Art Gallery game

ALLANO Hugo

1431
Université de Poitiers

Table des matières

1. Introduction

Contexte et objectifs du projet

Présentation du projet :

Le projet consiste à créer un jeu d'aventure textuel inspiré de *Colossal Cave Adventure*, l'un des premiers jeux d'aventure interactifs de l'histoire du jeu vidéo. Ce type de jeu fonctionne principalement par des échanges textuels, où le joueur interagit avec le programme en entrant des commandes par le biais d'une interface en ligne de commande. Le programme réagit aux actions du joueur en affichant des descriptions textuelles des situations, des lieux, des objets ou des personnages présents dans le jeu.

Dans ce projet, l'objectif est de concevoir et de développer un jeu où le joueur incarne un héros qui explore un univers composé de différentes *Locations* (lieux), interagit avec des *Items* (objets) et des *Characters* (personnages), et traverse des *Exits* (sorties) pour avancer dans l'aventure. Le jeu doit être suffisamment interactif pour permettre au joueur de prendre des décisions qui influenceront le déroulement de l'histoire.

Objectifs à atteindre :

1. Conception d'un univers interactif : Développer un environnement riche où le joueur peut se déplacer entre différentes *Locations*, interagir avec des objets et des personnages, et résoudre des énigmes pour progresser dans le jeu.
2. Implémentation d'un système de commandes : Le joueur doit pouvoir interagir avec le jeu via une série de commandes comme *GO*, *LOOK*, *TAKE*, *USE*, *ATTACK*, etc. Ces commandes doivent être interprétées et exécutées correctement par le programme.
3. Création d'un système d'inventaire et d'interaction avec les objets : Le héros doit être capable de collecter des objets, les utiliser avec d'autres éléments du jeu

(par exemple, utiliser une clé pour ouvrir une porte), et interagir avec des personnages ou des objets.

4. Gestion de l'état du jeu : Le jeu doit suivre l'évolution de l'état du joueur, notamment sa position, ses objets, sa santé et d'autres éléments qui pourraient influencer le gameplay. Un système de "fin de jeu" doit être en place (par exemple, "Game Over" ou "You Win").
5. Extensibilité du code : Le code doit être conçu de manière modulaire pour permettre d'ajouter facilement de nouvelles fonctionnalités, commandes et éléments de jeu.

Outils utilisés :

- Langage de programmation : Java, car il permet de structurer efficacement le code et est un langage largement enseigné dans les cursus informatiques. Il offre également des fonctionnalités permettant de gérer l'entrée/sortie, la gestion des objets et des collections (par exemple, les *Maps* pour gérer les *Exits*).
- Environnement de développement : Visual Studio Code, un environnement de développement populaire pour les projets Java, offrant des outils puissants de débogage et de gestion de projets.
- Contrôle de version : Git et GitHub, utilisés pour gérer l'historique des versions du code et faciliter la collaboration entre les membres du groupe.
- JUnit pour les tests unitaires, afin de s'assurer que les fonctionnalités du jeu sont correctement implémentées et que le programme fonctionne comme prévu.

Durée du projet :

Le projet a été réalisé dans le cadre d'un examen de licence en informatique. La durée totale du projet était de 4 semaines, réparties comme suit :

- **Semaine 1** :

  Planification et conception de l'architecture du jeu, définition des classes principales (Locations, Exits, Items, etc.), établissement des règles du jeu et des commandes.

  Développement des premières fonctionnalités du jeu, y compris la gestion des Locations, des Exits et des commandes de base (*GO*, *LOOK*, etc.).

- **Semaine 2 :**

  Implémentation des objets, de l'inventaire du héros et des interactions possibles avec les objets et les personnages, ainsi que les tests de base.

Finalisation du projet avec l'ajout de nouvelles fonctionnalités, tests unitaires, documentation du code, et rédaction du rapport final.

Ce projet a permis de travailler sur des concepts clés de la programmation orientée objet, tels que la gestion des classes, l'encapsulation, et l'héritage, tout en se concentrant sur la création d'une interface textuelle simple mais interactive pour le joueur.

