Rapport de Projet

Projet sur la création

d'un jeu vidéo sous base Java

DI BELLO Norman

DUEZ Benjamin

SOMMAIRE

Table des matières

Introduction

Nous avons décidé de choisir, comme projet, la création d'un jeu vidéo de type RPG car il coïncidait entre nos deux points de vues. Comme cette année, nous avons débuté l'animation avec le JavaFx, nous avons opté pour utiliser le langage Java pour pouvoir mettre à bien nos idées.

But du projet

Un point de vue large

Le but de ce projet est, donc, comme son nom le fait comprendre la création d'un jeu vidéo.

Nous avons pour objectif de faire un jeu vidéo avec trois étapes différentes. La première sera un combat basique entre l'équipe de personnages, que le joueur va contrôler, et une équipe de monstres qui est géré de façon "aléatoire". La deuxième partie consistera en un puzzle-game, c'est-à-dire, une partie où les statistiques des personnages n'étaient pas nécessaires à la bonne réussite de la partie mais les capacités que le joueur (celui qui commande les personnages) a pour résoudre l'énigme qui va se présenter devant lui. La dernière partie sera, de nouveau, un combat mais cette fois-ci il s'agira d'un combat de boss, soit un combat contre une entité d'un calibre supérieur à celui des monstres dans la première phase.

De façon plus concrète

Le jeu vidéo va se baser dans un open-world, c'est-à-dire, un univers sans "limite" où le joueur peut visiter le monde de long en large sans but direct, ce qui a pour conséquence de devoir créer une map monde (l'endroit où le joueur va se déplacer) assez grande et diversifiée pour éviter quelconque ressentiment de redondance. Cette map monde devrait être composée de plusieurs composantes.

Il y aura bien évidemment des villages, où le joueur pourra remettre les capacités vitales de ses personnages à leur maximum, le joueur pourra également acheter de nouveaux équipements pour que ses personnages se voient être attribués de nouvelles capacités et/ou statistiques et les villageois auront la possibilité de donner des quêtes secondaires au joueur pour que celui-ci puisse continuer de progresser en dehors des combats.

Une autre composante de la map monde serait les donjons, dans ceux-ci nous allons avoir la partie phare de notre projet, les combats. Les combats seront générés aléatoirement, ceux-ci ne seront donc pas prévus à certaines positions où le joueur se déplacera, à la place le joueur aura de plus en plus de chances de tomber face à des ennemis au fur et à mesure de ses déplacements. Les donjons offriront des récompenses au joueur à leur complétion. Nous pourrons trouver des coffres dans lesquels des objets uniques seront potentiellement obtenable.

Partie Théorique

Globalement

Comme dit précédemment, dans la partie théorique, nous avons décidé de créer trois packages pour séparer les êtres, les items et les combats.

Le package des êtres va constituer tout ce que le joueur va contrôler, affronter ou, bien encore, interagir avec.

Dans le cas du package des items, les classes composant ce package correspondront aux éléments qui seront rattachés au personnage et qui les aideront aux combats.

Finalement, le dernier package est celui des combats. Celui-ci va permettre de définir comment un combat se déroule, comment les dégâts sont infligés ou reçus.

Tandis que certaines classes ont été créées, certaines ne seront pas utilisées, comme par exemple, les classes dans le package Item.

Package Etre

Tout d'abord, nous avons commencé par la création des classes dans l'ordre. Nous avons créé la classe Etre_vivant qui va regrouper les statistiques que tout être vivant va posséder. Celles-ci sont séparer en 2 groupes, celles qui vont permettre de calculer les compétences et celles qui regroupent les capacités vitales.

Pour le calcul des compétences, nous avons, donc, 4 notions de base, l'attaque, la magie, l'esprit et l'armure. L'attaque va permettre de calculer les dégâts infligés grâce à une attaque physique. La magie va permettre de calculer, principalement, une attaque si celle-ci est magique mais également de faire part aux calculs d'un soin. L'esprit fait lui aussi part au calcul d'un soin et comme la magie est une statistiques a double utilisation car elle permet, également, de calculer, avec l'armure, le bouclier qu'un personnage va se créer s'il choisit de se défendre.

Pour les capacités vitales, nous avons deux statistiques les points de vie (hp) et les points de mana (mp). Que cela soit les points de vie ou bien les points de mana, nous avons deux paramètres différents les hp_max et les hp, et, les mp_max et les mp.

