compréhension du joueur.

5. Affichage Graphique :

- L'affichage graphique englobe la représentation visuelle de tous les éléments du jeu, y compris le personnage, les ennemis et les objets. Il repose sur l'utilisation judicieuse de sprites et d'animations pour donner vie aux différentes entités du jeu. L'intégration harmonieuse des éléments visuels contribue à l'immersion du joueur dans l'univers du jeu. Des techniques telles que le rendu par lots (batch rendering) peuvent être employées pour optimiser les performances graphiques. Une attention particulière est portée à l'adaptation cohérente au style visuel global du jeu, assurant ainsi une expérience visuelle plaisante et homogène.

6. Système de Connexion :

Cette caractéristique qui, au départ devait faire partie intégrante de notre jeu, a été remise à plus tard. Nous pourrions nous concentrer dessus par la suite afin de développer une partie très intéressante dans les jeux-vidéos et dans l'informatique en général : la sauvegarde des données de progression via une gestion mémoire des comptes utilisateur.

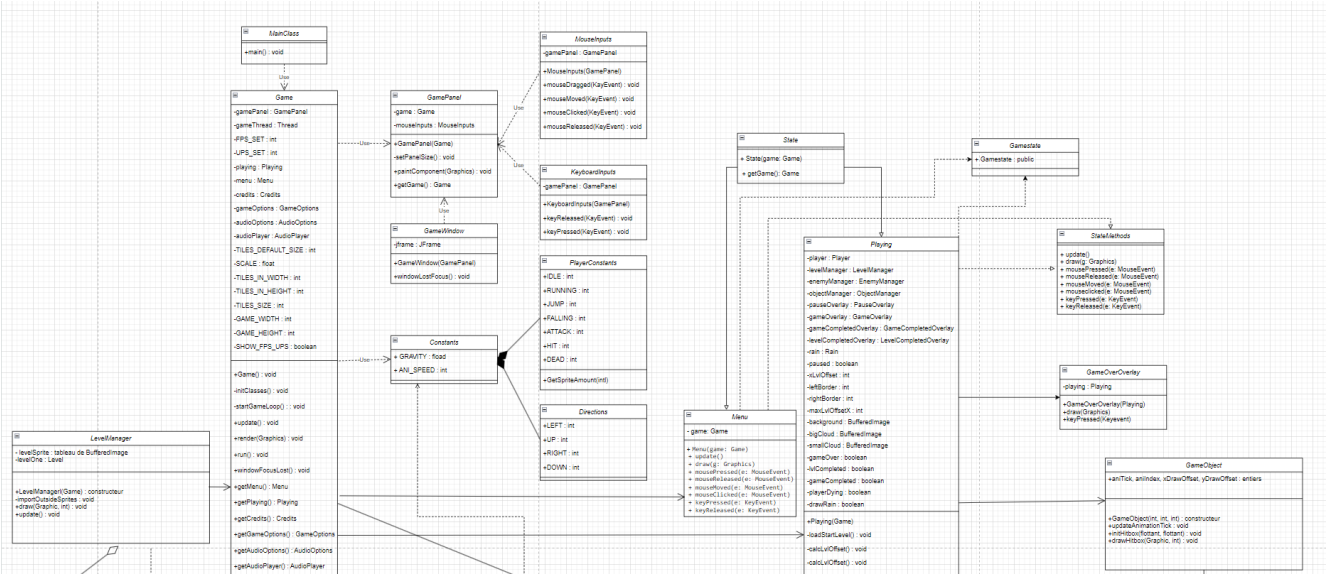
Conception Détaillée du Système :