Présentation générale du jeu :

Description brève du jeu :

Le jeu est un jeu d'aventure textuel inspiré de *Colossal Cave Adventure*. Le joueur incarne un héros qui explore un univers composé de différentes *Locations* (lieux), interagit avec des *Items* (objets) et des *Characters* (personnages), et traverse des *Exits* (sorties) pour avancer dans l'aventure.

Thème :

Le jeu se déroule dans une galerie d'art rempli de mystères, d'objet, et de monstres scellé en tant qu'humain jusqu'à l'arrivé d'un élément déclencheur. Le joueur doit naviguer à travers des pièces pour vaincre les ennemis et survivre.

Gameplay :

Le jeu fonctionne principalement par des échanges textuels. Le joueur interagit avec le programme en entrant des commandes via une interface en ligne de commande. Le programme réagit aux actions du joueur en affichant des descriptions textuelles des situations, des lieux, des objets ou des personnages présents dans le jeu.

Règles :

1. Le joueur peut entrer des commandes textuelles pour interagir avec le jeu (par exemple, *GO NORTH*, *LOOK*, *TAKE KEY*).
2. Le joueur peut se déplacer entre différentes *Locations* en utilisant des commandes directionnelles (par exemple, *GO NORTH*, *GO SOUTH*).
3. Le joueur peut collecter et utiliser des objets avec des utilités et pouvoirs spécifiques pour progresser dans le jeu.

4. Le joueur peut interagir avec des personnages non-joueurs (PNJ) pour obtenir des informations et faire avancer l'intrigue.
5. Le jeu suit l'évolution de l'état du joueur, y compris sa position, ses objets, et sa santé.

Objectifs du joueur :

1. Explorer l'univers du jeu en visitant différentes *Locations*.
2. Collecter des objets et les utiliser pour survivre.
3. Interagir avec des personnages pour obtenir des indices et avancer dans l'intrigue.
4. Atteindre la fin de l'aventure en surmontant les obstacles.
5. Atteindre un état de "You Win" en accomplissant l'objectif principale du jeu.

---

2. Documentation utilisateur

Cette section décrit le jeu du point de vue de l'utilisateur, en expliquant comment le jouer et ce qu'il faut savoir pour bien profiter de l'expérience.

2.1. Présentation du jeu

Objectif du jeu :

C'est un jeu de découverte basé sur la curiosité du joueur. L'intrigue principale n'est pas du tout guidée, le joueur peut se déplacer dans toutes les pièces avec différents civils et différents objets. L'objectif commence quand le joueur casse le vase de la salle grecque avec la commande USE VASE FLOOR. Suite à cela le sceau est levé et les civils se transforment en démons qu'il faut purger pour gagner.
Il est évidemment conseillé de tester tous les objets puisqu'il ont chacun des effets uniques.

2.2. Prérequis et installation

Installation

- **Prérequis**

- Java Development Kit (JDK) 17 ou supérieur
- Apache Maven ou Gradle
- Un IDE comme IntelliJ IDEA ou Visual Studio Code

- **Étapes d'installation**

1. Téléchargez et extrayez le fichier ZIP du projet.

2. Ouvrez un terminal et naviguez vers le répertoire du projet extrait :

-cd <NOM_DU_REPERTOIRE>

Compilation :

- Get-ChildItem -Path src/main/java -Recurse -Filter *.java | ForEach-Object { $_.FullName } > sources.txt

-javac -d bin "@sources.txt" // Faites attention de bien enregistrer le fichier sources.txt en UTF-8

Lancement :

-java -cp bin com.projetcave.Main

- **Commandes du jeu**

HELP : Affiche les commandes disponibles.
LOOK : Affiche la description de la localisation actuelle.
GO <direction> : Déplace le héros dans la direction spécifiée.
TAKE <item> : Prend l'objet spécifié et l'ajoute à l'inventaire.
USE <item> <target> : Utilise l'objet spécifié sur la cible.
INTERACT <person> : Interagit avec la personne spécifiée.
ATTACK <target> : Attaque la cible spécifiée.
INVENTORY : Affiche l'inventaire du héros.
SAVE : Sauvegarde la partie.
LOAD : Charge la partie sauvegardée.
QUIT : Quitte le jeu.

