Time Runner
http://esiee.fr/~robiny

Naji Astier, Yohann Robin, 6B

A3P 2013

Sommaire

I Présentation

I.A Auteurs

Naji ASTIER, Yohann ROBIN.

I.B Thème

Un personnage voyage dans le temps à l'aide d'une télécommande.

I.C Résumé du scénario

Le personnage lors d'un voyage temporel au Moyen-âge a brisé la télécommande lui permettant de voyager dans le temps. Il aura donc besoin de récupérer des matériaux afin de bidouiller sa télécommande pour pouvoir espérer rentrer chez lui. Mais son voyage sera semé d'embûches.

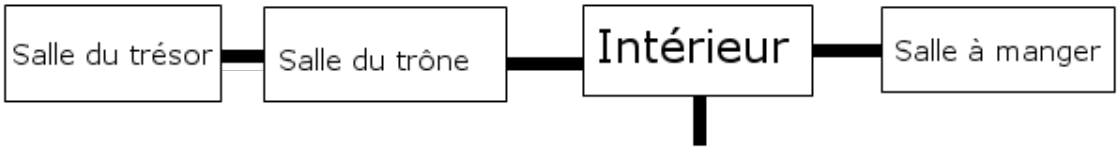
I.D Commentaires

Nous avons privilégié les exercices au détriment du scénario. De plus, notre jeu existera en deux versions. Une version dite "zuul" qui est peu développé du scénario et répondant aux exercices ainsi qu'une version plus complète.
Cette version complète est en réalité une véritable jeu-vidéo. En effet, il s'agit d'un action-rpg réalisé à l'aide de la librairie Slick2D. Ce jeu ne répondant pas essentiellement aux exercices, nous avons préféré le garder pour le présenter le jour de projets.

I.E Plan

Voici le plan de l'extérieur du château.

Et le plan de l'intérieur du château (image non disponible pour le moment).



I.F Scénario détaillé

Il y aura quatre niveaux divers et variés. Le premier se déroule au Moyen-âge, le personnage devra faire face à ses ennemis et trouver le composant de base pour le faire avancer aléatoirement dans le temps.
Le deuxième niveau, après le câblage du composant fait voyager notre protagoniste à la Renaissance où il croisera Léonard de Vinci et des personnages interactifs qui lui indiqueront quoi faire.
Le troisième niveau l'enverra en 1940 où notre personnage se retrouvera au milieu d'un conflit planétaire. Il devra se frayer un chemin à travers les nazis et d'autres ennemis afin de trouver l'équivalent du composant manquant de cette époque.
Le dernier niveau, le personnage a réparé sa télécommande mais celle-ci l'envoie dans le futur. En effet, il a la possibilité de laisser sa télécommande et de retourner dans son présent.

I.G Détail des lieux, items, personnages

Le premier niveau est composé de sept pièces. Le personnage commence dans la cour où il peut se diriger vers la tour de guet, la tour de garde ainsi que le château. Une fois dans la château, le personnage a le choix entre la salle à manger, la salle du trône et la possibilité de sortir du château.
Items : chaque pièce possède des items particuliers. En effet dans la salle de garde, des armes, dans la salle à manger, de la nourriture et enfin dans la salle du trésor, de l'argent et le beamer.
Personnages : de nombreux gardes sont réparties sur l'ensemble du niveau avec des paysans et autres. Enfin le roi siègeant sur son trône.

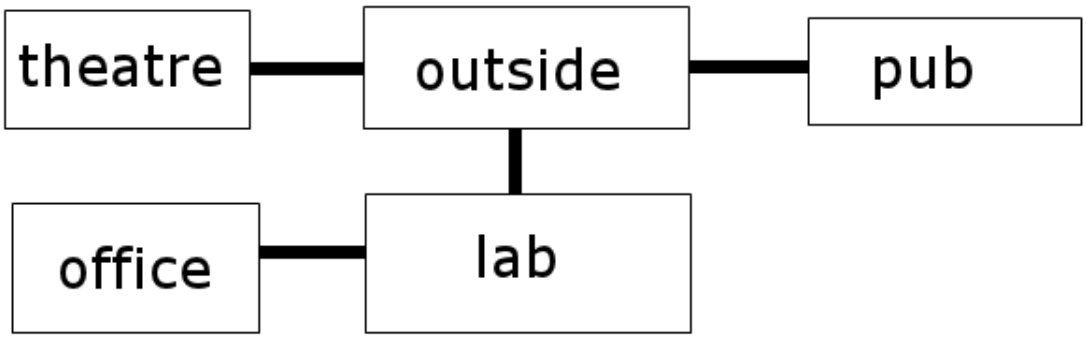
I.H Situations gagnantes et perdantes

Le but de ce niveau est de récupérer de l'étain qui est une danrée rare à cettte époque. Le problème est que ce matériau est stocké dans la salle du trône, il va s'en dire, que si notre héros réussit à trouver l'étain, cela signifie qu'il a tué l'ensemble de ces ennemis et à donc à terminé le niveau.

