The Stolen Phone : https://perso.esiee.fr/~palaysil/Projet/TheStolenPhone

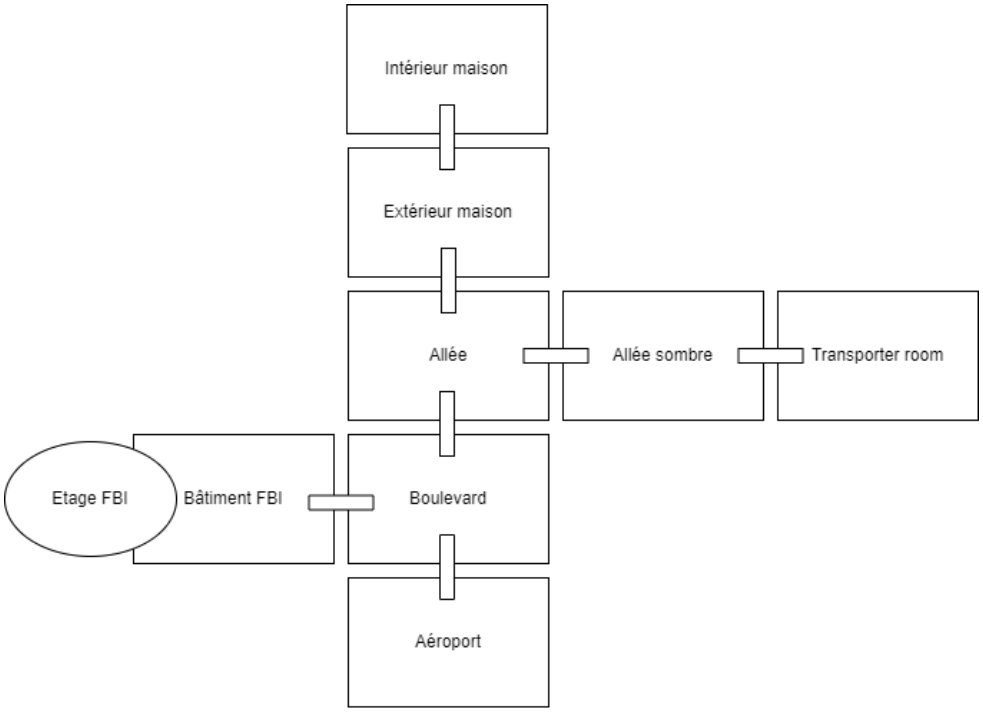
(IPO 2021/2022 G8)

I)
I.A) Auteur : ***Luca***

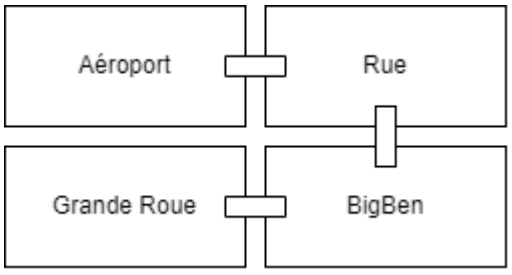
I.B) À travers le monde : Un musicien connu doit récupérer son téléphone volé.

I.C) Un musicien très connu, avant de prendre l'avion, se rend compte que son téléphone a disparu, il contient pleins de nouvelles musiques qui ne sont pas encore sorties. Il part donc chez lui pour voir s'il ne l'a pas oublié mais sur le chemin, plusieurs personnes lui parlent de rumeurs sur de nouvelles musiques pas encore sorties… Il arrive à tracer les différentes copies du téléphone et va en angleterre, pour aller chercher le téléphone à la grande roue.

I.D) plans :




<u>Carte des Etats-Unis :</u>




<u>Carte de l'Angleterre</u>

Les aéroports seront connectés entre eux. D'autres cartes sont à venir (plus petites) et pour l'instant, seulement la première carte est intégrée dans le jeu.

I.E) Le personnage se retrouve à l'aéroport, mais sans son téléphone. Il part donc vers sa maison pour voir s'il s'y trouve mais il ne trouve rien. Il va aller en direction du bâtiment du FBI pour récupérer un ticket vers l'Angleterre. Une fois rendue là-bas, il se dirige vers la grande roue et se bat avec Oscar, le grand méchant qui détient son téléphone, avec un couteau qu'il a trouvé à Big Ben. Le combat gagné, il repart vers sa maison, où quand il pose son téléphone, il se sent bien et repose en paix avec ses musiques exclusives.

I.F) Musicien, plusieurs pnj, méchant. Items : Sandwich, KeyFBI, Prot, KeyHouse, Ticket, Beamer, Knife. Deux cartes, une en Amérique et une en angleterre.

