École Polytechnique de Bruxelles

INFO-H-200 : Programmation orientée objet

Rapport projet d'informatique : Dungeon Game

Ibrahim TAOUAOU
Mohamed SIGHAOUI

14 Mai 2017

1 Introduction

Ce projet donné en Ba2 polytechnique dans le cadre du cours d'informatique (INFO-H-200) consiste en la création d'un jeu de type donjon en java en utilisant les concepts de la programmation orientée objet vus aux cours durant l'année.
Mis à part la musique provenant du jeu mobile Endless Frontier ainsi que les images des blocks, des coffres et de l'argent, tout a été fait par le groupe grâce à divers logiciels.

2 Brève description du jeu

Le concept du jeu est simple : choisir une classe (guerrier ou sorcier) et entrer dans l'arène dans le but d'atteindre l'étage le plus haut sans mourir. Pour ouvrir la porte de passage de niveau, il faut tuer les ennemis. Le joueur a un inventaire avec lequel il peut intéragir (utiliser et vendre) et peut trouver des objets à ramasser durant son périple. Il dispose également d'un système d'équipement lui permettant de changer son équipement (plastron, gants, bottes, casque, collier, bague, arme).

3 Affichage graphique

Lors du lancement du jeu, le joueur aura le choix entre quatre boutons : *Start* qui lance le jeu à partir de la sauvegarde trouvée (si le jeu n'en trouve pas, ce bouton n'apparaît pas), *New Game* qui lance une nouvelle partie, *Shop* qui permet au joueur d'acheter des objets beaucoup plus puissants que ceux trouvés en jeu, *Help* qui donne les informations nécessaires au bon déroulement du jeu et enfin *Exit* pour quitter.
Lorsque le joueur choisit de lancer une partie, il a alors le choix entre un guerrier ou un sorcier ainsi que le choix d'une taille d'arène. En jeu, la fenêtre est divisée en deux :

- La partie basse qui dessine l'avancée du jeu.

- La partie haute qui reprend toutes les informations : la vie du joueur, la vie de l'ennemi (elle apparaît lorsque le joueur s'en approche), l'état de la compétence, l'attaque et la défense du joueur et de l'ennemi, l'inventaire, le niveau actuel ainsi que l'argent économisé.

FIGURE 1 – Menu

FIGURE 2 – Ecran de jeu

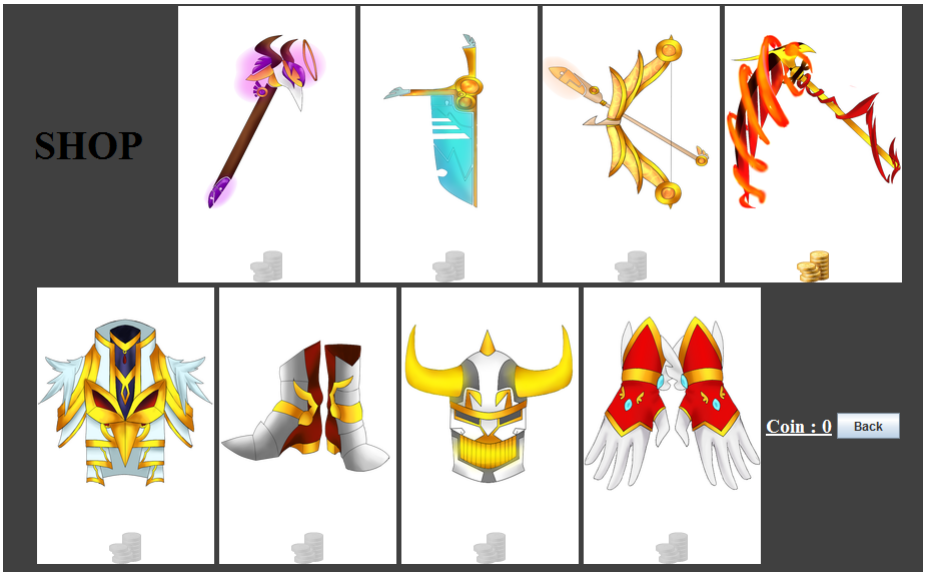

FIGURE 3 – Boutique

En jeu, une barre de menu donne la possibilité au joueur d'ouvrir la gestion d'équipement ou un bref récapitulatif des touches possibles.