Les hp et les mp permettent de calculer combien de vie (ou de mana) un être vivant a au moment présent. Si ces paramètres tombent à 0, cela veut dire qu'ils sont soit à court de points de vie, c'est-à-dire K.O. ou sans mana, donc qu'ils ne pourront plus utiliser de compétences nécessitant de points de mana.

Les hp_max et mp_max, eux, ne servent pas spécialement, ils forment plus une barrière pour éviter qu'un soin ne permettent aux personnages d'avoir plus d'hp (mp) que possible, par exemple, un Tank avec 60 hp_max et 45 hp qui reçoit un soin de 20, ne recevra qu'un soin de 15 qu'à sa limite d'un hp qui correspond aux hp_max n'est que supérieur de 15 à ces hp, ce qui aura pour effet que la quantité de soin ne pourra être donné à son plein potentiel.

Par la suite, nous avons utilisé beaucoup d'héritage qui est l'un des points forts de Java et l'une des principales raisons pour laquelle nous avons utiliser ce langage. En-dessous de la classe des Etre_vivants, nous avons fondé deux sous-classes que sont la classe Humain et la classe Monstre.

Monstre

La classe Monstre va donc regrouper tout ce qui est de proche ou de loin liés aux monstres, comme son nom l'indique. Cette classe fille à la classe Etre_vivant sert principalement de paliers à la création des classes pour chaque type de monstres sur lesquels nous nous attarderons plus tard. Cette classe n'a que deux paramètres un paramètre id qui permet de différencier tous les monstres les uns des autres par exemple, deux loups seront distincts par le biais de ce paramètre. En plus de l'id, nous avons le paramètre exp_gagne, celui-ci aura une valeur différente pour chaque type de monstre tué au cours d'un combat. Elle sera, donc, utilisé dans la classe Combat.

Pour en finir avec la classe Monstre, nous avons la fonction calcul_competence qui retourne soit les dégâts, soit le soin, soit le bouclier que le monstre va faire. Ces compétences sont calculées directement avec les statistiques que le monstre possède. Nous avons essayé de rendre les calculs les plus corrects possibles pour éviter que les dégâts soient trop énormes ou bien même que le bouclier soit lui aussi trop grand pour les personnages du joueur puissent faire des dégâts aux monstres.

Humain

La classe Humain n'a pas grand intérêt dans le cas présent car elle est censé être la séparation entre les PNJ (entités non-jouable, par exemple, un villageois) et les PJ (entités jouables, ce que le joueur contrôle) mais étant donné que nous n'avons pas fait la gestion des items, la partie des PNJ n'est donc pas faite. Nous passons directement à l'explication de la classe PJ.

Comme dans la classe Monstre, nous retrouvons des paramètres liés à l'expérience. Mais dans ce cas-ci, les PJ pourront les utiliser et non les distribuer lors de combats. Nous avons trois paramètres en rapport avec l'expérience que sont l'exp, l'exp_limit et le lvl. Les paramètres exp et exp_limit interagissent comme les hp, l'exp est l'expérience actuelle d'un personnage alors que l'exp_limit est l'expérience nécessaire pour pouvoir augmenter de niveau (lvl). La conséquence sera visible par la suite dans les classes des types de PJ.

De nouveau, nous retrouvons la fonction calcul_competence, mais dans ce cas-ci, le calcul n'est pas juste fait sur les statistiques des personnages, le calcul est aussi basé sur les attaques des personnages qui est une map dans cette même classe mais qui est initialisé dans les classes de types de PJ.

Type de Monstre

Pour le projet, nous avons décidé de choisir quatre type de monstres qui ont chacun une statistiques plus forte que les autres. Les quatre type de monstres sont le squelette qui aura comme statistique principale l'armure, le loup sera plus orienté dégâts physique, la sorcière comme bien entendu aura la plus grande magie et finalement, la licorne qui sera celle avec le plus d'esprit.

Chaque classe de type de monstre ne sont pas si différente les unes des autres, les seuls changements se feront sur le constructeur utilisé qui changera la répartition des statistiques.

Type de PJ

Comme pour les types de monstres, peu de choses sont modifiés entre chaque type de PJ.

Contrairement aux monstres, les personnages ont leur statistiques initialisés grâce à la fonction attributionStats, celle-ci est différente selon le type de PJ. Pour donner un peu de hasard à l'attribution des statistiques, nous avons décidé d'utiliser la fonction Math.random entre deux valeurs pour chacun des paramètres.