I.G) Pour gagner : Avoir récupéré le téléphone contenant beaucoup de musique et le drop à l'intérieur de la maison.
Pour perdre : ne plus avoir de déplacements disponible.

I.H) Pour récupérer le ticket pour aller en angleterre, il faut aller dans la tour du FBI, 2 solutions possible : prendre la clé et ouvrir la porte du bâtiment ou alors donner la clé à Alex qui vous donnera le "cookie magic" dans le jeu "prot" et qui ouvrira le bâtiment du FBI.
Ensuite prendre le ticket et aller à l'aéroport, puis 'open south' pour y avoir accès.
Pour gagner le combat, à la grande roue, il faut prendre le couteau, qui est à Big Ben.

I.I) Ajout d'autres "énigmes", "jeu", grandeur du terrain… Une interface plus sophistiquée et des changements des pnj en fonction du jeu.

II)
7.5) J'ai dû créer une méthode "printLocationInfo" qui envoie la description et les sorties possibles pour éviter d'écrire plusieurs fois le même code. Je l'ai appelé dans la méthode "goRoom" ainsi que "printWelcome" et probablement d'en d'autres méthodes ou fonctions.

7.6) Dans la classe Room : J'ai mis en privée les attributs "sorties" en plus de créer un attribut statique pour gérer un futur problème. Créer un accesseur pour avoir les sorties en fonction de la direction.
Dans la classe Game : Optimisation de la méthode goRoom grâce à l'accesseur des sorties de la classe Room. Utilisation de la solution 3 pour toujours avoir le message "Unknown direction !" (avec l'attribut statique de la classe Room. Changement de la méthode printLocationInfo pour ajuster avec les accesseurs de la classe Room.

7.7) Dans la classe Room : Création de la fonction getExitString pour retourner toutes les sorties possibles.
Dans la classe Game : Changement de la fonction printLocationInfo en fonction de la fonction getExitString.
La classe Room contient toutes les informations d'une pièce donc elle va pouvoir les donner directement au lieu de passer par les accesseurs. La classe Game va pouvoir utiliser cette fonction pour pouvoir seulement les afficher.

7.8) Ajout d'une hashmap pour remplacer les attributs de sorties dans la classe Room, modification de nombreuses méthodes pour qu'elles soient compatibles avec la hashmap.
Modification de createRooms dans Game pour fonctionner avec la nouvelle méthode setExits dans Room.

7.8.1) Ajout des bureaux du FBI et donc des sorties "up" et "down" dans la méthode getExitString.

7.9) Modification de getExitString selon le code 7.7 du livre.

7.10) 1ere ligne : déclare et crée la variable String returnString avec comme donnée "Exits:"
2eme ligne : récupère la liste de toutes les clés de l'attribut hashmap des sorties dans une collection.
3 à 5 lignes : la boucle va effectuer le code pour chaque élément de la collection donc ici pour chaque sortie, la direction de la sortie va être ajoutée à la variable returnString en mettant un espace entre chacune d'elle pour que l'affiche soit propre.
6eme ligne : retourne la variable returnString

7.10.2) Ajout de la javadoc sur le site web

7.11) Création de la fonction getLongDescription dans la classe Room qui retourne une String de la description + les sorties possibles qui permet d'optimiser la fonction printLocationInfo dans la classe Game.

7.14-7.15) Ajout des commandes looks et eat.
Look permet de donner le lieu et les sorties possibles en fonction de la position du joueur.
Eat permet de faire manger le joueur (pour l'instant ne retourne qu'il a assez mangé).

7.16) Création de la méthode showAll dans CommandWords pour permettre d'afficher toutes les commandes possibles, enregistrer dans une constante en attribut de la même classe. La méthode utilise une boucle for each qui est bien adaptée à cette situation. Pour permettre de relier la classe Game à CommandWords, nous utilisons le fait que la classe Parser est étendue de la classe CommandsWords. Dans Parser nous créons donc une méthode, showCommands qui permet d'appeler showAll. Enfin nous venons modifier dans la classe Game, printHelp, qui utilise désormais la méthode showCommands pour afficher les commandes valides.

7.18) Modification de showAll, dans CommandWords → transformé en getCommandList, qui ne retourne maintenant qu'un String avec toutes les commandes au lieu de les afficher.
Adaptation de la procédure showCommands dans Parser pour fonctionner avec le nouveau nom de showAll.
On affiche ce String dans la classe Game, avec la procédure printHelp.