II Réponses aux exercices

7.1 - découverte de zuul-bad

zuul-bad.jar contient un jeu textuel où le joueur peut se déplacer en écrivant la commande “go” suivie d’une direction cardinale. Les commandes utilisables sont “go” pour se déplacer, “help” pour afficher une aide et “quit”.
Les pièces sont “outside”, “theatre”, “pub”, “lab” et “office”. Voici leur agencement :



7.5 - printLocationInfo

On a implémenté la commande **printLocationInfo()** de la même manière que dans le livre. On a ensuite remplacé l’ancien affichage des sorties dans **printWelcome()** et dans **goRoom()** par un appel de la méthode **printLocationInfo()**. Cela nous permet d’éviter la duplication de code.

7.6 - getExit

On a fait les changements demandés. Encore une fois, ce changement permet d’éviter la duplication de code.

7.7 - getExitString

La méthode **getExitString()** dans la classe Room s’occupe de concaténer la liste des sorties dans une String qu’elle retourne. Cette String est récupérée dans la méthode **printLocationInfo()** du **GameEngine**.

7.8 - HashMap, setExit

Les sorties sont maintenant stockées dans une HashMap. Celle-ci associe à une String contenant la direction une Room. Maintenant, il faut appeler la méthode **setExit()** de la classe Room autant de fois qu’une direction doit être ajoutée.

7.9 - keySet

La méthode keySet retourne les indices du tableau associatif, la HashMap. Par exemple pour la HashMap des direction : “north”, “east”, “south”, “west”.

7.10 - getExitString CCM ?

La méthode getExitString fonctionne tout d'abord en utilisant la méthode keySet de l'HashMap afin de récupérer la liste des directions définies. Cette liste est stockée dans une collection de type Set. Celle-ci accepte les clés de manière unique. Ensuite on parcourt cette liste à l'aide d'une boucle for each et concatène une chaîne de caractères contenant toutes les directions.

7.11 - getLongDescription

La méthode getLongDescription renvoie maintenant la description de la pièce suivie de la liste des sorties possibles.

7.14 - look

La commande a été ajoutée en commençant par ajouter la String "look" dans le tableau validCommands de la classe CommandWords. Ensuite dans la méthode processCommand() de la classe Game, il faut rajouter une condition :

```
else if (commandWord.equals("regarder"))
{
    look();
}
```

Enfin, on rajoute la méthode look() dans le GameEngine.

```
private void look()
{
    System.out.println(currentRoom.getLongDescription());
}
```

7.15 - eat

On a ajouté la commande eat de la même manière que la commande look.

7.16 - showAll, showCommands

La méthode showAll() de la classe CommandWords affiche une String contenant la liste des commandes. Pour la générer, elle parcourt le tableau validCommands à l'aide d'une boucle for each et concatène toutes les valeurs. La méthode showCommands se contente d'appler la méthode showAll.

7.17 - changer Game ?

La méthode printHelp de la classe Game affichera bien cette commande, mais il faudra toujours afficher la méthode processCommand.

7.18 - getCommandList

A présent, la méthode getCommandList renvoie la liste de toutes les commandes sous la forme d'une String. C'est la méthode printHelp du GameEngine qui s'occupe d'afficher cette String.

7.18.5 - Les objets Room ...

L'ensemble des Room du premier niveau est maintenant sauvegardé dans une HashMap static. Celle-ci est donc accessible depuis n'importe quelle classe.

7.18.7 - addActionListener et actionPerformed

addActionListener est une méthode permettant d'ajouter un listener sur une action de l'utilisateur. Un listener permet de détecter si une action est réalisé ou pas.
actionPerformed est une méthode abstraite de l'interface ActionListener. Elle est appelée lorsque un listener capture une action et exécute alors des instructions.

7.18.8 - Ajout d'un bouton

L'ajout d'un bouton se fait par la création d'un nouvel objet JButton. Celui-ci est ensuite ajouté à l'interface. Ensuite, on utilise addActionListener pour enregistrer un listener sur le bouton. La méthode actionPerformed s'occupera d'exécuter la commande associée au bouton.

7.20 - Item

On commence par créer une classe Item avec plusieurs attributs : un attribut de type String description, la description de l'Item, et un attribut de type double weight, le poids de l'Item. Ensuite on ajoute un attribut de type Item dans la classe Room et on crée les accesseurs et modificateurs nécessaires pour gérer l'Item.

7.21 - item description

La classe Item doit produire la chaîne de caractères décrivant l'objet. La classe Room récupère cette description et la concatène avec la description de la pièce. C'est la classe GameEngine qui affiche cette description car c'est elle qui gère l'affichage de la description de la pièce.

