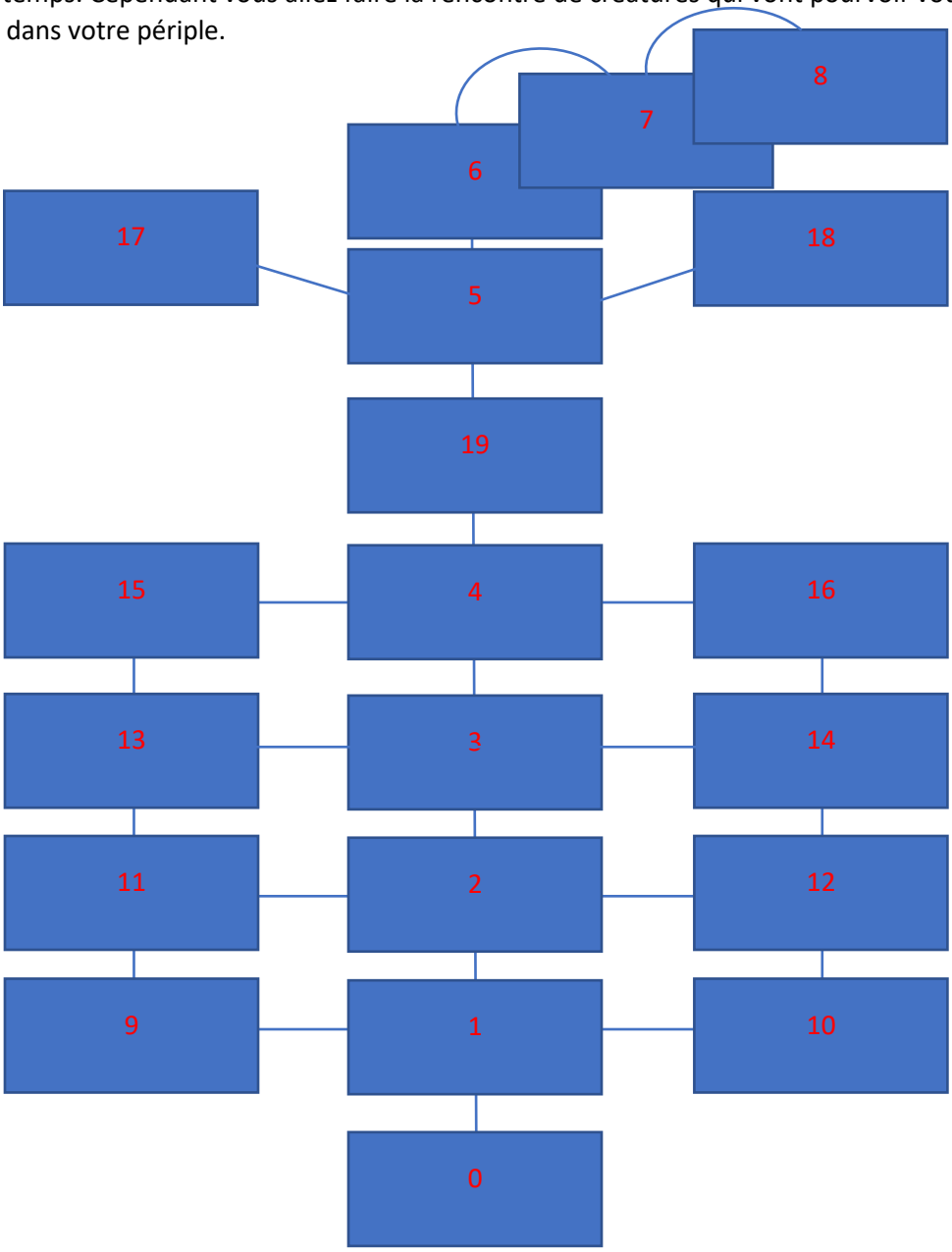
Moonnknown

SANGCHANMAHOLA Erwan

Phrase-type : Sur la Lune, vous êtes perdu, votre but est de retourner d'où vous venez.

Résumé du scénario : Vous vous réveillez inconscient sur la Lune, vous devez trouver un moyen de retourner chez vous, il ne vous reste seulement de quoi vous nourrir et vous hydrater pendant quelque temps. Cependant vous allez faire la rencontre de créatures qui vont pourvoir vous aider (ou non) dans votre périple.

Plan :



Salle d'exploration : 0,1,2,3,4,9,10,11,12,13,14,15,16

Salle d'énigmes : 5,6,7,17,18

Salle de la Tour : 6,7,8

Dans toutes les salles se trouvent un objet (utile ou pas)

Items :

-caillou

-champignon faisant office de « magic cookie »

-Plaque mystérieuse : objet nécessaire à l'accès de la tour

Personnage :

-Protagoniste

Situation gagnante :

-finir le 3e étage de la tour

Situation perdante :

-mourir de faim

-ne pas finir le 3e étage de la tour

-fin du timer

Ce qu'il reste à faire :

-Implémenter les fonctionnalités des Room

-Définir les « classes » et les ennemis

-Définir les énigmes

Ex7.5

On a dû créer une procédure printLocationInfo() pour éviter une duplication de code.

