Mathys LIOSON
Fazil MAHAMMAD

IUT CLERMONT - FD
UNIVERSITÉ
Clermont Auvergne

PROJET HONOR KNIGHT

SOMMAIRE :

Table des matières

Introduction

Dans son village le chevalier apprend que son père, le chevalier doré, a été vaincu par le grand chevalier-dragon, un tyran qui n'hésite pas à faire du mal pour dominer le monde.
Pour se venger, il va devoir se former, battre des ennemis féroces, récupérer l'épée de son père pour enfin affronter le chevalier dragon.

Nous allons réaliser un jeu d'aventure du genre plateformer où il y aura de l'aventure et du combat. Nous allons coder ce jeu en JavaFX et aussi avec du FXML en binôme.

Les objectifs de ce projet sont de nous initier au JavaFX afin de voir les bases et les fonctionnements de ce langage.

Présentation du projet

Ce jeu sera un plateformer 2D d'aventures avec du combat.
Au début on commencera dans une plaine. Le joueur devra pouvoir se battre contre des ennemis. Le personnage pourra se déplacer de droite à gauche et pourras sauter, il faut donc gérer un système de gravité. Il pourra aussi monter pour atteindre des plateformes en hauteur.
Il possède une barre de vie qui diminue s'il se fait toucher par un ennemi.
Pour l'instant il n'y aura qu'une seule carte où lieu mais d'autres lieux peuvent être ajoutés plus tard.
On aura ainsi un menu de démarrage avec des paramètres à régler comme le son, les touches.

Contexte

Honor Knight est un jeu de plateforme en 2D ou l'on incarne un chevalier qui dois avancer dans un monde hostile ou il devra surmonter les obstacle comme de redoutable serpent venimeux.

Fonctionnalité

Le personnage sera animé il pourra se déplacer de droite à gauche et pourra aussi sauter. Il possédera une arme qui sera animé. L'arrière-plan se déplacera au fil de l'avancement du joueur, le joueur sera donc au centre de l'écran, c'est à dire que s'il se déplace à droite, l'image des fils vers la gauche et inversement.
Il y aura plusieurs classes comme la classe personne, objet, carte, …

Résumé en Anglais

Honor Knight is a 2D platform game where you play a knight who must advance in a hostile world where he will have to overcome the obstacle like a fearsome poisonous snake.

Description du diagramme de classe

Tout d'abord, nous avons la classe personne. Elle contient « spriteAttaque » qui permet de changer l'image de l'attaque du joueur et « spriteNumAttaque », un compteur qui permet aux images de l'attaque de changer.
Nous avons la même chose pour le compteur de la direction « statique », mais aussi pour le déplacement avec respectivement « spriteCounter » et « spriteCounterStatic » puis « spriteNum » et « spriteNumStatic ».

Nous avons la classe joueur, qui contient un booléen qui permet de savoir si le joueur est en état de saut. Un point 2D qui permet de situer le joueur dans l'espace et les variables x et y permet de le construire à l'endroit souhaité. Nous avons aussi la hauteur et la largeur pour définir sa taille.
Nous retrouvons les points de vie avec « hp », le nombre de pièces ainsi que « nbsprite » qui va permettre de gérer l'animation de l'attaque.
Nous avons le délai de l'attaque qui vaut par défaut 1, et un booléen pour l'état de l'attaque.
Nous avons aussi une chaîne de caractères qui va définir la direction de déplacement et aussi la direction de l'attaque.
Ensuite, nous définissons toutes les images pour les déplacements, pour l'animation quand le personnage est statique, pour le saut, pour les nombres de cœurs, pour les animations de l'attaque à droite et aussi à gauche.
Le constructeur du joueur, permets de le créer à un endroit voulu dans le monde un joueur avec une hauteur et une largeur.
La méthode « gainPiece » permet de gagner une pièce et « getStrPieces » retourner cette valeur en chaîne de caractères.
La méthode « prendreDegat » va nous permettre de diminuer les points de vie de notre joueur quand notre joueur est attaqué.
La méthode « movePlayerX » permet a la vue du joueur de se déplacer sur l'axe x et la méthode « movePlayerY » permet se déplacer sur l'axe y de haut en bas mais due à la gravité.
Le déplacement du joueur se fait avec les touches du clavier. À chaque fois qu'on appuie sur une touche du clavier, on l'associe à une direction et ceci fait changer l'image du joueur pour réaliser l'animation et le déplace avec la méthode « movePlayerX ».
Si on ne Touche pas le clavier, le joueur lance une animation lorsqu'il est statique.
La méthode « isPressed » permet de retourner la touche pressée.
La méthode « jumpPlayer » permet au joueur de sauter, c'est à dire de donner une impulsion pour le saut.
La gravité s'applique au joueur avec la méthode gravite. La méthode observer permet de voir le joueur au centre lorsqu'on se déplace au niveau des x ou des y.
La méthode attaque permet de réaliser une animation sur l'attaque en fonction de la direction.

