ESIEE PARIS

Projet Zuul : 404 error : PC not found

Rapport Projet Zuul encadré par Denis BUREAU

Année : 2024/2025

Par Pierre MATAR.

Table des matières :

1) Présentation du projet Zuul

L'objectif de ce projet est de développer un jeu d'aventure interactif en Java, basé sur le concept de Zuul. Le but étant de réaliser une introduction à la programmation objet en Java.

1.1) Auteur

Pierre MATAR, étudiant en B3 (3ème année du bachelor BestM) à ESIEE Paris.

1.2) Thème

Dans une entreprise de technologie, Joe doit trouver son ordinateur pour résoudre un bug bloquant sur un projet avant une présentation devant les dirigeants de l'entreprise.

1.3) Résumé du scénario

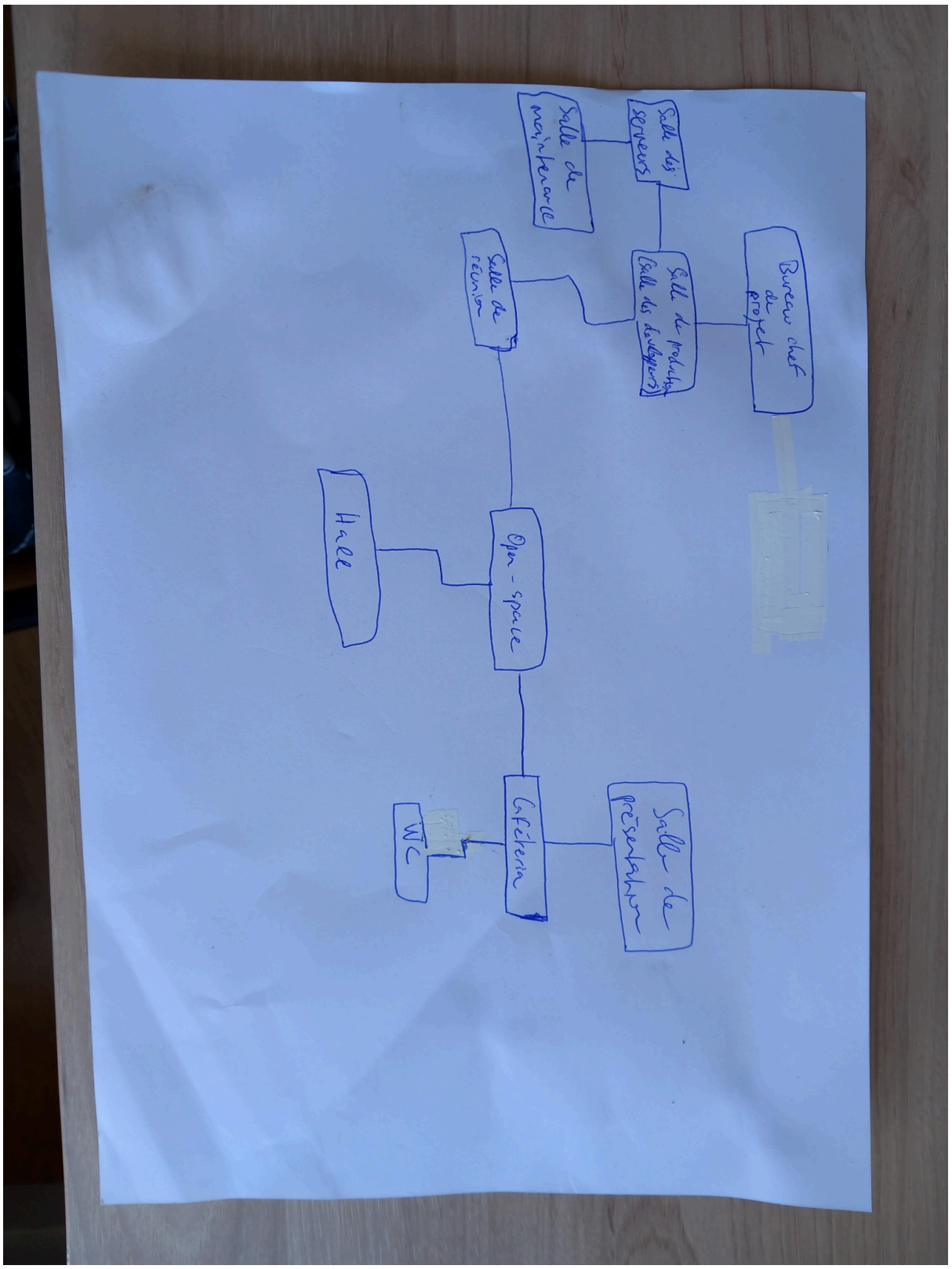
Joe, un jeune développeur de 25 ans, a toujours rêvé de travailler dans une grande entreprise de technologie. Fraîchement diplômé, il décroche son premier emploi en tant que développeur full-stack junior. Plein d'ambition, Joe voit ce travail comme l'opportunité idéale pour prouver sa valeur et gravir les échelons vers son rêve de devenir chef de projet.