Ensuite, nous avons créé la fonction levelUp, encore une fois, nous y avons mis un peu d'aléatoire. Le principe de level up est que lorsque le niveau d'un personnage augmente, le personnage voit certaines de ses statistiques augmenter de façon aléatoire. Nous avons donc décidé de simuler un lancer de dé pour pouvoir augmenter les statistiques totales d'un personnage d'un certain montant. Pour faire en sorte que l'augmentation de niveau ne soit pas trop rapide, à chaque fois qu'un personnage augmente de niveau, l'expérience limite est multiplié part la fonction exponentielle (e) soit a peut près 2,7.

Les statistiques qui sont augmentés sont dépendantes de la classe du personnage.

Vu que nous n'avons pas encore dit qu'elle classe existait, nous allons les citer. En premier, nous avons le CaC, soit un personnage qui est orienté dégâts physiques, le Tank qui va avoir plus d'armure que les autres, le Sorcier comme la sorcière aura plus de magie que les autres, et finalement le Prêtre, qui a des affinités la capacité de pouvoir soigner ses alliés.

Pour finir avec les classes du package Etre, il reste une dernière fonction dans les classes de types de PJ, il s'agit de creationAttaques(), cette fonction est ce qui montre le plus l'affinité des personnages par rapport à quelle capacité ils vont utiliser lors d'un combat, par exemple, même si un Sorcier et un Prêtre avait les mêmes statistiques de magie et d'esprit, le Prêtre aura toujours une plus grande capacité de Soin que le Mage et le Mage fera plus de dégâts magiques que le Prêtre.

Combat

Maintenant, que le package Etre est passé, le plus gros package de la partie théorique reste à décrire. Le package Combat n'a qu'une seule classe mais pas des moindres car c'est autour de cette classe que notre projet se repose. La classe Combat n'a que deux paramètres qui sont deux LinkedList pour regrouper les alliés (personnage du joueur) et les ennemis (monstres rencontrés).

Il y a donc deux fonctions pour ajouter/enlever des éléments des listes.

Les fonctions suivantes seront les plus importantes dans la partie théorique.

Premièrement, on va utiliser plusieurs boucles pour pouvoir continuer le combat tant que l'un des deux côtés n'a pas été battu. La première boucle globale regarde si les listes sont vides ou pas.

A l'intérieur de cette grande boucle, on y retrouve deux boucles pour faire jouer le joueur et faire tourner les monstres. Dans le tour du joueur, on demande au joueur quelle capacité il veut utiliser avec quel personnage. Tout dépendant de quelle capacité utilisée, une fonction différente va être utilisé. Dans le cas d'une attaque, nous allons nous diriger vers la fonction deroulementCombatAttaque(J/M), cette fonction va calculer les dégâts infligés à un monstre ou joueur. Les dégâts sont calculés par rapport aux fonctions calcul_competence vu précédemment, et par rapport à l'armure s'il s'agit d'une attaque physique, ou à l'esprit, s'il s'agit d'une attaque magique. Dans le cas où l'armure ou l'esprit est supérieur à l'attaque ou la magie de façon respective, le cas des dégâts est divisés par 2, pour rendre les statistiques défensives plus importantes que juste faire un bouclier. A ne pas oublier, si la personne ciblée a un bouclier, alors la valeur valeur du bouclier va être soustraite aux dégâts normalement infligés. Si la capacité utilisé est un soin, on va se diriger vers les fonctions deroulementCombatSoin(J/M) qui va soigner l'allié choisit jusqu'à la limite des hp_max de la cible. Finalement, si la capacité choisit est un bouclier, le personnage ayant utilisé cette capacité va voir augmenter son bouclier en fonction de son armure et de son esprit.

Partie Graphique

Dans cette partie, nous traiterons de l'évolution de la partie graphique du projet allant de la conception des images à l'animation des personnages. Tout d'abord, nous verrons la partie déplacement sur une carte pour, ensuite, passer à la gestion d'un combat.

Design

Tout d'abord, avant parler du code, nous avons dû mettre en place les différents designs et sprites que l'on utilisera tout au long de l'élaboration.

Pour cela nous avons utilisé le logiciel RPG-Maker qui permet de créer des maps et des personnages que l'on peut exporter ensuite en png pour avoir les sprites.

Pour les sprites des monstres et les maps de combat, nous avons choisi d'utiliser des sprites trouvés sur Internet (ce que l'on ne ferait pas normalement, pour cause de droit d'auteur).