7.18.2) Intégration du StringBuilder dans CommandWords.getCommandList() et Room.getExitString() pour optimiser la concaténation des String.

7.18.5) Ajout d'une HashMap dans Room en attribut. Modification de createRooms pour ajouter les rooms dans la HashMap.

7.18.6) Fusion du projet actuel avec zuul-with-images,  changement dans toutes les classes sauf Command et CommandWords. Ajout de la classe GameEngine et UserInterface. Cette fusion permet l'ajout d'une interface graphique pour le jeu.

7.18.7) addActionListener() permet d'ajouter à un composant, ici une entrée de texte, un écouteur qui permet de détecter dès qu'une action a été réalisée sur un composant.
actionPerformed() est appelé dès qu'une action a été réalisée sur un composant, ici une entrée de texte.

7.18.8) Ajout d'un bouton qui permet d'effectuer la commande look. Modification de la classe UserInterface, ajout d'un attribut pour le bouton et des procédures createGUI() et ActionPerformed(). Dans ActionPerformed, la procédure regarde maintenant quel type d'action l'utilisateur a fait, ici cliquer sur le bouton.

7.19) Le MVC (Model View Controller) permet à l'utilisateur d'interagir avec l'interface graphique. Cela permet aussi de plus facilement gérer l'interface sur le long terme et d'ajouter de nouveaux éléments graphiques.
7.19.2) Création du répertoire Images dans la racine du projet pour accueillir les images des différentes salles du jeu. Ajout des images dans le jeu. J'ai ajusté la dimension de la fenêtre à 800x800 pour bien voir l'image et le texte sans problème.

7.20) Création de la classe Item, qui contient : la description, le poids et le prix d'un objet.
Ajout d'en cette classe des getters pour tous les attributs. Ajout dans la classe Room, d'un attribut Item et : une procédure pour ajouter un item à cette salle et d'une méthode qui retourne l'objet de la salle. Modification de la méthode getLongDescription() pour accueillir les items.

7.21) La classe qui doit produire les String est la classe Item car c'est elle qui a les attributs des items. La classe qui doit utiliser les String produites par Item est Room dans la description d'une salle.
La classe qui doit l'afficher est GameEngine car c'est ici que l'on affiche tous.

7.21.1) Ajout d'un attribut Nom dans la classe Item qui permet d'identifier rapidement l'item pour le joueur. Ajout de la méthode getItem dans Room pour retourner l'item contenu dans la salle. Modification de la procédure look qui permet de vérifier l'égalité entre le nom de l'item fourni par le joueur et le nom de l'item de la pièce. La procédure retourne la longue description de l'item.

7.22) Dans Room : Ajout d'un attribut HashMap d'une liste d'item avec son nom en clé, modification de setItem à addItem car maintenant il y a une HashMap d'item avec son nom et pas seulement une variable. Ajout de getItem(final String pName) qui permet de retourner l'item souhaité grâce à son nom, ajout de getAllItem() qui retourne la HashMap de tous les items. Modification de getItemString qui renvoie la totalité des items, ligne par ligne avec leur description.
Dans Item : Modification de getDescription, pour ajouter le nom de l'item.
Dans GameEngine : modification de la procédure look pour regarder un item en particulier et que cela fonctionne avec la HashMap de la classe Room.

7.22.1) J'ai utilisé une HashMap car cela est plus pratique pour gérer un nombre raisonnable d'items par room.

7.23) Ajout de la commande back. Pour cela, dans la classe GameEngine, ajout de la procédure back qui retourne le joueur dans la salle où il était précédemment.
Ajout de la commande dans la constante dans CommandWords.

7.24-25) La commande ne fonctionne pas si l'on tape un deuxième mot, comme cela est convenu. Si l'on utilise la commande alors que l'on n'a pas bougé, cela envoie un message pour dire qu'on ne peut pas aller en arrière. Cependant lorsque nous appuyons sur back deux fois d'affilés, nous revenons au même endroit, ce qui peut être embêtant si nous voulons remonter notre chemin.

7.28.1) Ajout de la commande test dans la classe GameEngine qui lit un fichier, ligne par ligne, et exécute les commandes pour éventuellement voir les erreurs. Il test, toutes les directions, avec go, la commande back, eat, look en plus de regarder certains items.

7.28.3) Ajout de la commande RoomIs pour pouvoir checker dans quelle room le joueur se situe et quel item il y a, et ainsi voir si cela correspond bien avec la salle à laquelle il doit se trouver. Si ce n'est pas le cas alors le programme indique qu'il y a une erreur et arrête de tester la suite.

