

Trade Infinity (A3P 2021/2022) J4

Léo ROULLOIS

5 décembre 2021

Table des matières

1	Description du projet	2
1.1	Auteur	2
1.2	Thème (phrase-thème validée)	2
1.3	Résumé du scénario (complet).	2
1.4	Plan (complet, avec indication de la partie "réduit" si exercice 7.3.3)	2
1.5	Scénario détaillé (complet, avec indication de la partie "réduit" si exercice 7.3.3)	2
1.6	Détail des lieux, items, personnages	3
1.7	Situations gagnantes et perdantes	4
1.8	Eventuellement énigmes, mini-jeux, combats, etc.	4
1.9	Commentaires (ce qui manque, reste à faire, ...)	4
2	Réponses aux exercices (à partir de l'exercice 7.5 inclus).	5
3	Mode d'emploi (si nécessaire, instructions d'installation ou pour démarrer le jeu)	16
4	Déclaration obligatoire anti-plagiat.	17

Chapitre 1

Description du projet

1.1 Auteur

Léo Roullois

1.2 Thème (phrase-thème validée)

Au World Trade Center, Laylow un jeune trader doit multiplier son capitale par 1000 grâce aux cryptomonnaies pour gagner le jeu.

1.3 Résumé du scénario (complet).

Un trader possède un capital de départ de 10k\$ et devra atteindre la somme de 100k\$ pour gagner la partie. S'il descend sous les 100\$ (ou 1000\$, à décider en fonction de la difficulté du jeu et des énigmes). Pour cela, il traversera différentes salles, sera confronté à des énigmes, devra faire des investissements, participer à sécuriser le réseau d'une blockchain, créer des NFTs et convaincre le marché de les acheter... En fonction de ses actions, il sera rémunéré plus ou moins. Il y a également un facteur chance qui apparaît : une cryptomonnaie peut chuter ou bien exploser à la hausse.

1.4 Plan (complet, avec indication de la partie "réduit" si exercice 7.3.3)

(Plan encore un peu brouillon, version propre et épurée à venir. J'espère que vous allez quand même réussir à me comprendre.)

1.5 Scénario détaillé (complet, avec indication de la partie "réduit" si exercice 7.3.3)

Un trader a un capital de départ : 10k\$.

Pour gagner le jeu il doit multiplier son capital par 100 à l'aide des cryptomonnaies.

Pour cela, Laylow sera confronté à différentes salles :

- BTC (Bitcoin)
- ETH (Ethereum)

- DEFI (BSC, blockchains ethereum...)
- NFTs (Non Fungible Token)
- ShitCoins (Dogecoin, Shiba INU, SafeMoon, ...)
- Trading (analyse fondamentale, technique, ...)
- Minage (PoW, PoS, ...)
- Hack (Scam, manipulation du marché, attaque à 51%) : une fois la salle débloquée, potentiel gain très grand mais 10% de risque de perdre ses prochains investissements, sauf si la transaction a été effectuée depuis un wallet ledger. (Le joueur est prévenu une fois la salle débloquée).
- ICOs

Dans chaque salle il y aura des énigmes et des quêtes concernant les thèmes en question. Pour un total de 9 salles, le joueur devra en visiter au moins 7 avant de finir le jeu.

Le trader commence dans la salle **BTC** avec 10k\$. Une énigme lui sera posée (à définir). S'il réponds correctement, il peut choisir de placer un certain pourcentage de son capital dans le BTC : de 0% à 30% (investissement qui sera nécessairement rentable par la suite, mais le joueur ne le sais pas encore).

Dans la salle **ETH**, il y aura également des énigmes (à propos des smart contracts, initiation à la finance décentralisée...), qui permettront en fonction des réponses de débloquent un ou plusieurs investissements en ETH. Dans la suite du jeu, le joueur devra repassé dans la salle ETH pour trouver la clé qui ouvrira la salle DEFI (en complétant tous les challenge lié à la finance décentralisée).