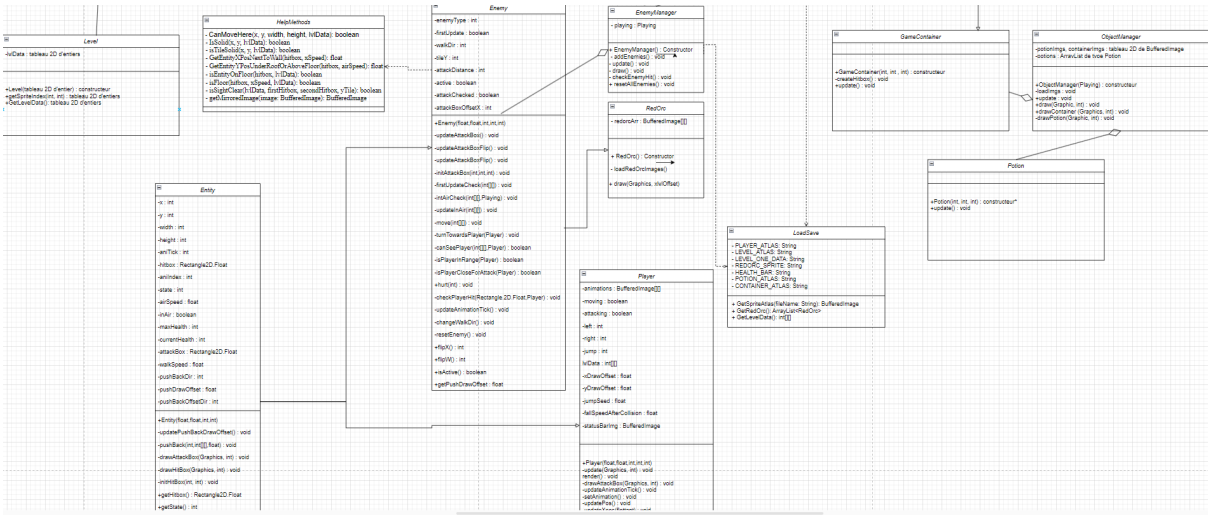
- Classes détaillées pour le personnage principal, les ennemis, les niveaux, etc.

- Méthodes spécifiques pour la gestion des collisions, des objets, etc.

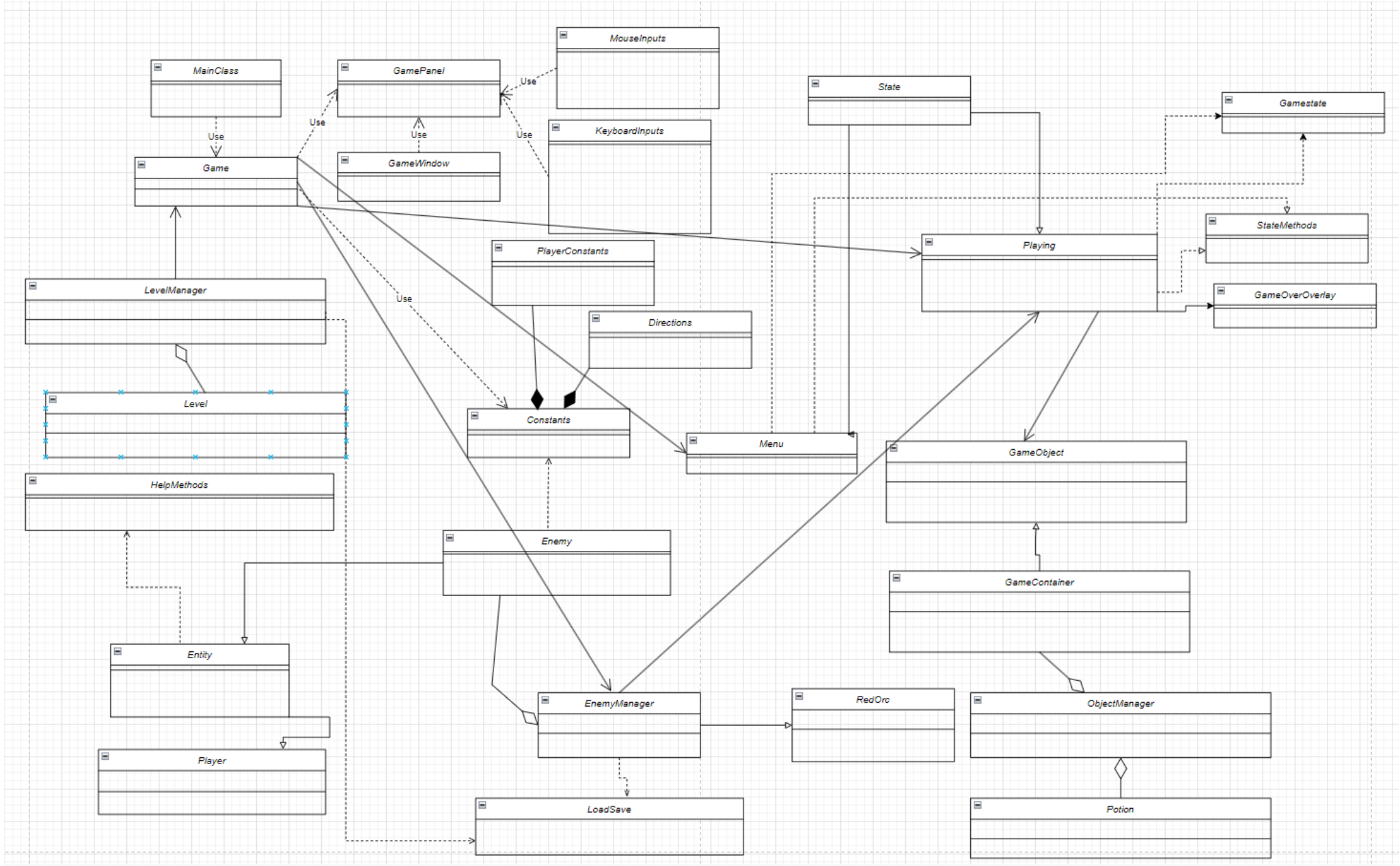
Différentes Classes :