7.29) Ajout de la classe Player, avec 3 attributs, une String avec le nom du joueur, la room actuelle et un Stack avec toutes les anciennes rooms. L'ajout d'un inventaire ainsi que le maximum du poids de l'inventaire est à prévoir pour la suite. La classe contient toutes les méthodes qui sont directement reliées au joueur, comme goRoom découpée de façon à seulement déplacer le joueur et renvoyer la room dans laquelle le joueur se trouve. La méthode back qui fonctionne de la même manière. De plus, des accesseurs pour que gameEngine garde la salle actuelle.

7.30) Ajout des commandes take et drop. Ajout dans GameEngine, des gestions des commandes, qui affichent les messages de réussites ou d'erreurs. Dans Player, ajout d'un attribut qui permet de stocker un item, et ajouter des 2 méthodes take et drop qui permettent d'ajouter l'objet au player mais d'aussi supprimer ou ajouter l'item à la room actuelle.

7.31) Ajout d'une HashMap à l'attribut du player, avec en clé une string et un item en value, ce qui permet de stocker plusieurs items en fonction de leur nom. Ajout aussi des commandes "playerHas", "playerHasNot", "roomHasNot" qui me permet dans le fichier de test de tester toutes les fonctionnalités directement. Cela est très pratique.

7.31.1) Création de la classe ItemList qui permet de gérer un attribut HashMap qui stocke en clé une String et en value un Item. Cela permet d'éviter la répétition du code dans les classes Room et Player qui ont toutes les deux les mêmes types de HashMap. Cette classe contient toutes les interactions nécessaires avec une hashmap. Grâce au fichier test, les tests s'effectuent très rapidement !

7.32) Ajout de deux attributs à Player, le poids maximum qu'il peut porter ainsi que le poids qu'il porte actuellement. J'ai modifié la fonction take qui regarde si le poids ne dépasse pas le maximum avant de prendre l'objet et lui ajoute. J'ai aussi modifié la fonction drop qui enlève le poids de l'item du poids du joueur.

7.33) Ajout de la commande "inventory" qui permet au joueur de voir son inventaire. Modification de getItemString dans ItemList, la boucle foreach remplacée par un itérateur pour savoir la dernière itération et ainsi mettre ou pas une ligne (légère optimisation). Ajout dans Player de la méthode pour récupérer le String de l'inventaire qui est généré de la même manière que les items dans une room.

7.34) Ajout d'un item "prot" qui permet de doubler le poids maximum qu'un joueur peut porter. Dans la méthode eat, dans GameEngine, ajout du test du nom de l'item à manger pour voir si c'est le bon item.

7.35/7.35.1) Ajout d'une nouvelle classe CommandWord qui contient un enum avec le raccourci du nom des commandes. Cela permet d'associer le nom d'une commande à plusieurs noms. Par exemple, si je veux traduire mon jeu en français, la commande "go" peut-être utilisée mais aussi "aller". L'association se fait dans la classe CommandWords, dans une HashMap avec comme clé une String et un enum en valeur. Dans la classe GameEngine, la méthode processCommand qui sert à traiter les commandes a été modifiée avec un switch au lieu de tous les if/elseif, les cases sont les enum.

7.38) Ajout des commandes traduites en français en plus de celles en anglais. Le joueur peut donc écrire les commandes en français ou anglais.

7.41.1) Création d'un constructeur dans la classe CommandWord avec deux String en paramètre qui permet de mettre une commande en anglais et une en français.
Deux méthodes supplémentaires qui renvoient une, la commande en anglais, et l'autre la commande en français. Changement dans la classe CommandWords, dans le constructeur, une boucle qui initialise une HashMap avec en clé, la commande, et en valeur, le enum.

7.42) Ajout d'un attribut à la classe player qui contient les déplacements restants, un accesseur. Modification des fonctions goRoom et back pour enlever un déplacement restant à chaque déplacement réalisé et d'une condition pour checker s'il reste un déplacement. Si non, c'est la fin du jeu.
Création graphique d'une barre de progression en verticale à gauche de l'interface pour savoir combien de déplacements restants avant la fin du jeu et donc la perte. (JProgressBar)

7.43) Création de la classe Door en prévision de l'exercice 45. Deux attributs, un boolean pour savoir si la porte est fermée et l'objet avec lequel elle peut être ouverte. Un accesseur pour le boolean. Ajout d'une constante dans Room, d'une Room fermée pour pouvoir reconnaître facilement les portes fermées. La fonction back qui est réglée pour ne pas pouvoir franchir les portes fermées. Pour cela on récupère la direction où est l'ancienne room puis on regarde si la porte est fermée.