Dans la salle **DEFI**, il y aura 2 niveaux :

- Etage BSC (Binance Smart Chain) : Le joueur peut se créer des revenus passifs en participant à des pools de liquidité ou en farmant des tokens sur la Binance Smart Chain (gains moyens, frais peu élevés, investissement plutôt sûres).
- Etage ETH (Ethereum) : De même, sauf que la blockchain Ethereum propose des gains plus élevés mais également des frais plus élevés (donc rentable sur le long terme...).

Dans la salle **Minage**, le joueur aura la possibilité de miner des cryptomonnaies et donc d'être rémunérer s'il valide des blocs (probabilité + ou - élevée en fonction de la puissance de calcul qu'il possède).

Dans la salle **NFT** le joueur peut créer des NFT et tenter de les vendre sur les marchés. Plus le NFT sera considéré comme jolie, plus le joueur pourra le vendre cher.

Dans la salle **ICO**, le joueur peut participer à des ICO, la majorité d'entre elles vont perdre de la valeur lors du lancement, mais au contraire, si elle prends de la valeur, le gain est potentiellement énorme.

Dans la salle **Trading**, le joueur va pouvoir suivre des analyses fondamentales et répondre à des quiz qui vont lui permettre d'apprendre les bases en trading et ensuite d'investir dans certains coins qu'il trouve intéressant.

Dans la salle **Hack** le joueur aura la possibilité d'organiser des attaques sur des blockchains : si l'attaque est validée, le joueur gagne instantanément la partie, même s'il n'a pas visiter le nombre de salles requis.

Dans la salle **Shitcoin** le joueur peut investir dans des shitcoins qui ont peut de chances de prendre de la valeur, mais si tel est le cas, alors le shitcoin en question prendrais minimum + 1000% de valeur en l'espace de quelques jours.

Enfin, le joueur pourra adopter des chiens, acheter des ordinateurs ou cartes graphiques (pour augmenter sa puissance de calcul) afin d'effectuer des attaques dans la salle **Hack** ou bien pour miner des cryptomonnaies dans la salle **Minage**. Le joueur pourra parler à des célébrités tel qu'Elon Musk afin qu'il l'aide ses investissements sur le shitcoin DogeCoin. Il pourra acheter un wallet Ledger afin de sécuriser toute ses transactions, dans le cas contraire il y a un risque que ses transactions échouent. Il pourra également manger un gateau magique qui va accélérer le temps et faire prendre de la valeur à certaines cryptomonnaies (définie aléatoirement).

1.6 Détail des lieux, items, personnages

A venir

1.7 Situations gagnantes et perdantes

A venir

1.8 Eventuellement énigmes, mini-jeux, combats, etc.

A venir

1.9 Commentaires (ce qui manque, reste à faire, ...)

Beaucoup de chose;)

Chapitre 2

Réponses aux exercices (à partir de l'exercice 7.5 inclus).

Exercice 7.5 (printLocationInfo).

```
public void printLocationInfo() {
    System.out.println("You are " + this.aCurrentRoom.getDescription());
    String vOutput = "";
    if (!(this.aCurrentRoom.aNorthExit == null)) {
        vOutput = vOutput + "north ";
    }
    if (!(this.aCurrentRoom.aEastExit == null)) {
        vOutput = vOutput + "east ";
    }
    if (!(this.aCurrentRoom.aSouthExit == null)) {
        vOutput = vOutput + "south ";
    }
    if (!(this.aCurrentRoom.aWestExit == null)) {
        vOutput = vOutput + "west";
    }
    System.out.println("Exits: " + vOutput);
}
```

Exercice 7.6 (getExit).

1) Car un attribut non initialisé par le constructeur prends la valeur null.