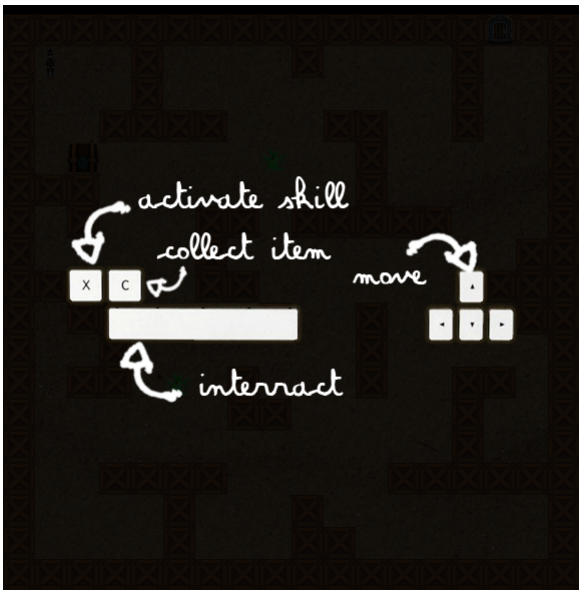

FIGURE 4 – Rappel des touches

A partir de l'inventaire, le joueur peut passer la souris sur un item pour voir ses caractéristiques ou appuyer dessus pour interagir avec lui.

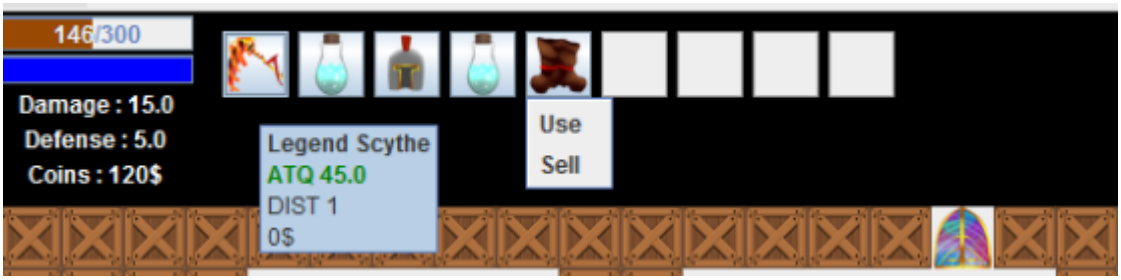

FIGURE 5 – Inventaire : interactions

4 Fonctionnalités implémentées

4.1 Affichage

— Boutique
— Barre de vie et informations du joueur
— Barre de vie et informations du dernier ennemi croisé
— Barre d'état de la compétence du joueur
— Equipement
— Inventaire dynamique
— GameOver dynamique

4.2 Joueur

— Gestion d'équipement
— Gestion d'inventaire à taille fixe
— Attaque au corps à corps
— Attaque à distance
— Compétence spéciale selon la classe du joueur

4.3 Ennemi

— Compétence spéciale du boss
— Caractéristiques augmentées au fur et à mesure de l'avancée du jeu

4.4 Bonus

— Potion de vie instantanée
— Portail téléportant 3 niveaux plus loin

4.5 Mapping

— Arène générée aléatoirement parmi 4 matrices pouvant subir plusieurs rotations et symétries
— Positions des personnages et objets générées aléatoirement sur la map

5 Description de l'architecture du jeu

Le programme se base sur la méthode de conception *MVC* (Model-View-Controller) permettant la séparation du code en 3 packages distincts. Les principales avantages de ce design pattern est la lisibilité et la facilitation des modifications et des maintenances du code.

5.1 Model

Le package *Model* regroupe les classes permettant l'instanciation de tous les objets nécessaires. Toutes les données du jeu y sont regroupées.

5.2 View

Comme son nom l'indique, ce package s'occupe de l'affichage afin de simplifier l'interaction humain-machine. Ce modèle prend tout ce dont il a besoin dans *Model* sans pour autant y apporter de modification.

5.3 Controler

Ce package sert "d'écoute" afin de "traduire" les actions de l'utilisateur et de demander à *Model* ou *View* d'apporter les modifications demandées.