---

3. Documentation développeur

Dans cette section, tu expliqueras la conception du projet en mettant l'accent sur l'architecture du code, les choix techniques, et la structure du projet.

3.1. Architecture du projet

- .vscode/ : Contient les configurations spécifiques à Visual Studio Code.
    - launch.json : Configuration pour le lancement et le débogage du projet.
    - settings.json : Paramètres spécifiques à l'éditeur.
- bin/ : Contient les fichiers compilés (.class) du projet.
- src/main/java/ : Contient le code source principal du projet.
    - com/projetcave/ : Package principal du projet.
        - commands/ : Contient les classes liées aux commandes du jeu.
            - Commands.java : Classe pour gérer les commandes entrées par le joueur.
        - exits/ : Contient les classes liées aux sorties des locations.
            - Exit.java : Classe représentant les sorties des locations.
        - game_state/ : Contient les classes liées à l'état du jeu.
            - GameState.java : Classe pour suivre l'état du jeu.
        - hero/ : Contient les classes liées au héros.
            - Backpack.java : Classe pour gérer l'inventaire du héros.
            - Hero.java : Classe représentant le héros du jeu.
        - items/ : Contient les classes liées aux objets.
            - Item.java : Classe représentant les objets que le joueur peut collecter.
        - locations/ : Contient les classes liées aux locations.
            - Location.java : Classe représentant les différentes locations du jeu.
        - people/ : Contient les classes liées aux personnages.
            - Person.java : Classe représentant les personnages non-joueurs (PNJ).

        - Main.java : Classe principale qui initialise et lance le jeu.
- src/test/java/ : Contient les tests unitaires pour le projet.
    - com/projetcave/ : Package principal des tests.
        - commands/ : Contient les tests pour les commandes.
            - CommandsTest.java : Tests unitaires pour la classe Commands.
        - exits/ : Contient les tests pour les sorties.
            - ExitTest.java : Tests unitaires pour la classe Exit.
        - game_state/ : Contient les tests pour l'état du jeu.
            - GameStateTest.java : Tests unitaires pour la classe GameState.
        - hero/ : Contient les tests pour le héros.
            - BackpackTest.java : Tests unitaires pour la classe Backpack.
            - HeroTest.java : Tests unitaires pour la classe Hero.
        - items/ : Contient les tests pour les objets.
            - ItemTest.java : Tests unitaires pour la classe Item.
        - locations/ : Contient les tests pour les locations.
            - LocationTest.java : Tests unitaires pour la classe Location.
        - people/ : Contient les tests pour les personnages.
            - PersonTest.java : Tests unitaires pour la classe Person.
- build.gradle : Fichier de configuration pour Gradle.
- pom.xml : Fichier de configuration pour Maven.
- README.txt : Fichier contenant des informations sur le projet.
- sources.txt : Fichier listant les chemins des fichiers source Java.
- target/ : Contient les fichiers compilés et les résultats des tests.

3.2. Diagramme de classes

Main: // Classe principale qui initialise et lance le jeu.

+ main(args: String[]): void

GameState: // Classe qui suit l'état du jeu, y compris les locations, le héros, et la position actuelle.

- locations: List<Location>
- hero: Hero
- currentLocation: Location
- locationCount: int
+ getLocations(): List<Location>
+ getHero(): Hero
+ setHero(hero: Hero): void
+ getCurrentLocation(): Location
+ setCurrentLocation(currentLocation: Location): void
+ getLocationCount(): int
+ setLocationCount(locationCount: int): void
+ saveGame(): void
+ loadGame(): void

Location: // Classe représentant les différentes locations du jeu, avec des sorties, des objets, des héros et des personnages.