2) et 3) J'ai créer cette fonction dans la classe Room :

```
public Room getExit(final String direction) {
    if (direction.equals("north")) {
        return this.aNorthExit;
    }
    if (direction.equals("east")) {
        return this.aEastExit;
    }
    if (direction.equals("south")) {
```

```

        return this.aSouthExit;
    }
    if(direction.equals("west")) {
        return this.aWestExit;
    }
    return this;
}

```

Puis j'ai modifié ceci dans la procédure goRoom de Game :

```

String vDirection = pCommand.getSecondWord();
Room vNextRoom=aCurrentRoom.getExit(vDirection);
if (this.aCurrentRoom==vNextRoom) {
    System.out.println("Unknown direction !");
}else if (vNextRoom == null) {
    System.out.println("There is no door !");
    return;
} else {
    this.aCurrentRoom = vNextRoom;
    this.printLocationInfo();
}

```

Exercice 7.7 (getExitString).

J'ai modifié la procédure suivante dans Game :

```

public void printLocationInfo() {
    System.out.println("You are " + this.aCurrentRoom.getDescription());
    System.out.println(this.aCurrentRoom.getExitString());
}

```

Et créer cette fonction dans Room :

```

public String getExitString() {
    String vOutput = "";
    if (this.aNorthExit != null) {
        vOutput = vOutput + "north ";
    }
    if (this.aEastExit != null) {
        vOutput = vOutput + "east ";
    }
    if (this.aSouthExit != null) {
        vOutput = vOutput + "south ";
    }
    if (this.aWestExit != null) {
        vOutput = vOutput + "west ";
    }
    return ("Exits: " + vOutput);
}

```

Exercice 7.8 (HashMap, setExit)

Voici le nouvel attribut qui remplace les 4 attributs de salle précédents :

```
private HashMap<String,Room> aExits;
```

Et les modifications dans la classe room :

```
public Room(final String pDescription) {
    this.aDescription = pDescription;
    this.aExits = new HashMap<String,Room>();
}
```

et j'ai supprimé la procédure setExits pour la remplacée par :

```
public void setExit(String pDirection, Room pNeighbor) {
    if (pDirection != null) {
        this.aExits.put(pDirection, pNeighbor);
    }
}
```

puis la fonction get Exit :

```
public Room getExit(final String direction) {
    return this.aExits.get(direction);
}
```

et enfin getExitString :

```
public String getExitString() {
    String vOutput = "Exits : ";
    Set<String> allKeys = this.aExits.keySet();
    for (String vKey : allKeys) {
        vOutput+=vKey+" ";
    }
    return vOutput;
}
```

Des modifications ont également été faites dans Game pour coller aux nouvelles fonctions.

Exercice 7.9 (keySet).

Modifications déjà effectuées lors de l'exercice précédent.

Exercice 7.10 (getExitString CCM?).

Afin de lister toutes les sorties possibles d'une salle à partir de la HashMap de cette même salle, il faut commencé par créer une collection de String qui contiendra toutes les clés de cette HashMap. Une fois cette collection créée, nous allons itérer sur cette collection grâce à la syntaxe for vue précédemment. A chaque itération, nous ajouterons la valeur de la clé à une variable, qu'on retournera à la sortie de cette boucle for.

Exercice 7.11 (getLongDescription)

Dans la classe Game :

```
public void printLocationInfo() {
    System.out.println(this.aCurrentRoom.getLongDescription());
}
```


Et dans la classe Room :

```
public String getLongDescription() {
    return "You are "+this.aDescription+".\n"+this.getExitString();
}
```

Exercice 7.14

Dans la classe Game :

```
private void look(final Command pCommand) {
    if (pCommand.hasSecondWord()) {
        System.out.println("I don't know how to look at something in particular yet.");
    } else {
        System.out.println(this.aCurrentRoom.getLongDescription());
    }
}

private boolean processCommand(final Command pCommand) {
    if (pCommand.isUnknown()) {
        System.out.println("I don't know what you mean...");
        return false;
    } else {
        if (pCommand.getCommandWord().equals("go")) {
            this.goRoom(pCommand);
        } else if (pCommand.getCommandWord().equals("help")) {
            this.printHelp();
        } else if (pCommand.getCommandWord().equals("look")) {
            this.look(pCommand);
        }

        if (pCommand.getCommandWord().equals("quit")) {
            return quit(pCommand);
        } else {
            return false;
        }
    }
}
}
```

Et dans CommandWords :

```
private static final String[] VALID_COMMANDS = {
    "go", "quit", "help", "look"
};
```

Ainsi que des modifications mineures pour accéder au tableau VALID_COMMANDS.