Voici notre diagramme des classes (pour des soucis de précision et de manque de visibilité, nous rendons accessible notre diagramme des classes via le lien suivant : https://drive.google.com/file/d/1GwYjXcoV_UdtPWSzuXO1fTfNwTFGY_In/view?ts=656de2bb):





La lisibilité de ce diagramme de classe et le lien entre chacune des classes n'étant pas optimale, voici, ces mêmes informations sont présentes dans une version allégées de notre diagramme :


MouseInputs
MainClass
GamePanel
State
Gamestate
Use
Use
Use
Use
Use
KeyboardInputs
Game
GameWindow
StateMethods
PlayerConstants
Playing
LevelManager
GameOverOverlay
Use
Directions
Level
Constants
Menu
GameObject
HelpMethods
Enemy
GameContainer
Entity
EnemyManager
RedOrc
ObjectManager
Player
LoadSave
Potion

Informations principales :

La classe objets à pour objectif de fournir au joueur une quantité de soin limitée mais suffisante pour rendre le jeu challengeant. Des potions et des boîtes contenant des items permettent au joueur de régénérer une partie de sa santé après en avoir consommé. Néanmoins, par soucis de temps et pour se concentrer sur un jeu fonctionnel à chaque instant, la partie objets n'a pas pu être terminée mais nous laisse une marge et une liberté de développement même après le rendu de notre projet.
L'objectif de cette classe, à terme, serait de faire en sorte que chacun des objets présent sur la carte soient nécessaires pour le joueur : il faudrait donc ainsi effectuer quelques réglage sur la puissance des ennemis et la fréquence d'apparition des objets pour rendre le jeu excitant et doté d'un certain niveau de difficulté.

A vrai dire, nous avons pris la décision de mettre de côté la classe objet pour le moment en nous concentrant sur les choses plus essentielles afin de rendre un jeu propre et complet, tout en nous laissant une marge d'amélioration pour la suite. Cependant, comme le montre l'image ci-dessous, nous avons bien réussi à implémenter différents types d'objets et à les afficher à l'écran. Il s'agit là d'un bon début pour la suite de notre jeu.

Nous voyons ici deux potions gardées par les ennemis. Dans l'idéal, lorsque le joueur entre en collision avec l'une de ces potions, il régénère une partie de la vie perdu durant les combats précédents. Nous avons prévu de procéder de même avec des boîtes appelées "containers" dans notre code qui contiendrait ou bien des pièces, ou bien du soin.