- name: String
- description: String
- exits: List<Exit>
- items: List<Item>
- heroes: List<Hero>
- people: List<Person>
+ getName(): String
+ getDescription(): String
+ getExits(): List<Exit>
+ getItems(): List<Item>
+ getHeroes(): List<Hero>

+ getPeople(): List<Person>

+ addItem(item: Item): void

+ addHero(hero: Hero): void

+ addExit(exit: Exit): void

+ addPerson(person: Person): void

+ displayLocation(hero: Hero): void

Exit: // Classe représentant les sorties des locations, avec une direction, une destination, et un état de verrouillage.

- direction: String

- destination: Location

- locked: boolean

- keyName: String

+ getDirection(): String

+ getDestination(): Location

+ isLocked(): boolean

+ unlock(): void

+ getKeyName(): String

Hero: // Classe représentant le héros du jeu, avec un inventaire, des points de vie, et des capacités d'attaque.

- name: String

- backpack: Backpack

- currentLocationIndex: int

- lifePoints: int

- vaseBroken: boolean

- distracted: boolean

- hasSculpture: boolean

- attackDamage: int

- distractionCounter: int

+ getName(): String

+ getBackpack(): Backpack

+ getCurrentLocationIndex(): int

+ setCurrentLocationIndex(currentLocationIndex: int): void

+ getLifePoints(): int

+ setLifePoints(lifePoints: int): void

+ isVaseBroken(): boolean

+ setVaseBroken(vaseBroken: boolean): void

+ isDistracted(): boolean

+ setDistracted(distracted: boolean): void

+ hasSculpture(): boolean

+ setHasSculpture(hasSculpture: boolean): void

+ displayInventory(): void

+ useItemOnItem(item1: Item, item2: Item): boolean

+ takeDamage(damage: int): void

+ attack(): int

+ attack(person: Person): void

+ isAlive(): boolean

Backpack: // Classe pour gérer l'inventaire du héros, avec des objets, un volume et un poids.

- volume: double

- weight: double

- items: List<Item>

+ getVolume(): double

+ getWeight(): double

+ getItems(): List<Item>

+ addItem(item: Item): void

+ removeItem(item: Item): void

+ displayItems(): void

Item: // Classe représentant les objets que le joueur peut collecter, avec un nom, une description, et des propriétés.

- name: String

- description: String

- canBeTaken: boolean

- volume: double

- weight: double

+ getName(): String

+ getDescription(): String

+ canBeTaken(): boolean

+ getVolume(): double

+ getWeight(): double

+ use(hero: Hero, currentLocation: Location, target: String): void

Command: // Classe pour gérer les commandes entrées par le joueur, en modifiant l'état du jeu.

+ executeCommand(command: String, state: GameState): void

Person: // Classe représentant les personnages non-joueurs (PNJ), avec des interactions, des points de vie, et des capacités d'attaque.

- name: String

- fact: String

- health: int

- attackDamage: int

+ getName(): String

+ getFact(): String

+ interact(): void

+ getHealth(): int

+ takeDamage(damage: int): void

+ attack(hero: Hero): void

+ isAlive(): boolean

+ transform(): void

+ getAttackDamage(): int

---

4. Difficultés rencontrées et solutions apportées

1. Conception de l'architecture du jeu

Difficulté : La conception initiale de l'architecture du jeu, y compris la définition des classes principales et leurs interactions, a été un défi majeur. Il était crucial de s'assurer que le code soit modulaire et extensible pour permettre l'ajout de nouvelles fonctionnalités à l'avenir.

Solution : Pour surmonter cette difficulté, une approche méthodique a été adoptée. Un diagramme de classes détaillé a été créé pour visualiser les relations entre les différentes classes. Cela a permis de structurer efficacement le code et de définir clairement les responsabilités de chaque classe. Les concepts de la programmation orientée objet, tels que l'encapsulation et l'héritage, ont été utilisés pour organiser le code de manière modulaire.

2. Implémentation du système de commandes

Difficulté : L'implémentation d'un système de commandes textuelles interprétées par le jeu a été complexe. Il fallait s'assurer que les commandes soient correctement interprétées et exécutées, tout en gérant les différentes actions possibles du joueur.