Exercice 7.15.

J'ai ajouté la commande « buy » qui permet d'acheter des cryptomonnaies (bitcoin, ethereum...).

Exercice 7.16

Dans la classe CommandWords :

```

/**
 * Affiche toute les commandes du jeu
 */
public void showAll() {
    for (String pCommand : VALID_COMMANDS) {
        System.out.print(pCommand+" ");
    }
    System.out.println();
}

```

Dans la classe Parser :

```

public void showCommands() {
    aValidCommands.showAll();
}

```

Et dans la classe Game :

```

/**
 * Affiche un message d'aide
 */
private void printHelp() {
    System.out.println("You are lost. You are alone.");
    System.out.println("Your command words are:");
    aParser.showCommands();
}

```

Exercice 7.18

J'ai modifier la procédure showAll dans CommandWords que j'ai renommé getCommandList et transformé de sorte qu'elle renvoie un string contenant toute les commandes. Ensuite j'ai modifier la fonction showCommands dans Parser de façon à ce qu'elle s'adapte aux modifications précédentes.

Exercice 7.18.2

Amélioration de la performance de la concaténation de String dans getExitString (Room) grâce à StringBuilder.

Exercice 7.18.5

Création d'une HashMap qui répertorie toutes les salles du jeu. Ajout de l'affichage de toutes ses salles dans printWelcome.

Exercice 7.18.6

Toutes les modifications pour importer le projet zuul-with-image dans mon jeu Trade Infinity ont été effectuées :

- Déplacement de la classe Game dans GameEngine
- Création de la nouvelle classe Game
- Dans GameEngine, ajout des images en paramètre lors de la création de chaque Room
- Création de la classe UserInterface

- Autres modifications comme processCommand qui devient interpretCommand, son paramètre devient un String, tous les System.out.println deviennent des this.aGui.println
- Modification des imports dans UserInterface

Exercice 7.18.7

Le code composant.addActionListener(écouteur) ajoute un écouteur d'événement (ici l'écoute du clavier et de la touche Entrée) à l'objet composant. Une fois qu'un événement est écouté, l'objet écouteur va exécuter sa fonction actionPerformed() afin de faire un traitement.

Exercice 7.18.8

Dans la classe UserInterface, création d'une HashMap qui contiendra tous les boutons du jeu, et création d'un panel qui va me permettre de choisir la disposition de ceux-ci :

```
private HashMap<String, JButton> aButtons;
private JPanel aPanelButtons;
```

Le constructeur devient donc :

```
public UserInterface(final GameEngine pGameEngine) {
    this.aButtons = new HashMap<String, JButton>();
    this.aPanelButtons = new JPanel();
    aPanelButtons.setLayout(new GridLayout(2,1));
    this.aEngine = pGameEngine;
    this.createGUI();
} // UserInterface(.)
```

Ensuite, dans la procédure createGUI, instantiation des boutons via la classe JButton, puis ajout de ces nouveaux bouton dans la HashMap :

```
JButton bitcoinBtn = new JButton("Buy Bitcoin");
JButton quitButton = new JButton("Quit");
this.aButtons.put("bitcoin", bitcoinBtn);
this.aButtons.put("quit", quitButton);
```

Ensuite, parcours de la HashMap pour ajouter les boutons au nouveau panel et leur ajouter un écouteur d'événement, puis ajout du nouveau panel au panel principal :

```
Set<String> allKeys = this.aButtons.keySet();
for(String vKey : allKeys) {
    this.aPanelButtons.add(this.aButtons.get(vKey));
    this.aButtons.get(vKey).addActionListener(this);
}
vPanel.add(this.aPanelButtons, BorderLayout.EAST);
```

Pour finir, la méthode actionPerformed gère quelle action sera exécutée lors du clic sur chacun des boutons. Une procédure disableButtons a été créée pour désactiver les boutons (par exemple lorsque l'on quitte le jeu) :

```
public void disableButtons() {
    Set<String> allButtons = this.aButtons.keySet();
    for (String vButton : allButtons) {
        this.aButtons.get(vButton).setEnabled(false);
    }
}
```

Exercice 7.19.2

Toutes les images ont été déplacées dans un dossier nommé img.

