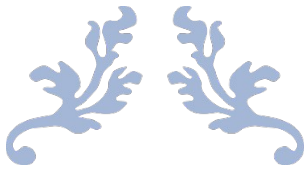
Université de Strasbourg
UFR de mathématiques et d'informatique



Programmation orientée objet 2

Rapport du Projet : Jeu d'explorations de salles

26 décembre 2022

Auteurs :
GHAMAT Seyedehsetayesh
FARID Yasmina

Introduction

Contexte :

Dans le cadre de l'UE de Programmation Orientée Objet 2 nous étions amenés à concevoir un projet en langage Java d'exploration de salle qu'on a intituler **Cave Exploration**.

Le jeu consiste déplacer un joueur ayant des points de vie et d'attaque de page en page en faisant face aux différents monstres et pièges tout en ayant a sa disposition des objets qu'il peut collecter et utiliser en explorant les différentes salles de la page jusqu'à en arriver à la sortie et passer à la page suivante. Une fois toutes les pages passées le joueur gagne.

Répartition des taches :

Durant la conception de ce projet notre méthode de travail consistait à réfléchir et discuter ensemble des algorithmes optimaux pour l'implémentation des fonctions qui gèrent l'évolution du jeu, l'écriture du code a été faite alternativement par chacun de nous pour chaque partie de fonction implémentée. Cette méthode de travail nous a permis de gagner du temps dans la réalisation du projet vu que à chaque fois que l'une de nous avait une meilleure compréhension du concept à implémenter elle s'occupait de celle-ci tandis que l'autre prend en charge la prochaine tâche.

Conception

UML :

en fichier pdf joint avec le projet.

Modélisation :

Initialement, nous avons pensé aux différentes classes a créé pour le jeu, on les a ensuite mis dans des catégories selon le rôle dans le programme ainsi nous avons décidé de créer trois packages :

Package GameObjects : contient les classes représentants les objets du jeu

- **Gameobject** qui est une classe abstraite parente de toutes les autres
- **ObjectID** : une énumération des différentes classes (objets du jeu) pour faciliter la reconnaissance de ceux-là par la classe abstraite **GameObject.**

Objets joueurs :

- **Wizard** : notre joueur
- **Mana** : une potion qui donne des points de magie (d'attaque) au joueur une fois collectée.
- Potion : une potion qui donne des points de vie au joueur une fois pris collectée.
- **FireBall** : une boule de feu que le joueur lance sur les ennemis pour les tuer, à chaque boule de feu lancé il perd des points d'attaque.

Objets ennemis :

- **Enemies**: les adversaires du joueur qui bougent dans l'image, certains bougent sur les cases qui ne sont pas des murs et d'autres peuvent traverser les murs.si le joueur entre en collision avec eux il perd des points de vie
- **Spikes** : des pièges sous forme de clou qui une fois le joueur marche dessus il meurt instantanément,

Package MainGame : contient les classes qui gèrent la partie fonctionnelle du jeu

- **Constants** : permet de récupérer les éléments de chaque niveau de jeu : la largeur, hauteur de l'image et des éléments
- **Spritesheet** : permet de récupérer les images des objets du jeu en répartissant une image en plusieurs sous image selon l'état de l'objet pour rendre les objets dynamiques graphiquement.
- **Keyinput** : gère les entrées clavier

- **MouseInput** : gère les actions de la souris
- **GameHandler:** gère les actions à exécuter reçu par la classe **Game,** sert d'organisateur du déroulement du jeu en étant à jour avec toutes les actions reçues par **Game.**
- **Game** : la classe ayant le rôle fonctionnel le plus important, elle crée le jeu initial avec les objets du jeu, charge les niveaux, c'est elle qui start et stop le jeu selon l'état de celui-ci. Elle sert de lien entre les objets, le déroulement du jeu et la partie graphique pour mettre a jour l'affichage selon les actions.

Package UI : contient les classes gérant la partie interface graphique pour le joueur

- **GameMenu** : crée une fenêtre initiale avant le commencement du jeu, un GameMenu pour choisir de jouer ou de quitter
- **ImageLoader** : charge une image depuis un chemin de fichier
- **GameWin :** crée une fenêtre qui affiche au joueur qu'il a gagné et lui demande s'il veut rejouer ou quitter
- **GameOver** : crée une fenêtre qui affiche au joueur qu'il a perdu et lui demande s'il veut rejouer ou quitter
- **Camera :** permet de suivre les coordonnées des objets

Une classe **App** qui contient le main elle permet de lancer le programme en instanciant **GameMenu.**

Implémentation

La classe **App** contient le main qui instancie **GameMenu.**

Dans la classe GameMenu :

- Instancie **Constants** qui contient la méthode loadconfig() qui initialise préalablement les éléments constants de la fenêtre du jeu (le titre, les fichiers de niveaux (même s'ils ne sont utilisé que si on décide de jouer), la hauteur largeur de la fenêtre et des éléments pour créer la fenêtre.