Solution : Une classe Command a été créée pour gérer les commandes entrées par le joueur. Cette classe utilise des méthodes pour interpréter et exécuter les commandes, en modifiant l'état du jeu en conséquence. Des tests unitaires ont été écrits pour vérifier que les commandes fonctionnent comme prévu, ce qui a permis de détecter et de corriger les erreurs rapidement.

3. Gestion de l'état du jeu

Difficulté : Suivre l'évolution de l'état du jeu, y compris la position du joueur, les objets collectés, et la santé du héros, a été un défi. Il était important de s'assurer que l'état du jeu soit cohérent et puisse être sauvegardé et chargé correctement.

Solution : La classe GameState a été créée pour gérer l'état du jeu. Cette classe contient des méthodes pour sauvegarder et charger l'état du jeu, ainsi que pour suivre la position actuelle du joueur et les objets collectés. L'utilisation de la sérialisation a permis de sauvegarder et de charger facilement l'état du jeu.

4. Interaction avec les objets et les personnages

Difficulté : Permettre au joueur d'interagir avec les objets et les personnages dans le jeu a nécessité une gestion complexe des interactions possibles. Il fallait s'assurer que les objets puissent être collectés, utilisés, et que les personnages puissent interagir avec le joueur de manière cohérente.

Solution : Des classes spécifiques ont été créées pour les objets (Item) et les personnages (Person). Ces classes contiennent des méthodes pour gérer les interactions possibles, telles que la collecte d'objets, l'utilisation d'objets, et les dialogues avec les personnages. Des tests unitaires ont été écrits pour vérifier que ces interactions fonctionnent correctement.

5. Tests et débogage

Difficulté : Étant seul dans le groupe, il était difficile de tester et de déboguer toutes les fonctionnalités du jeu de manière exhaustive. Il fallait s'assurer que le jeu fonctionne correctement dans toutes les situations possibles.

Solution : L'utilisation de JUnit pour les tests unitaires a été cruciale. Des tests unitaires ont été écrits pour chaque classe et chaque fonctionnalité du jeu, ce qui a permis de détecter et de corriger les erreurs rapidement. De plus, des sessions de débogage régulières ont été effectuées pour s'assurer que le jeu fonctionne comme prévu.

6. Gestion du temps

Difficulté : La gestion du temps a été un défi majeur, surtout en étant seul dans le groupe. Il fallait s'assurer que toutes les fonctionnalités soient implémentées et testées dans le délai imparti de 4 semaines.

Solution : Un planning détaillé a été établi dès le début du projet, avec des objectifs hebdomadaires clairs. Chaque semaine était dédiée à une phase spécifique du projet (conception, développement, tests, finalisation). Cette

planification rigoureuse a permis de gérer le temps efficacement et de s'assurer que toutes les tâches soient accomplies dans les délais.

Conclusion

Malgré les défis rencontrés, le projet a été mené à bien grâce à une planification rigoureuse, une conception méthodique, et l'utilisation d'outils et de techniques appropriés. Les difficultés ont été surmontées en adoptant une approche structurée et en utilisant des tests unitaires pour garantir la qualité du code. Travailler seul a nécessité une gestion du temps et des ressources efficace, mais a également permis de développer des compétences précieuses en gestion de projet et en résolution de problèmes.

---

5. Bilan du projet

Résumé des objectifs atteints

Le projet avait pour objectif de créer un jeu d'aventure textuel inspiré de *Colossal Cave Adventure*. Les principaux objectifs étaient :

1. **Conception d'un univers interactif** : Un environnement riche a été développé, permettant au joueur de se déplacer entre différentes *Locations*, d'interagir avec des objets et des personnages, et de résoudre des énigmes pour progresser dans le jeu.
2. **Implémentation d'un système de commandes** : Un système de commandes textuelles a été implémenté, permettant au joueur d'interagir avec le jeu via des commandes comme *GO*, *LOOK*, *TAKE*, *USE*, *ATTACK*, etc.
3. **Création d'un système d'inventaire et d'interaction avec les objets** : Le héros peut collecter des objets, les utiliser avec d'autres éléments du jeu, et interagir avec des personnages ou des objets.
4. **Gestion de l'état du jeu** : Le jeu suit l'évolution de l'état du joueur, y compris sa position, ses objets, et sa santé. Un système de "fin de jeu" est en place (par exemple, "Game Over" ou "You Win").
5. **Extensibilité du code** : Le code a été conçu de manière modulaire pour permettre d'ajouter facilement de nouvelles fonctionnalités, commandes et éléments de jeu.