7.22 - items

Dans la classe Room, l'attribut de type Item est maintenant devenu un attribut de type HashMap<String,Item>. Les items étant maintenant dans une collection, il faut créer une méthode addItem(nom, Item) pour ajouter un item et getItem(nom) qui récupère un item.

7.23 - back

L'ajout de la commande back nécessite un nouvel attribut previousRoom dans le GameEngine ayant pour rôle de sauvegarder la pièce précédemment visitée. Ensuite, la commande back ne fera que donner pour valeur previousRoom à la currentRoom puis de raffraîchir l'affichage.

7.24 - back test

Si le joueur écrit un second mot après la commande back, celle-ci fonctionne toujours correctement puisque le second mot n'est pas utilisé.

7.25 - back back

Nous avons écrit la fonction de telle sorte quelle inverse la pièce actuelle et la pièce précédente, par conséquent, nous pouvions nous en servir à l'infinie.

7.26 - Stack

Une collection de type Stack est une pile où on ne peut accéder uniquement au dernier élément ajouté. Cela qui est adapté à la sauvegarde d'un itinéraire. Avec la méthode push de la Stack, on ajoute une Room dans l'itinéraire. Avec la méthode empty de la Stack, on vérifie si l'itinéraire n'est pas vide. Avec la méthode pop de la Stack, on récupère la dernière Room sauvegardée et celle-ci est automatiquement supprimée de la Stack.

7.29 - Player

La classe Player contient le nom du joueur, le poids maximum qu'il peut porter ainsi que sa position actuelle et l'itinéraire qu'il a parcouru. On doit donc déplacer les méthodes associées à sa position et à l'itinéraire.

7.30 - take et drop

On ajoute un attribut de type Item à la classe Player. Avec les accesseurs et modificateurs nécessaires pour pouvoir faire les commandes take et drop.

7.31 - porter plusieurs items

On change l'attribut de type Item dans la classe Player en une HashMap<String,Item> pour que le joueur puisse porter plusieurs items. Il faut alors adapter le code en conséquence.

7.31.1 - ItemList

La classe ItemList contient donc une gestion commune d'une liste d'item. On a pour seul attribut une HashMap<String,Item>. On a pour méthode : addItem, removeItem, getItem, getTotalWeight (renvoie le poids total de la liste) et getItemString (renvoie la description de l'ensemble des Item). Finalement, il faut remplacer les HashMap dans les classes Player et Room par une ItemList et retirer les méthodes associées.

7.32 - poids max

Il faut rajouter une condition dans la méthode take. Cette condition vérifie si le poids actuelle de l'inventaire du joueur auquel on ajoute la poids de l'Item voulu, est supérieur au poids maximum que peut porter le joueur.

7.33 - inventaire

Rajouter une commande se fait toujours de la même manière que précédemment.

7.34 - magic cookie

Il faut mettre à jour la commande eat pour qu'on puisse manger des items. On rajoute alors un attribut dans la classe Item : edible, qui permet de savoir si l'objet est commestible ou non. Ensuite pour manger un Item, on fait les mêmes vérifications que dans la commande drop, on vérifie d'abord si le joueur possède l'Item dans son inventaire ou pas.

7.35.1 - switch

```
    /**
     * Given a command, process (that is : execute) the command.
     * If this command ends the game, true is returned, otherwise false is
     * returned.
     *
     * @param commandLine the command line
     */
    public void interpretCommand(final String commandLine)
    {
        this.aGui.println("\n" + commandLine);
        Command vCommand = aParser.getCommand(commandLine);
        CommandWord vCommandWord = vCommand.getCommandWord();

        switch(vCommandWord)
        {
            case UNKNOWN :
                this.aGui.println("I don't know what you mean...");
            break;

            case HELP :
                this.printHelp();
            break;

            case GO  :
                this.goRoom(vCommand);
            break;

            case EAT :
                this.eat(vCommand);
            break;

            case LOOK :
                this.look();
            break;

            case QUIT :
                if(vCommand.hasSecondWord())
                {
                    this.aGui.println("Quit what?");
                }
                else
                {
                    endGame();
                }
            break;

            /* Pleins d'autres cases.... */
        }
    }
```

7.37 - Translate

Oui il suffit de modifier la classe CommandWords, en ajoutant la commande en Français dans la HashMap. Ainsi dans la HashMap, plusieurs clés pointent vers la même commande.

7.38 - help

Nous remarquons que la commande fonctionne mais affiche un message dans la langue d'origine.

7.40 - look

Il n'y a pas besoin de changer le type de CommandWord.

7.42 - time limit

On a choisi d'ajouter un compteur de déplacement du joueur. On limite donc ceux-ci et si le joueur voit son compteur tombé à zéro, il perd la partie.