- Crée la fenêtre principale avec JFrame avec les éléments reçus par loadconfig de la classe **Constants**, elle affiche un GameMenu avec deux

boutons : Start pour commencer le jeu et Quit pour quitter et ajoute des ‘eventlisteners’ sur ceux la.

- Dans le cas où le bouton Start est cliqué elle appelle dispose() qui free cette fenêtre puis instancie la classe **Game,** si le bouton Quit est cliqué on free la fenêtre et on appelle System.exit(0) pour mettre fin au programme.

Dans la classe Game :

- **Game** étends **Canvas qu’on** a utilisé pour faciliter le dessin des éléments dans la fenêtre graphique, et elle implémente l’interface **Runnable** pour implémenter run () qui va gérer les actions du jeu selon le temps avec les evenements reçu et le temps calculé avec les methodes de la classe System pour alterner proprement les actions du jeu.

- Elle instancie **Window** qui construit une fenêtre avec les paramètres : hauteur,largeur,titre et game (elle-même avec this) en utilisant les configurations de **Constants.**

- Elle appelle la méthode start qui met l'attribut isRunning a true et crée un thread et le lance, permettant d’améliorer la performance. Si le programme reçoit une exception qui a l’interrompt elle appelle printStackTrace().

- Elle instancie **GameHandler** qui contient les méthodes pour gérer le jeu avec une liste de **GameObject** ou elle peut ajouter, supprimer et mettre a jour l’emplacement des objets puis envoyer à **Game** les changements (notamment ceux du niveau au cas où le joueur a réussi un niveau) afin qu’il affiche le nouvel état du jeu.

- Elle instancie **ImageLoader** qui permet de charger les images des objets du jeu pour en créer des **SpriteSheet** (nous avons fait des recherches sur les Sprite Sheets et appris comment les utiliser afin d’avoir un rendu graphique dynamique qui mettra en valeur l’interface graphique.

- On charge les niveaux (les pages) du jeu qu’on a préalablement crée et mis dans le répertoire lib/maps en utilisant la méthode loadImage.

- Une fois les objets et les niveaux chargés dans les attributs de notre objet Game, on appelle loadLevel qui prend l’image et place les éléments du niveau dans notre image selon la couleur de la case on détermine quel objet sera dans la case de l’image, en ajoutant chaque objet dans la liste d'objets dans **GameHandler** pour qu’elle gère les actions du jeu.

Dans les classes objets du jeu :

- Toutes les classes héritent de la classe **GameObject** des coordonnées x,y(coordonnées),h,w(hauteur, largeur de l'objet),velX,velY(vélocité) et les méthodes à écrire pour chacune des classes selon ses propres actions prédéfinies, pour le joueur **Wizard.**

- tick() : détermine l'action a executer selon le **GameHandler**, si c'est le joueur il bouge selon le keyboard input et attaque selon le mouse input.

- render(g) : dessine l'image du joueur et les deux barres en haut de la fenêtre représentant ces points de vie et les points d'attaque.

- getbounds() ; crée un **Rectangle** qui définit sa zone d'action pour tester s'il y'a une collision avec la zone d'un autre objet pour l'affecter.

- Pour le joueur **Wizard** on a une méthode collision () qui teste l'intersection du joueur avec avec les autres objets du jeu (Block,Mana,Enemy,Potion,WinTile,Spikes) et effectue l'action requise (perdre des points de vie/d'attaque, gagner des points de vie/d'attaque,mourir etc)

Les fonctionnalités bonus :

- L'utilisation des spritesheets pour la partie graphique afin d'avoir des éléments dynamiques qui changent d'apparence selon l'état.

- L'objet d'attaque Fireball qui suit la trajectoire du joueur jusqu'au point où on a reçu le clic de la souris et cause des dégâts à l'ennemi.

Utilisation du jeu

- Lancer java -jar Cave-Exploration.jar dans le terminal
- Bouger le joueur avec les touches Z,Q,S,D respectivement haut, gauche, bas, droite.
- Cliquer sur un point dans la fenêtre du jeu pour lancer une boule de feu qui suivra une trajectoire vers ce point. (Ciblez les monstres pour les attaquer).
- Pour pouvoir prendre les potions (de vie ou de mana) il faut avoir moins que la moitié des points attribués au départ.
- Si vous perdez cliquer simplement sur restart et vous rejouerez depuis le premier niveau.

Conclusion

Ce projet a été très enrichissant car nous avons pu réaliser un certain nombre de recherches sur les différentes notions de programmation orientée objet et trouver des méthodes efficace pour programmer en Java.