Tous ces objectifs ont été atteints, et le jeu développé offre une expérience interactive et engageante.

Qualité du jeu développé

Le jeu développé est de bonne qualité, avec une architecture bien structurée et un code modulaire. Les fonctionnalités principales sont implémentées et fonctionnent correctement. Les tests unitaires ont permis de s'assurer que les fonctionnalités du jeu sont correctement implémentées et que le programme fonctionne comme prévu. Le jeu offre une interface textuelle simple mais interactive, permettant au joueur de prendre des décisions qui influencent le déroulement de l'histoire.

Résultats obtenus

Le jeu permet au joueur de :

- Explorer un univers composé de différentes *Locations*.
- Interagir avec des objets et des personnages.
- Utiliser des commandes textuelles pour progresser dans l'aventure.
- Suivre l'évolution de l'état du jeu, y compris la position, les objets, et la santé du héros.
- Atteindre une fin de jeu en accomplissant les objectifs principaux.

Perspectives d'amélioration

Idées pour améliorer le jeu dans une future version

1. **Ajout de nouvelles fonctionnalités :**
    - Introduction de quêtes secondaires pour enrichir l'expérience de jeu.
    - Ajout de nouveaux types d'énigmes et de défis pour diversifier le gameplay.
    - Intégration de mécanismes de combat plus complexes avec des stratégies et des compétences spéciales.
2. **Amélioration de l'interface utilisateur :**
    - Développement d'une interface graphique (GUI) pour rendre le jeu plus accessible et visuellement attrayant.
    - Ajout de sons et de musiques d'ambiance pour améliorer l'immersion du joueur.
3. **Optimisation du code :**

- Refactorisation du code pour améliorer la lisibilité et la maintenabilité.
- Optimisation des performances pour gérer des univers de jeu plus vastes et plus complexes.

4. **Extensibilité :**
    - Création d'un éditeur de niveaux pour permettre aux joueurs de créer et de partager leurs propres aventures.
    - Mise en place d'un système de plugins pour ajouter facilement de nouvelles fonctionnalités et extensions au jeu.

Leçons apprises

Retours d'expérience personnels

1. **Gestion de projet :**
    - Travailler seul sur un projet de cette envergure a nécessité une planification rigoureuse et une gestion efficace du temps. La création d'un planning détaillé avec des objectifs hebdomadaires clairs a été essentielle pour rester sur la bonne voie et respecter les délais.
2. **Programmation :**
    - La conception et le développement d'un jeu d'aventure textuel ont permis de renforcer les compétences en programmation orientée objet, en particulier en ce qui concerne la gestion des classes, l'encapsulation, et l'héritage.
    - L'importance des tests unitaires a été mise en évidence, car ils ont permis de détecter et de corriger les erreurs rapidement, assurant ainsi la qualité du code.
3. **Travailler seul :**
    - Travailler seul a présenté des défis uniques, notamment en termes de motivation et de gestion de la charge de travail. Cependant, cela a également permis de développer des compétences en autonomie, en résolution de problèmes, et en prise de décision.
    - La nécessité de documenter le code et de maintenir une organisation claire a été cruciale pour gérer efficacement le projet et faciliter les futures améliorations.

En conclusion, ce projet a été une expérience enrichissante qui a permis de développer des compétences techniques et de gestion de projet, tout en créant un jeu d'aventure

textuel interactif et engageant. Les perspectives d'amélioration offrent de nombreuses opportunités pour continuer à développer et à enrichir le jeu à l'avenir.

---

6. Annexes

Visualisation de la map « ART GALLERY » :