Nous avons aussi la classe ennemi qui contient un booléen pour savoir s'il est en état de saut, un point 2D qui permet de situer l'ennemi dans l'espace, x et y pour le construire à l'endroit souhaité. On a aussi la hauteur et la largeur pour définir sa définition.
On a un booléen pour savoir si l'ennemi et en vie.

On a aussi une chaîne de caractères qui va définir la direction et aussi la direction de l'attaque. Ensuite on définit toutes les images pour les déplacements à gauche, à droite.
Le constructeur de l'ennemi va permettre de le créer à un endroit voulu dans le monde avec une hauteur et une largeur voulue.
La méthode « moveEnnemiY » permet à la vue de ennemi de se déplacer et aussi, permet de rester au sol due à la gravité.
Le déplacement de l'ennemi se fait automatiquement de gauche à droite. On associe à une direction la vue automatiquement et ceci fait changer l'image de l'ennemi pour réaliser l'animation et se déplace avec la méthode « moveEnnemiX ». La gravité s'applique à l'ennemi avec la méthode « gravite ».
La méthode « isAlive » permet de voir si l'ennemi est en vie et « killEnnemi » permet de passer la variable alive à false.

Dans notre modèle nous avons la classe « Map1 » qui crée un tableau qui contient des caractères qui vont être traduits par des blocs physiques plus tard dans la classe « monde ».
On retrouve ensuite la classe créateur qui va nous créer des textures à partir de la position x et y, c'est de la taille w et h et de l'image pour le créer dans le monde.

Dans la classe « VueJoueur » on retrouve le constructeur du joueur qui permet d'ajouter les images par-dessus le joueur. On a aussi les images pour les points de vie.
La vue du joueur est construite selon x et y avec une hauteur et une largeur qui est récupéré par polymorphisme puisque notre vue du joueur était étendue du joueur.
On initialise la vue du joueur avec ces images de base, ont affecté la taille les différentes images et on l'ajoute au « gameroot » qui va nous afficher la vue du joueur et au « uiroot » tu vas nous afficher la barre de vie.
On retrouve la méthode de l'animation de l'attaque du joueur qui selon la direction empruntée par le joueur va adapter les images à afficher lors d'une attaque.
On retrouve aussi l'animation du joueur qui réalisent la même chose que l'animation de l'attaque mais celle-ci permet de changer les images lors du déplacement du joueur ou quand il ne bouge pas.
On retrouve aussi la méthode « healthRefresh » qui vient appliquer la bonne image pour mettre à jour la barre de vie.

Dans la vue de l'ennemi on retrouve les mêmes méthodes que celle de la vue du joueur sans l'animation de l'attaque et mise à jour de la barre de vie.

Dans nos vues, nous retrouvons la classe Monde qui va nous servir pour réaliser l'affichage de la carte, des personnages, des ennemis, etc.
Dans la méthode d'initialisation, pour tous les caractères contenus dans Map1, nous créons une plateforme avec la méthode « CreerTexture » en utilisant la classe Créateur.
On va aussi fixer la taille de notre background et charger en FXML le label pour les pièces.
On rajoute ainsi toutes ces plateformes a « approot » ainsi que le background, le label des pièces et « uiroot » que contient la barre de vie.

La classe « TitreMenu » permet de créer un titre pour notre menu.

Ensuite on retrouve la classe « MenuItem » qui permet de réaliser la mise en forme de nos items qui seront à l'intérieur de notre « MenuBox ». Ces items vont être des boutons qui auront des animations en fonction des actions que l'on va réaliser dessus.
Pour ce qui est des menus box, cette classe va stocker en colonne les items créer avec le « MenuItem » et on crée aussi une séparation entre les différents items.
Notre classe « MainMenu » va nous permettre d'afficher en tant que scène notre menu qui est créé à partir de la classe « createContent » dans lequel ont créé les différents éléments de notre menu qu'on va vouloir afficher avec le « MainMenu ».

On va lancer le menu Game over lorsque le joueur perd tous les cœurs avec la classe « gameOverMenu ». Ceci lui permettra soit de revenir au début du jeu soit de retourner à l'écran principal, c'est à dire le menu d'accueil.

Dans la classe « GamePanel », on va gérer les vues et les modèles de notre jeu. C'est là qu'on va créer le monde, le joueur et les ennemis et les faire apparaître dans notre monde. C'est aussi ici qu'on va gérer l'apparition et la disparition des pièces, la réapparition du joueur et la collision entre l'attaque du joueur et un ennemi.

Nous créons le « GamePanel » dans « MainGame ». « MainGame » créer un « Thread » qui fera office de boucle de jeu avec la méthode « run » qui tourne en continu. C'est aussi ici qu'on lance la musique de notre jeu ainsi que la fenêtre.

Enfin, on a la classe « ControllerParametre » qui va nous permettre de gérer notre menu paramètres en FXML.

Conclusion

Nous n'avons pas pu faire tout ce qu'on avait prévu à la base de notre projet car nous avons prévu plein d'autre ajout comme de nouveau ennemis, de nouvelles armes, de nouvelle carte, des checkpoint etc.
Néanmoins, se fut une bonne expérience, avec un bonne coalition de groupe et un bon travail en commun.