Concernant l’IHM :

- Présence d’un menu**:**

Dans la mise en œuvre de la partie IHM, une attention particulière sera accordée à la conception du menu. L'objectif est de créer une interface conviviale et esthétiquement agréable. Cela implique la mise en place d'une disposition claire et intuitive, permettant aux joueurs de naviguer facilement entre les différentes options du menu, comme le démarrage du jeu, les paramètres et l'option de sortie. Des éléments visuels attrayants et des animations subtiles peuvent être utilisés pour améliorer l'expérience utilisateur.

- Ergonomie du système de connexion :

Pour faciliter l'interaction entre le joueur et le jeu, un système de connexion ergonomique sera mis en place. Cela peut inclure la possibilité pour les joueurs de créer des profils personnalisés, de sauvegarder leur progression, et de personnaliser les paramètres du jeu. L'utilisation de boutons clairs et d'instructions simples rendra le processus de connexion intuitif, contribuant ainsi à une expérience utilisateur fluide.

- Boutons intuitifs :

Les boutons dans le jeu sont conçus de manière à être intuitifs pour le joueur. Cela signifie qu'ils seront placés de manière logique et identifiable, correspondant aux attentes naturelles des utilisateurs. Par exemple, les boutons de contrôle du personnage seront positionnés de manière à refléter les mouvements attendus du joueur. Les actions telles que sauter, attaquer ou accéder au menu seront attribuées à des boutons facilement repérables, favorisant ainsi une prise en main rapide et une utilisation instinctive.
Les touches de jeu sont aussi indiquées avant le début de chaque partie, de manière à rendre le jeu plus simple d’utilisation et allant droit au but.

L'implémentation de ces éléments d'IHM vise à rendre le jeu accessible et convivial pour un large public. L'objectif est de minimiser la courbe d'apprentissage, permettant aux joueurs de se plonger rapidement dans l'expérience de jeu sans être freinés par des complexités inutiles. En fin de compte, une IHM bien pensée contribuera à l'attractivité globale du jeu en offrant une expérience utilisateur agréable et sans friction.

Rôles et Responsabilités :

Aris :

Partie Ennemis : Aris sera principalement responsable de la conception et de la mise en œuvre des ennemis dans le jeu. Cela inclut l'intégration prédéfinie des ennemis sur la carte, la gestion de leurs comportements, et l'utilisation d'assets spécifiques (sprites) pour assurer leur représentation visuelle.

- *Level* : Aris sera également chargé de la gestion des niveaux du jeu. Cela implique le chargement, l'affichage, et la gestion des différents environnements de jeu, en veillant à ce que l'expérience de jeu soit progressive et captivante.

Jonathan :

Player : Jonathan prend en charge la gestion du personnage principal, incluant les mouvements, les sauts, et les collisions. Il veillera à ce que le contrôle du joueur sur le personnage soit précis et réactif, contribuant ainsi à une expérience de jeu fluide.

Gravité : La gestion de la gravité sera également de la responsabilité de Jonathan. Il s'assurera que la physique du jeu, en particulier les mécanismes de saut et de chute, soit cohérente et adaptée au style du jeu.

Objets : Jonathan supervisera l'intégration des objets dans le jeu. Cela inclut la définition des propriétés et des comportements des objets, ainsi que l'utilisation d'assets spécifiques pour leur représentation visuelle.

Anass :

Gameloop : Anass sera responsable de la mise en place du gameloop, l'élément essentiel qui gère la séquence continue du jeu. Il veillera à ce que le jeu fonctionne de manière cohérente et réactive, en contrôlant les mises à jour régulières du jeu.

Collision : La gestion des collisions, un aspect crucial pour assurer l'interaction réaliste entre les entités du jeu, sera également de la responsabilité d'Anass. Il s'assurera que les collisions sont détectées de manière précise et que les conséquences appropriées sont appliquées.

Gamestate et Menu : Anass sera chargé de la gestion du Gamestates, y compris l'implémentation du menu. Il garantira une transition fluide entre les différents états du jeu, comme le menu principal, le gameplay, et les écrans de fin de partie.