Ex 7.6

Voir code

Ex 7.7

Comme les attributs sont privés, la classe Room n'a pas accès aux attributs de Game et vice-versa, donc nous devons appeler les méthodes des classes respectives pour obtenir ces informations.

Ex 7.8-7.8.1

Voir code

Ex 7.9

keySet() renvoie un Set des éléments de type choisi de l'objet.

Ex 7.10

On créer un String vReturnString qui va contenir le String à retourner.

On crée une variable vkeys de type Set<String> puisqu'on veut les directions possibles qui sont de type String de la hashmap aExits

On fait une boucle : pour chaque String « exits » de vkeys on ajoute la sortie au String vReturnString.

Puis on retourne vReturnString.

Ex 7.11

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

Ex7.14

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

Ex7.15

```
private void eat(final Command pCommand){
    if(!pCommand.hasSecondWord()) {
      this.aGui.println("Eat what?");
      return;
    }
    String vItemName =
pCommand.getSecondWord().substring(0,1).toUpperCase()+pCommand.getSecondWord().substri
ng(1).toLowerCase();
    if (!this.aPlayer.getItemListString().toLowerCase().contains(vItemName.toLowerCase()))
      this.aGui.println("You don't have this");
    else {
      switch(vItemName){
        case "Champignon":
        this.aPlayer.useItem(this.aPlayer.getItem("Champignon"));
        this.aPlayer.setMaxWeight(this.aPlayer.getMaxWeight()*2);
        this.aGui.println("Champignon successfully eaten."+ '\n'+ "You can carry more items!");
        break;
        default:
        this.aGui.println("Item "+vItemName+" is not edible.");
        break;
      }

    }
```

```
  }
```

Ex7.16

```
public void showCommands(){
      System.out.println(aValidCommands.getCommandList());
  }
```

Ex7.17

Non, seul la classe CommandWords va être modifié, ainsi que la methode processCommand()

Ex7.18

```
public String getCommandList(){
      String vS="";
      for(String vCommand : sValidCommands){
         vS=vS+vCommand+" ";
      }
      return vS;
  }


private void printHelp(){
      System.out.println("You are lost. You are alone."+"\n"+
         "You wander around, you can see the Earth from here"+"\n"+"\n"+
         "Your command words are:");
         This.aParser.showCommands();
  }
```

Ex7.18.5

On créer une HashMap en attribut dans Game qui va stocker toutes les room.

Ex7.18.6

On a déplacé une grande partie de Game dans GameEngine.

On ajoute un attribut a Room ce qui permet d'afficher des images a l'aide de la classe UserInterface qui génère une interface graphique. »

Ex 7.18.7

addActionListener() permet de ré cupérer une action (appui du boutton,…)

et actionPerformed() permet de récupérer ce qui a déclenché de l'action.

Ex 7.18.8

Dans la méthode createGUI(), on ajoute la ligne « this.aButton= new JButton("look"); », ce qui permet de créer un bouton avec un String.

On ajoute la ligne « vPanel.add(this.aButton, BorderLayout.WEST); » pour ajouter le bouton a la bordure gauche de la fenêtre graphique.

On ajoute « this.aButton.addActionListener(this); » pour qu'on puisse détecter une action sur le bouton.

```
 public void actionPerformed( final ActionEvent pE )
   {
      // no need to check the type of action at the moment.
      // there is only one possible action: text entry
      if(pE.getSource()==this.aButton) this.aEngine.interpretCommand("look");
      else{
         this.processCommand();
      }
   }
```

Ici, on interprète la commande « look » lorsque le bouton est pressé.

Ex 7.20-.22

On ajoute la classe Item :

```java
public class Item
{

    private int aItemW;
    private String aItemD;
    private String aItemN;
    /**
     * Create Item with name, description and weight
     * @param pN Item's Name.
     * @param pD Item's Description.
     * @param pW Item's Weight.
     */


    public Item(final String pN, final String pD, final int pW )
    {
        this.aItemW  = pW;
        this.aItemD = pD;
        this.aItemN = pN;
    }
    /**
     * @return The Item's weight.
     */
    public int getWeight()
    {
        return this.aItemW;
    }
    /**
     * @return The Item's Description.
     */
    public String getDescription()
    {
        return this.aItemD;
    }
    /**
     * @return The Item's Name.
     */
    public String getName()
    {
        return this.aItemN;
    }
    /**
     * @return String with the Name, Description, Weight of the Item.
     */
    @Override public String toString(){
        return this.aItemN+": "+this.aItemD+" ("+this.aItemW+" g"+")";
    }

}
```