Enfin pour tout ce qui concerne l'interface, nous avons tout fait nous-même avec deux logiciels gratuits : Paint et Photopea.

Code

Ensuite, nous avons commencé par établir les différentes classes.

MyFantasy

MyFantasy sert à exécuter tout le programme en utilisant Javafx, elle est, donc, héritée de la classe Application qui gère l'exécution. Ainsi, il y a plusieurs fonctions principales qui permettent cela.

La fonction main qui lance le programme avec *launch(args) ;*

La fonction start qui permet de setup les différentes informations au moment du lancement tel que la taille de la fenêtre (ici 816*624), le lieux d'exécution en prenant en paramètre un Stage qui nous sert de base auquel nous rajoutons un titre MyFantasy et une Icône.

Ensuite, nous pouvons lancé au choix afficheMap() qui permet de lancer la phase exploration et afficheCombat() qui permet de lancer une phase de combat.

La fonction afficheMap() permet de gérer une phase d'exploration, pour cela nous utilisons la map001, qui correspond au village.

Pour cela, nous devons définir une image qui est créée en ouvrant un fichier à partir du chemin du répertoire plus le chemin jusqu'au fichier. Ensuite, ce fichier est transcrit en format URI qui est transformé en String ce qui sert à initialiser l'image.

Une fois l'image ouverte, on peut modifier différents paramètres tel que la position ou bien la taille de l'image.

Enfin, on crée une ImageView, prenant en paramètre l'image précédemment créée, que l'on ajoute à root pour qu'elle s'affiche.

Nous utilisons une imageView car root ne prend en compte que les classes étant filles de la classe node, or, image n'est pas fille de la classe node alors que ImageView oui. Cette méthode sera réutilisée pour charger une image.

Ensuite, nous avons mis en place un curseur qui s'affiche lorsque la souris rentre dans la fenêtre.

Puis, nous avons mis en place les éléments qui nous sont utile.

Avec placeCase() qui crée des rectangles invisibles de taille 48*48 pixel qui constitue l'entièreté de la fenêtre ce qui pourrait servir à savoir si le personnage peut ou non marcher sur certaines cases, toutes ces cases sont rangées dans un tableau.

Puis, avec ajoutPersonne() on crée un pion de type PersoImg ayant pour localisation une case de la map. C'est ce pion qui sera contrôlé par le joueur.

Pour finir, nous mettons en place la gestion des évènements donc lorsque le joueur appuie sur une touche cela lance la fonction animation() qui prend en paramètre le caractère de la touche pressée et la fonction stopAnim() lorsque la touche est relâchée.

Les dernières lignes servent à dire que la fenêtre ne peut pas être redimensionnée et a enregistrée la scène dans la fenêtre et de l'afficher.

La fonction animation() lance l'animation en appelant la fonction lancerAnim, de l'objet Anim animation instancié précédemment, prenant en paramètre un String qui correspond à la destination/touche pressée.

Au début, nous n'utilisions pas la classe Anim qui est héritée de la classe transition. Nous utilisions un switch où, pour chaque cas, nous enlevions l'image de root pour replacer la nouvelle mais cela soulevait 2 problèmes :

-le premier étant le fait que l'on ne peut pas retirer une image affichée donc les sprites du personnage se chevauchaient .

-le deuxième est le framerate, la capture n'est pas définie selon un cycle donc la vitesse du perso correspond au taux de capture de la touche.

Pour remédier à ces problèmes, nous avons décidé d'utiliser une librairie supplémentaire appelée Slick2D qui est faite pour le jeu vidéo. Malheureusement, cela fut un échec car les logiciels auxiliaires permettant la création des maps (tiled map) fournissait des maps au format trop récent pour la librairie.

Une troisième option était d'utiliser les deux librairies Javafx et Slick2d en même temps mais les deux étaient incompatibles.

Pour finir, nous avons utilisé la solution qui est de créer la classe Anim qui est héritiée de la classe transition, ce qui nous permet d'avoir la main sur tous les éléments de l'animation.

Puis au lieu de continuer sur l'amélioration de la phase d'exploration en ne permettant pas le déplacement dans certaines cases, nous avons décidé de passer à l'élaboration d'une phase de combat qui nous permettrai de mettre en commun nos différents travaux.

Pour cela, nous avons créé la fonction afficheCombat() qui comme afficheMap() génère l'interface d'un combat.