7.43 - trap door

Actuellement, le passage d'une pièce à l'autre est défini dans les deux pièces. Il suffit de retirer une des deux sorties pour créer une trapdoor.

7.44 - beamer

On utilise l'héritage pour créer ce nouvel objet. La classe Beamer hérite de la classe Item et ajoute de nouveaux attributs : la pièce à enregistrer ainsi que l'état du beamer (chargé ou pas).

7.46 - transporter room et RoomRandomizer

On crée une classe RoomRandomizer. Celle-ci ressoit en paramètre dans son constructeur une liste de Room qu'elle convertie en tableau. Lorsque la méthode nextRoom() est appelé sur un objet de type RoomRandomizer, l'indice de la Room dans le tableau à renvoyer est généré aléatoirement à l'aide d'un objet de type Random.

7.46.1 - alea

La commande alea permet de supprimer le mode aléatoire du RoomRandomizer. En écrivant "alea cour", le RoomRandomizer renverra forcément vers la cour. En écrivant à nouveau "alea", le RoomRandomizer repasse en mode aléatoire. Nous n'avons pas fait de mode de test, puisqu'il suffit de commenter les lignes nécessaires pour passer le projet en production.

7.47 - abstract Command

Chaque commande est maintenant définie dans une classe. Nous n'avons donc plus besoin du switch qu'il y avait dans interpretCommand dans la classe GameEngine.

7.47.1 - Paquetages

Nous avons maintenant quatre paquetages : pkg_commands, pkg_items, pkg_rooms et pkg_others. Finalement, il ne reste que la classe Game qui n'est pas mise dans une paquetage.

7.48 - Character

Nous avons créer une classe Character. Les Character sont très semblables aux Player, cependant par un soucis de clareté, nous n'avons pas fait hérité la classe Character de la classe Player. Les personnages possèdent un nom, une phrase à dire, un objet recherche ainsi qu'une position.

7.49 - moving character

En l'état, nos Character sont déjà aptes à être déplacé à l'aide de la méthode moveCharacter().

7.50 - maximum

La méthode calculant le maximum entre deux entiers est max. Voici sa signature :

```
public static int max(int a, int b)
```

7.52 - curentTimeMillis

```
/**
 * Classe de test pour mesurer le temps d'exécution
 * d'une boucle de comptage.
 */
public class Test
{
    public static void essai ()
    {
        /* On prend l'heure avant la boucle */
        long vTemps1 = System.currentTimeMillis();

        /* Il faut 0ms pour compter jusqu'à 100
         * On a mis 2147483647 à la place
         */
        for(int i = 1; i < 2147483647; i++)
        {
        }

        /* On prend l'heure après la boucle */
        long vTemps2 = System.currentTimeMillis();

        System.out.println ("La boucle a mis " + (vTemps2 - vTemps1) + "ms.");
    }
}
```

7.53 - main

On a ajouté une méthode main en respectant la signature requise. En créeant un fichier .jar, le jeu peut maintenant se lancer sans BlueJ.

7.56 - test static

Les méthodes statiques peuvent être appelés depuis n'importe où, dans une méthode statique ou d'instance. Pour appeler une méthode d'instance dans une méthode static, il faut tout d'abord créer une instance de classe.

7.57 - numberOfInstances

Il est possible d'ajouter un attribut entier static dans une classe servant à compter les instances. Ensuite, on incrémente cet attribut à chaque fois que le constructeur est appelé.

```
1  /**
2   * Classe de test pour compter le nombre d'instances
3   * crées à partir d'une classe.
4   */
5  public class Test
6  {
7      private static int instances = 0;
8
9      public Test()
10     {
11         Test.instances++;
12         /* On peut faire d'autres choses dans le constructeur */
13     }
14
15     public static int numberOfInstances()
16     {
17         return Test.instances;
18     }
19
20     /* d'autres méthodes */
21 }
```

III Mode d'emploi

Au démarrage, le jeu propose de lancer la version zuul ou la version slick2D. Il se peut aussi que le logiciel détecte une mise-à-jour.
La version slick2D se joue à l'aide des touches "qsdz" pour se déplacer, "x" pour parler ("F3" pour activer le debug).
La version zuul se joue en ligne de commande ou à l'aide des quelques bouttons disponibles. Pour se déplacer, il suffit de taper la commande "go" suivie de la direction. Il est également possible de se déplacer à l'aide des flèches du clavier ou à l'aide des quatre bouttons disponibles à cet effet. Pour obtenir la liste des commandes, il faut taper la commande "help".

IV Déclaration anti-plagiat

Aucune ligne de code n'a été empruntée à qui que ce soit, en dehors des fichiers zuul-*.jar fournis dans la liste officielle des exercices.
Concernant les images, trois d'entres elles sont tirées du jeu complet Time Runner non disponible pour le moment. Les autres sont tirées de zuul-with-images.