Exercice 7.20

Une nouvelle classe Item a été créée, possédant 2 attributs (price et description). Les Getters et Setters ont été écrits, ainsi que le constructeur naturel. Ensuite, dans la classe Room j'ai créé un nouvel attribut aItem, initialisé à null dans le constructeur. Une nouvelle méthode setItem a été créée pour initialiser un item. Enfin, la fonction getItemString permet de récupérer la description de l'item.

Exercice 7.21

Toutes les informations à propos d'un item doivent être créées dans la classe Item. La classe qui produit la chaîne décrivant l'objet est la classe Item. La classe qui affiche la description de l'item est la classe GameEngine, car c'est ici que ces items sont créés.

Exercice 7.21.1

Modification de la commande look pour qu'elle puisse nous informer sur les items présents dans une salle.

Exercice 7.22

Modification de l'attribut String aItem en HashMap<String,Item> aItems, et modifications qui s'en suivent : addItem...

Exercice 7.22.1

Dans l'exercice 7.22 j'ai choisi d'utiliser une HashMap comme collection car cela permet d'accéder rapidement à un item via son nom.

Exercice 7.22.2

Intégration des items présents dans mon jeu.

Exercice 7.23

Création d'une nouvelle commande back. Pour cela, il aura fallu créer un nouvel attribut dans GameEngine qui mémorise la valeur de la salle précédente.

Exercice 7.26

Création d'une pile via la classe Stack dans GameEngine. Cette pile contient les salles dans l'ordre où l'on s'est déplacé, ce qui permet d'améliorer la fonction back afin de retracer tous notre chemin.

Exercice 7.26.1

Génération des javadocs utilisateur et programmeur.

Exercice 7.28.1

Création d'une nouvelle commande test qui exécute un fichier test. Pour cela création d'une procédure test dans GameEngine :

```
private void test(final Command pCommand) {
    if (!pCommand.hasSecondWord()) {
        this.aGui.println("What file do you want for the test ?");
    } else {
        Scanner myFile = null;
        try {
            myFile = new Scanner(new BufferedReader(new FileReader(pCommand.getSecondWord() + ".txt")));
            while (myFile.hasNext()) {
                String aLigne = myFile.nextLine();
                this.interpretCommand(aLigne);
            }
        } catch (final FileNotFoundException e) {
            this.aGui.println("Error : " + e.getMessage());
        } finally {
            if (myFile != null) {
                myFile.close();
            }
        }
    }
}
```

Exercice 7.28.2

Création de deux fichiers de tests : un qui test la situation gagnante et un autre qui test toutes les possibilités.

Exercice 7.28.3

A faire.

Exercice 7.29

Création de la classe Player qui contient toutes les informations liées au joueur : son nom, ses items, son argent, la salle actuelle et les salles visitées précédemment. De plus, cette classe contient tous les setters et getters nécessaires, ainsi que deux méthodes :

```
public void changeRoom(Room pNextRoom) {
    this.aPrevRooms.push(this.aCurrentRoom);
    this.aCurrentRoom = pNextRoom;
}
```

et :

```
public void goBack() {
    this.aCurrentRoom = this.aPrevRooms.pop();
}
```

Ajout d'un attribut aPlayer dans GameEngine, et beaucoup de modifications effectuées pour adapter l'ancien code.

Exercice 7.30

Création d'un attribut Item aItem et deux nouvelles fonctions dans la classe Player :

```
public String dropItem(final String pName)
```

et :

```
public String takeItem(final String pName)
```

qui respectivement permettent de déposer et de prendre un item. Elles renvoient un string avec confirmation de si l'opération s'est effectuée avec succès ou non. De plus ajout de deux mots-clés dans CommandWords : take et drop.