7.44) Création de la classe Beamer qui s'étend sur Item. Un seul attribut, celui de la room chargée. Accesseur pour savoir s'il y a une room dans l'attribut. Ajout de deux commandes,

“charge” et “fire” (aussi utilisable en français) qui permettent respectivement de charger la salle actuelle et de s’y téléporter depuis n’importe où. Il est à usage unique.

7.45) Ajout des commandes “close” et “open” qui permettent, quand on rajoute une direction, d’ouvrir ou de fermer une porte si l’on possède la clé. Ajout pour cela des accesseurs et getter des attributs de la classe Door.

7.46) Création des classes TransporterRoom, extends de Room, et de RoomRandomizer. La TransporterRoom override la fonction getExit de Room. Pour sortir une salle random, dans RoomRandomizer, on enregistre toutes les salles dans une ArrayList dans GameEngine, et on import “java.util.Random” pour retourner un pseudorandom int. Cela permet de faire une sortie vers une room aléatoirement. Ajout d’une fonction boolean “isTransporter” dans Room pour savoir si la salle est une TransporterRoom pour vider le stack des salles pour éviter le “back”.

7.46.1) Création de la commande alea et testmod. testmod permet de mettre le jeu en mode de test et d’utiliser seulement des commandes prévues à ce mode. Pour cela, ajout d’un attribut boolean pour déterminer si nous sommes dans ce mode. alea est une commande qui enlève l’aléatoire de RoomRandomizer pour la fixer sur une salle précise. Pour cela, teste si nous sommes dans le mode de test, puis si le second mot peut-être transformé en int entre et le nombre total de salle - 1, si cela correspond, ça sélectionne, à la position du nombre dans la arraylist de toutes les salles, une salle où la transporter room nous téléportera toujours. Pour annuler le non aléatoire, juste taper la commande alea.

7.46.2) La classe TransporterRoom est hérité de Room car c’est une sorte de Room mais qui a des attributs en plus. Cependant la classe RoomRandomizer n’hérite de rien car ce n’est pas une room mais son objectif est de stocker tous les éléments pour créer une fonction qui renvoie une fonction aléatoire. Enfin, la classe LockedDoor qui est une sorte de Door mais qui peut-être fermée ou ouverte si le joueur possède la clé.

7.47) Modification de la classe Command en classe abstraite, avec comme fonction abstraite : execute. Ensuite, création de toutes les commandes du jeu en classe avec chacune leur exécution à l’intérieur de la fonction execute. Modification de CommandWords avec l’ajout d’une HashMap<CommandWord, Command> pour permettre l’accessibilité des commandes, en String, à leur objet de classe.

7.47.1) Création de 5 paquetages :
-pkg_doors avec Door et LockedDoor
-pkg_items avec Item, ItemList et Beamer
-pkg_rooms avec Room, TransporterRoom et RoomRandomizer
-pkg_words avec toutes les classes des Command, CommandWords, CommandWord et Parser
-pkg_engine avec GameEngine, UserInterface et Player

7.48) Création d’un nouveau package pkg_characters avec Player et une nouvelle classe Character. Cette nouvelle classe contient 6 attributs : un String pour le nom, une ArrayList avec toutes les lignes de dialogue, une Room pour avoir la room où est le PNJ actuellement ,un int pour un compteur (pour afficher les différentes lignes de dialogue) et deux Item pour

savoir qu'est-ce que le PNJ veut et qu'est-ce qu'il donne. La classe contient un getter et setter pour la room actuelle, une méthode pour ajouter une ligne de dialogue et un getter des lignes de dialogues. Changement de Room pour afficher le nom et le dialogue du PNJ.

7.49) Création de la classe MovingCharacter dans la paquetage pkg_characters, avec deux nouveaux attributs : une ArrayList pour avoir toutes les salles où le pnj peut être et un compteur pour l'ordre des room.

7.49.1) Il n'y pas d'héritage entre les classe Item, Player et Character car ils n'ont pas d'attributs et fonctions en commun. Cependant, les classes Player et Character pourraient héritées d'une classe abstraite AbstractCharacter où ils auraient quelques points en commun. Je n'ai pas trouvé intéressant de l'introduire dans le jeu car il n'y en avait pas assez. Enfin, la classe MovingCharacter hérite de Character car c'est un Character mais avec des attributs qui permettent de le faire bouger.

IV) Je certifie n'avoir recopié la moindre ligne de code (sauf les fichiers zuul-.jar qui sont fournis évidemment). Je déclare donc n'avoir plagié personne.