Ses premiers jours se passent sans encombre, mais très vite, il est confronté à une mission d'une importance capitale : résoudre un bug bloquant dans un logiciel essentiel avant une présentation critique devant les hauts dirigeants de l'entreprise. C'est un projet sur lequel toute l'équipe travaille depuis des semaines, et Joe, désigné pour faire la dernière révision, doit prouver qu'il est à la hauteur.

Cependant, au moment de se préparer pour la tâche, Joe découvre avec stupeur que son ordinateur a disparu. Son PC, contenant les informations vitales pour corriger le bug, est introuvable. C'est ainsi que commence sa première véritable mission : retrouver son ordinateur et résoudre ce bug avant la réunion, ou risquer de compromettre l'avenir de l'entreprise... et sa carrière. Traversant les différents services, salles et énigmes de l'entreprise, Joe doit non seulement retrouver son ordinateur, mais aussi prouver ses compétences à ses collègues,

Le but du jeu étant donc de parvenir à réaliser sa première mission en entreprise, donc de pouvoir retrouver son PC et corriger le bug

1.4) Plan



Les relations en zig-zag représentent la possibilité de faire un Up-Down tandis que les relations directes représentent les directions Nord-Sud-Est-Ouest

1.6) Détail des lieux, items et personnages

Voici mes premières idées :

1. **Hall d'entrée** : Point de départ avec des instructions.
2. **Salle des développeurs** : Plusieurs PC et des post-it avec des indices utiles.
3. **Salle des serveurs** : Un serveur défectueux nécessitant une intervention technique.
4. **Salle de réunion** : Énigme de sécurité à résoudre pour accéder à l'étage supérieur.
5. **Cafétéria** : Un collègue qui détient un badge pour la salle de réunion.
6. **Bureau du chef de projet** : Contient le badge pour entrer dans la salle de présentation.
7. **Open-space** : Plusieurs personnages offrant des informations ou des distractions.
8. **Salle de présentation** : L'ordinateur à corriger pour terminer la mission.
9. **Salle de maintenance** : Cette salle contient des outils et des équipements nécessaires pour résoudre divers problèmes techniques.
10. **WC** : Rien pour l'instant

J'envisage par la suite d'ajouter des personnages et items.

1.7) Situations gagnantes et perdantes

- **Situation gagnante** : Joe parvient à corriger l'erreur dans le programme avant le début de la réunion avec les dirigeants.
- **Situation perdante** : Joe échoue à résoudre le problème à temps (potentiel implémentation d'un minuteur ou compteur), ce qui entraîne l'échec de sa mission et donc le game over.

1.8) Énigmes

Pas encore fait.

1.9) Commandes du jeu

- `go direction` : Se déplacer d'une salle à une autre.
- `help` : Affiche l'aide.
- `look` : Affiche le lieu où se trouve le joueur (Joe) ainsi que ses sorties possibles.
- `quit` : Quitte le jeu.
- `eat :` Affiche que le joueur a mangé.

D'autres commandes seront ajoutées par la suite.

1.10) Commentaires

Le jeu est encore en cours de développement.

2) Exercices

Exercice 7.5: Voici la méthode printLocationInfo(), implémenter dans la classe Game qui permet d'afficher la current room, donc la salle actuelle ou est le joueur ainsi que les exits possibles de cette salle. Elle nous a permis en particulier de mieux arranger notre code pour qu'il soit plus clair et plus facile à modifier par la suite en supprimant toute la duplication de code.

```
    private void printLocationInfo() {

        System.out.println( "You are " + this.aCurrentRoom.getDescription());
        System.out.print( "Exits: " );
        if(this.aCurrentRoom.aNorthExit != null) {
            System.out.print( "north ");
        }
        if(this.aCurrentRoom.aEastExit != null) {
            System.out.print( "east " );
        }

        if(this.aCurrentRoom.aSouthExit != null) {
            System.out . print( " south ");
        }
        if(this.aCurrentRoom.aWestExit != null) {
            System.out.print( " west " );
        }
        System.out.println();
    } //printLocationInfo
```

Exercice 7.6: Voici la méthode getExit() implantée dans la classe Room, prenant comme parametre la direction donnée et nous renvoyant la Room de sortie par rapport à ce paramètre. Cette méthode nous a également permis de supprimer la duplication de code dans la classe Game au sein de la méthode goRoom() en particulier.

```
public Room getExit(final String pDirection) {
    if (pDirection.equals("North"))
        return this.aNorthExit;
    if (pDirection.equals("South"))
        return this.aSouthExit;
    if (pDirection.equals("East"))
        return this.aEastExit;
    if (pDirection.equals("West"))
        return this.aWestExit;

    return null;


}
```

Ex 7.7: Nous devons créer une méthode getExitString() dans la classe Room permettant de connaitre les sorties possibles d'une salle. Elle nous permet de réduire la duplication de code notamment dans printWelcome et printLocationInfo.

```
public String getExitString() {
    String vExitString = "Exits:  ";

    if(this.getExit("North") != null) {
        vExitString += "north ";
    }
    if(this.getExit("South") != null) {
        vExitString += "south ";
    }

    if(this.getExit("East") != null) {
        vExitString += "east ";
    }
    if(this.getExit("West") != null) {
        vExitString += "west ";
    }
    return vExitString;

}
```

7.8: Méthode setExits rendu plus simple par l'importation d'une HashMap qui nous permet de stocker les exits un par un au lieu de 4 à la fois, elle nous facilite la création de nouvelles directions comme Up et Down.