En commençant par charger l'image de fond. Puis on initialise les 3 tableaux de boutons qui servent à répertorier les éléments cliquable de l'interface.

On setup phase=0 ce qui correspond à l'étape de la sélection (0 = sélection du héros) et l'interface avec afficheUI().

Ensuite, on capte lorsque la souris est relâchée et on effectue une Action() en conséquence.

afficheUI() permet d'afficher l'interface et de créer les différents boutons que l'on ajoute aux tableaux respectif.

Action() est une fonction qui rend les boutons cliquables en fonction de la phase et enregistre les informations. Si l'on est dans la phase 0 on doit choisir le personnage qui fera l'action ce qui le set en occupé, la phase 1 permet de sélectionner l'action faites et selon l'action cela lance soit aucune phase, soit la phase 2 qui correspond au choix de l'ennemie à attaquer, soit la phase 3 qui correspond au choix du héros à soigner.

PersoImg

Cette classe sert à créer une image de personnage dans le cadre de l'exploration d'une map.

Son constructeur prend en paramètre le nom du personnage qui sera localisé à la première case, ou bien, prend le nom et la case mais dans les deux cas le nom permet de récupérer le sprite de base du personnage.

Les autres fonctions servent à avoir d'autre image associée au personnage mais elles ne sont pas toutes utiles.

Les classes d'animation

Toutes les classes d'animation marchent toutes de la même manière, nous avons une ImageView qui correspond au sprite, un count qui permet de savoir en combien d'étapes s'effectue l'animation.

Le constructeur permet de set les variables mais aussi la durée d'un et d'utiliser la fonction setInterpolator() ce qui lance la fonction Interpolate() tous les cycles.

La fonction Interpolate() est la fonction la plus importante car c'est dans celle-ci que l'on va pouvoir changer le sprite, pour cela, il faut juste "remplacer" l'image en utilisant setViewport qui "découpe" l'image selon un rectangle et comme l'on "remplace" l'image elle change directement dans root car en Java les objets sont gérés par des pointeurs, on change donc juste la valeur en mémoire.

Enfin la dernière fonction importante et la fonction lancerAnim() qui lance l'animation en utilisant la fonction play(). Pour arrêter l'animation, il suffit d'utiliser la fonction stop().

Pour finir, les dernières classes de la partie graphique sont les classes bouton.

Ces classes sont des classes que nous avons créé car les classes button prévu en Javafx ne permettait pas de personnaliser suffisamment les éléments.

Bouton

Donc, pour la gestion de la classe Bouton nous avons l'ImageView qui affiche l'image qui est de base invisible, enter correspond à l'image lorsque la souris entre dans le bouton alors que pressed lorsque un clic souris est détecté.

Ensuite, nous avons un objet text qui représente le texte associé au bouton dans le cadre qu'un bouton d'action ce serai par exemple « Attaque »

Dans le constructeur, nous avons l'initialisation de tous ces éléments mais aussi leur ajout dans root qui doit être placé en paramètre.

Nous avons à la fin le lancement de la fonction mouseEvent()

Cette dernière sert à détecter les évènements de la souris lié à ce bouton, par exemple la souris qui passe sur le bouton, pour être redirigé vers la fonction opportun.

BoutonEtre

Cette classe est une classe fille de la classe Bouton mais qui représente des êtres vivants alors l'on rajoute 4 paramètres :

-une ImageView pointeur qui servira à afficher un pointeur au-dessus de l'être sélectionné

-un String nom qui permet de savoir le nom exact

-Deux booléens vivants qui permet de savoir si l'être est encore vivant et occ qui permet de savoir s'il est occupé

Cette classe est, donc, abstraite car on ne peut pas créer un être on peut créer soit un monstre soit un personnage.

BoutonMonstre

Cette classe est héritiée de BoutonEtre et ressemble un peu dans le fonctionnement à la classe bouton sauf que ici lorsque la souris passe au dessus d'un monstre un pointeur s'affiche, le text n'est pas affiché et lorsque le monstre meurt il ne peut plus être sélectionné.

BoutonPerso

Cette classe est aussi héritière de BoutonEtre sauf qu'elle doit prendre en compte la gestion des points de vie et des points de magie qui sont affichés en bas à droite de l'interface donc nous devons utiliser plus d'éléments visuels.

Nous avons fait le tour des classes concernant l'aspect graphique même si nous aurions pu séparer la classe MyFantasy en trois classes : l'une s'occuperait de l'exploration l'autre du combat et la dernière de l'exécution global.