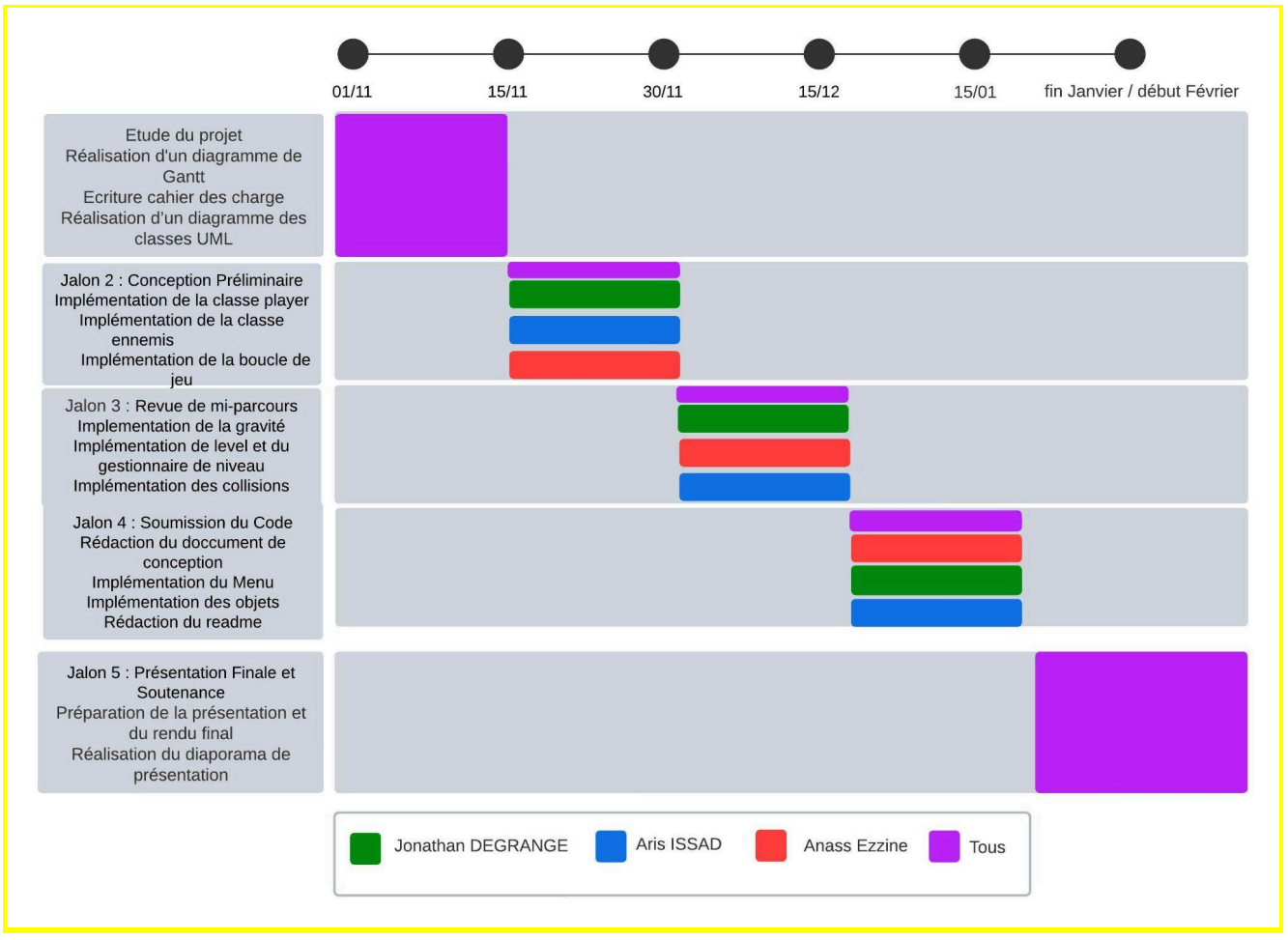
Cette répartition des rôles et responsabilités assure une division claire des tâches entre les membres de l'équipe, favorisant une collaboration efficace et une avancée cohérente du développement du jeu. Chaque membre apporte ses compétences spécifiques pour contribuer à différents aspects du projet, assurant ainsi une réalisation complète et bien coordonnée.