Exercice 7.31

Extension des fonctionnalités précédentes en remplaçant l'attribut Item aItem par une collection

```
private HashMap<String, Item> aItems;
```

. Les modifications nécessaires ont été effectuées dans les fonctions takeItem et dropItem de la classe player.

Exercice 7.31.1

Création d'une nouvelle classe ItemList suivante :

```
import java.util.HashMap;
import java.util.Set;

public class ItemList {
    private HashMap<String, Item> aItems;

    public ItemList() {
        this.aItems = new HashMap<String, Item>();
    }
    public Item getItem(final String pName) {
        return this.aItems.get(pName);
    }
    public double getPrice(final String pName) {
        return this.aItems.get(pName).getPrice();
    }
    public HashMap<String, Item> getItems() {
        return this.aItems;
    }

    public void addItem(final Item pItem) {
```

```

        this.aItems.put(pItem.getName(), pItem);
    }

    public void removeItem(final String pName) {
        this.aItems.remove(pName);
    }

    public String getItemsString() {
        String vAllItems = "";
        StringBuilder vSb = new StringBuilder(vAllItems);
        Set<String> allKeys = this.aItems.keySet();
        for (String vKey : allKeys) {
            vSb.append(" ");
            vSb.append(vKey);
        }
        return vSb.toString();
    }
}

```

Et modifications nécessaires dans les classes Player et Room.

Exercice 7.32

Ajout d'un système d'achat : pour cela dans la classe Player j'ai rajouter un attribut

```
private double aBalance;
```

qui représente l'argent du joueur. Chaque item possédant un attribut

```
private double aPrice;
```

j'ai pu effectuer les modifications nécessaires dans la fonction takeItem de Player.

Exercice 7.33

Implémentation d'une nouvelle commande items, pour cela ajout du mot clé items dans CommandWords et création d'une fonction lookItems dans Player :

```

public String lookItems() {
    String vItems = this.getItemsString();
    if (vItems.equals("")) {
        return "You don't have items yet";
    } else {
        return "Your items are :" + vItems;
    }
}

```

où

```

public String getItemsString() {
    return this.aItems.getItemsString();
}

```

et dans ItemList :

```

public String getItemsString() {
    String vAllItems = "";
    StringBuilder vSb = new StringBuilder(vAllItems);
    Set<String> allKeys = this.aItems.keySet();
    for (String vKey : allKeys) {
        vSb.append("\n- ");
        vSb.append(vKey);
        vSb.append(" (" +this.aItems.get(vKey).getPrice()+"$)");
    }
    return vSb.toString();
}

```

Exercice 7.34

Création d'une nouvelle commande : eat. Celle ci permet de manger le gâteau magique caché dans la salle Ethereum. Lorsque le gâteau est mangé, l'argent du joueur augmente de 20%. Voici la fonction dédiée dans la class Player :

```

public String eatCake(final Command pCommand) {
    if(!pCommand.hasSecondWord()) {
        return "Eat what ?";
    }
    Set<String> allKeys = this.aItems.getItems().keySet();
    for (String vKey : allKeys) {
        if(pCommand.getSecondWord().equals(vKey) && !vKey.equals("cake")) {
            return "You can't it a "+vKey+ ".";
        }
    }
    if (pCommand.getSecondWord().equals("cake")) {
        Item vCake = this.aItems.getItem("cake");
        if(vCake==null) {
            return "You don't have a cake in your inventory.";
        } else {
            this.aBalance*=1.20;
            return "You eated a magic cake ! It increases your wallet balance by 20%";
        }
    }
    return "Eat what ?";
}

```


Chapitre 3

Mode d'emploi (si nécessaire,
instructions d'installation ou pour
démarrer le jeu)

A venir

Chapitre 4

Déclaration obligatoire anti-plagiat.

Je n'ai recopié aucune ligne de code. Par contre, je me suis inspiré des idées et réponses proposées dans certains exercices du projet.