7.8.1: j'ai aussi ajouté par la suite 3 nouvelles salles notamment un hall d'entrée pour faire Up pour accéder à l'étage de l'entreprise, on peut également redescendre dans cette pièce en faisant down.

```
    public void setExits(final String pDirection, final Room pNeighbor){
        aExits.put(pDirection, pNeighbor);
    }
```

7.9: Amélioration de la méthode getExitString(), en la modifiant par une boucle for (for each), qui nous permet d'itérer sur les élements (les cléfs) de la HashMap aExits afin de savoir si leur valeur est différent de null afin de concaténer la cléf a vExitString pour renvoyer la liste des directions possibles.

```
public String getExitString()
{
    String vExitString = "Exits:  ";

    for (String element : this.aExits.keySet()) {
        if (this.aExits.get(element) != null)
            vExitString += " "+ element;
    }

    return vExitString;
}
```

7.10:
La méthode getExitString permet de renvoyer une chaîne de caractère avec toutes les sorties possibles. On va tout d'abord déclarer une String vExitString avec le contenu "Exits: " puis on va itérer sur toutes les clés (avec une boucle for foreach) de la hashmap aExits grâce à la méthode keySet() qui permet de récuperer les clés de la hashmap de les mettre dans un objet type Set. On itére donc sur cet objet et si l'exit est différent de la référence null, si cette condition est true alors on concatène avec la String déclarer comme variable locale de la méthode, ensuite on retourne cette String une fois la boucle finie.

7.10.1/7.10.2 : JavaDoc compléter et générer.

7.11 : Voici la méthode getLongDescription qui retourne une String par rapport à la description de la room ainsi que les sorties de celle-ci. On a implémenté cette méthode pour que chaque classe gère ce qui lui appartient, on ne doit pas produire la String à partir de la classe Game, on doit appeler une méthode de la classe Room qui le fait car c'est relatif à une room.

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

7.14: On implémente une nouvelle commande au jeu, look : qui ne fait simplement que afficher la description de la current room ainsi que ses exits par l'appel de la méthode getLongDescription.

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

7.15:
La méthode eat est une 2ème nouvelle commande au jeu, elle affiche que le joueur a bien mangé.

```
-     public void eat() {
-         System.out.println("You have eaten now and you are not
  hungry any more");
-     }
```

7.16 et 7.18 : Voici la méthode showAll implémenter dans la classe CommandWords et showCommands implémenter dans la classe Parser qui permettent d'obtenir la liste des commandes disponibles :

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

```
public void showAll() {
    for (String command : this.aValidCommands)
        System.out.print(command + " ");
    System.out.println();
}
```

On remarque ensuite que ce sont en fait des accesseurs donc on va changer leur nom et remplacer par get :

```
public String getCommands() {
    return this.aValidCommands.getCommandList();
}
```

```
public String getCommandList() {
    String vCommandList = "";
    for (String command : this.aValidCommands)
        vCommandList += command + " ";
    return vCommandList;
} //getCommandList
```

On oublie pas ensuite de faire la correspondance dans la classe Game comme ceci :

```
System.out.println(this.aParser.getCommands());
```

Cette ligne ajouté dans la méthode help de la classe Game permet d'afficher les commandes disponible à partir de son attribut aParser qui est un objet de type Parser et de pouvoir appellé sa méthode getCommands qui elle appelle la méthode getCommandList de la classe CommandWords afin d'obtenir les commandes dispoibles à partir de son attribut aValidCommands.

7.18.1 : Rien à changer après avoir fait la comparaison.

7.18.2 : Voici getExitString avec l'implémentation de StringBuilder qui est plus performant notamment lors de concaténations de String dans des boucles.

```
public String getExitString() {
    StringBuilder vSb = new StringBuilder("Exits:  ");

    for (String element : this.aExits.keySet()) {
        if (this.aExits.get(element) != null)
            vSb.append(element+" ");
    }

    return vSb.toString();

}//getExitString
```

Je l'ai implémenté dans tous les endroits nécessaires.

7.18.3 : Pas d'image encore trouvée, à faire plus tard.

7.18.4 : Nom du jeu ajouté dans la méthode printWelcome dans la classe Game. Nom du jeu -> 404 error : PC not found.

7.18.5 : Exercice optionnel à faire à l'exercice 7.46.

4) Déclaration plagiat

Pas de plagiat jusqu'à présent.