L’ensemble des participants de notre groupe, avant de commencer à développer les parties citées ci-dessus, s’est réunie et a longuement discuté de la multitude de directions possibles et imaginables qu’un jeu de la sorte pouvait présenter. De nombreux choix tels que l’utilisation de la 3D ou de la 2D, les différentes compétences du joueur et des ennemis, le concept de points de vie, etc, ont été débattus afin de faire converger nos différentes idées vers une direction commune. Ensuite, de longs échanges ont été effectués pour mettre au

point un diagramme des classes concret et fonctionnel et planifier l'avancement du projet avec un calendrier. Ainsi, avant même de commencer à coder, nous savions où nous allions et comment y arriver, tout en nous laissant une certaine liberté de notre propre expression au fur et à mesure du développement du jeu. Certaines fonctionnalités que l'on pensait importantes n'ont, par exemple, pas vu le jour, tels que les dessins en fond de page pour une meilleure qualité esthétique. Faute de temps, toutes ces caractéristiques mises à l'écart pour une meilleure finalisation du projet nous offrent une marge de progression pour la suite du développement de notre jeu.

Organisation:

Nous nous somme servi d'un diagramme de gantt afin de planifier toutes les étapes importantes du développement de notre projet :



Principes d'Architecture et de Conception :

Afin d'assurer un développement sûr et sans erreur dans notre projet, nous avons opté pour le cycle de développement en V. Le cycle en V est une méthode de développement logiciel permettant d'assurer le bon fonctionnement de chacun des niveaux d'avancement du jeu afin de pouvoir passer au niveau suivant. Bien que cette méthode de développement présente ses failles, elle nous à semblé correcte d'utilisation pour un projet de cette échelle.
- Modularité pour faciliter l'extension du jeu. L'ajout de nouveaux ennemis par exemple n'entraînera pas la modification de tout le code.
Chaque fonctionnalité majeure du jeu, qu'il s'agisse de la gestion des niveaux, des ennemis, du personnage, ou d'autres aspects, sera encapsulée dans un module distinct. Par exemple, le module Ennemis pourrait inclure toutes les classes et méthodes nécessaires à la création, la gestion, et le comportement des ennemis.
- Utilisation de l'approche orientée objet pour une conception flexible.

Design Patterns :

- Template Method Pattern :
    - La classe abstraite Enemy contient une méthode abstraite draw, qui est un exemple de modèle de méthode. Les sous-classes spécifient les détails d'implémentation tout en laissant la structure générale définie dans la classe de base.
    - Les méthodes mousePressed, mouseReleased, mouseMoved, mouseclicked, keyPressed, et keyReleased sont des exemples de modèle de méthode. Elles sont définies dans l'interface Statemethods et implémentées dans la classe Playing.
- Strategy Pattern :
    - Le code utilise une approche de stratégie pour gérer le comportement des ennemis. Par exemple, les méthodes comme updateMove, updateInAir, et updateAttackBox délèguent la logique spécifique aux sous-classes qui implémentent ces comportements.
- State Pattern :
    - La classe Enemy utilise un état interne pour déterminer le comportement actuel de l'ennemi. L'état est représenté par la variable enemyState, et les méthodes telles que newState changent l'état actuel de l'ennemi.
    - La classe Playing représente un état du jeu. Elle étend la classe abstraite State et implémente l'interface Statemethods. La gestion des états (levelComplete et gameOver) ainsi que les méthodes comme loadNextLevel montrent une utilisation du State Pattern.
- Observer Pattern :
    - La classe Enemy interagit avec un objet Player en passant cet objet en paramètre à la méthode update. Cela pourrait être interprété comme une utilisation du modèle observé, où l'ennemi "observe" le joueur et réagit en conséquence.
    - La classe Playing observe les événements liés à la souris (mousePressed, mouseReleased, mouseMoved, mouseclicked) et aux touches (keyPressed, keyReleased). La méthode update semble également observer l'état du

niveau (levelComplete et gameOver). Cela reflète une utilisation possible du pattern observer.

- Factory Method Pattern :
    - La méthode initAttackBox agit comme une factory method pour initialiser la boîte d'attaque de l'ennemi. Cependant, la complexité de cette initialisation peut également être vue comme une application du Pattern Builder.

Représentation Schématique :

- **Diagramme de Classe :** (https://drive.google.com/file/d/1GwYjXcoV_UdtPWSzuXO1fTfNwTFGY_In/view?ts=656de2bb)
- **Diagramme de Séquence :**(https://drive.google.com/file/d/1weVAp139uI-9IAIncEYR9GbKx07PnOpJ/view?usp=drive_link)

Assurance Qualité :

Commentaire Doxygen :

- Nous utilisons des commentaires Doxygen pour documenter nos classes et méthodes.

Fonctions bien nommées :

- Nous avons choisi des noms de méthodes et de variables significatifs, par exemple, loadNextLevel, checkCloseToboredr, resetAll, etc. Cela rend notre code lisible et compréhensible.

Code évolutif :

- La séparation des responsabilités entre différentes classes, telles que EnemyManager, ObjectManager, et LevelManager, ainsi que l'utilisation de méthodes comme loadNextLevel, suggère une approche de conception qui peut rendre le code plus évolutif.

Relire le code pour comprendre et chercher des bugs :

- L'utilisation de commentaires, de noms de variables explicites et de méthodes courtes facilite la relecture du code. Cela aide également à identifier des bugs potentiels ou des améliorations.

Outils de débogage, par exemple, drawAttackBox :

- La méthode drawAttackBox dans la classe Enemy est un exemple d'utilisation d'outils de débogage. Cette fonctionnalité permet de visualiser la boîte d'attaque, facilitant ainsi le processus de débogage.

Méthodes courtes pour éviter les bugs :

- Les méthodes dans nos classes semblent relativement courtes et spécifiques à leurs tâches respectives. Par exemple, la méthode update de la classe Playing est relativement concise, ce qui est une bonne pratique.

Organisation de l’équipe autour du projet:

Au commencement du projet, une collaboration étroite sera établie entre tous les membres de l'équipe pour organiser et définir les bases du projet. Cela comprend la création d'un plan global qui englobe les classes, les méthodes, et d'autres aspects cruciaux du code. Des discussions approfondies seront menées pour s'assurer que chaque membre comprend la vision globale du projet et est en accord sur l'approche à suivre. Cette phase initiale constitue également une opportunité pour l'équipe de proposer des idées et de définir des objectifs spécifiques du projet.
La création des diagrammes d'état, du modèle conceptuel de données (MCD), et du diagramme des classes sera réalisée en collaboration. Les membres de l'équipe travaillent ensemble pour définir les états du jeu, modéliser les données nécessaires, et concevoir la structure des classes du code. Ces diagrammes serviront de référence pour le développement ultérieur et seront soumis à des validations régulières par le tuteur. L'objectif est de s'assurer que la conception initiale est solide et conforme aux attentes avant de passer à la phase de développement.
Une fois que l'organisation du projet a été définie et validée, la répartition du code et de la charge de travail sera effectuée. Les tâches spécifiques seront attribuées en fonction des compétences et des domaines d'expertise de chaque membre de l'équipe. Il est essentiel que cette répartition soit équitable, tenant compte des forces individuelles de chaque membre tout en veillant à ce que chaque partie du projet soit bien couverte.
Un suivi régulier sera mis en place pour s'assurer que le développement progresse de manière cohérente. Des réunions d'équipe sont organisées pour discuter des avancées, résoudre les problèmes éventuels, et ajuster la répartition des tâches si nécessaire. La communication ouverte et transparente sera encouragée pour garantir une collaboration efficace tout au long du développement du projet.

Maquette : Notre toute première maquette de conception de notre jeu. Lorsque nous réfléchissons encore à toutes les possibilités offertes par ce projet. Nous avions pensé à un fonctionnement avec un boss de fin, des échelles pour augmenter l'interaction du joueur avec l'environnement du jeu, des attaques par boule de feu, etc. Toutes ces idées ne sont pas perdues et peuvent donner suite à une autre version du jeu DEIZZAD.