Et on ajoute quelques méthodes dans la classe Room, ainsi qu'un attribut HashMap.

```
public String getLongDescription(){
    return this.aDescription +
    ".\n" + this.getExitString() +
    ".\n" + this.getItemListString();
}
/**
 * Return a string with the Romm's items.
 */
public String getItemListString(){
    return "Item(s):"+this.aItems.getItemListString();
}
/**
 * Return a string describing the room's image name
 */
public String getImageName()
{
    return this.aImageName;
}
/**
 * Return the room that is reached if we go from this room in direction
 * "direction". If there is no room in that direction, return null.
 */
public Item getItem(final String pItem){
    return this.aItems.getItem(pItem);
}
/**
 * Return a string describing the room's image name
 */
public String getItemName(final Item pItem)
{
    return this.aItems.getItemName(pItem);
}
/**
 * Place an Item into the Room
 * @param pNItem String with which the specified Item is to be associated.
 * @param pItem Item to be associated with the specified String.
 */
public void addItem(final String pNItem,final Item pItem){

    this.aItems.addItem(pNItem, pItem);
}
/**
 * Remove an Item into the Room
 * @param pNItem String with which the specified Item is to be deleted.
 */
public void removeItem(final String pNItem){

    this.aItems.removeItem(pNItem);
}
```

Ex7.23

On ajoute un attribut Room aPreviousRoom dans GameEngine qui va stocker la Room précédente.

La méthode back() :

this.aCurrentRoom=this.aPreviousRoom; // on suppose que aPreviousRoom existe

this.aGui.println(aPlayer.getCurrentRoom().getLongDescription());

if(this.aPlayer.getCurrentRoom().getImageName() != null)

this.aGui.showImage(this.aPlayer.getCurrentRoom().getImageName());

Ex 7.24-.25

Il ne se passe rien car pour l'instant, seule la dernière room est stockée.

Ex7.26

```
private void back()
{
    if(this.aPlayer.getStackRooms().size()>0){
        this.aPlayer.setCurrentRoom(  (Room)this.aPlayer.peekStackRooms());
        this.aGui.println(aPlayer.getCurrentRoom().getLongDescription());
        if(this.aPlayer.getCurrentRoom().getImageName() != null)
            this.aGui.showImage(this.aPlayer.getCurrentRoom().getImageName());
        this.aPlayer.popStackRooms();
    }
    else{
        this.aGui.println("You can't go back anymore.");
    }
}
```

Ex7.28.1

```
private void test(final Command pCommand){
    if(pCommand.hasSecondWord()){
        String vS=pCommand.getSecondWord();

        if(!vS.contains(".txt")){
            aGui.println("Abra,Kadabra");
            vS=vS+".txt";
        }
        try{
            Scanner vSc = new Scanner (new File(vS));

            while(vSc.hasNextLine()){
                interpretCommand(vSc.nextLine());

            }
        }

        catch(final FileNotFoundException pFNFE){
            this.aGui.println("File not found");
        }
        catch(final Exception pE){
            this.aGui.println( "Exception message -> " + pE.getMessage() );
        }
        finally{

        }

    }
    else{
        this.aGui.println("You have add a file name");
    }
}
```

Ex7.28.2

All.txt (teste toutes les possibilités)

Speed.txt (arrive à la salle finale le plus rapidement)

Court.txt (fichier test)

Ex 7.29

```
public class Player
{
    private Room aCurrentRoom;
    private Stack aStackRooms;
    private HashMap<String,Room> aRooms;
    //private HashMap<String,Item> aItems;
    private ItemList aItemList;
    private int aMaxWeight;
    private int aCurrentWeight;
    /**
     * Create a Player
     */
    public Player()
    {
        this.aStackRooms=new Stack();
        this.aRooms=new HashMap<String, Room>();
        this.aItemList=new ItemList();
        this.aMaxWeight=5000;
        this.aCurrentWeight=0;
    }

    /**
     * Return a string with the Player's items.
     */
    public String getItemListString(){
        return this.aItemList.getItemListString();
    }
    /**
     * Return an item owned by the Player.
     * @param   pItem The item' String.
     */
    public Item getItem(final String pItem){
        return this.aItemList.getItem(pItem);
    }
    /**
     * Take an item from the current room and add the item's weight to the player.
     * * @param   pItem The item' String.
     */
    public void takeItem(final Item pItem){
        this.aCurrentWeight+=pItem.getWeight();
        this.aItemList.addItem(pItem.getName(),pItem);
        this.aCurrentRoom.removeItem(pItem.getName());
    }
    /**
     * Use an item and remove the item's weight to the player.
     * * @param   pItem The item' String.
     */
    public void useItem(final Item pItem){
        this.aCurrentWeight-=pItem.getWeight();
        this.aItemList.removeItem(pItem.getName());
    }
```

```java
public void dropItem(final Item pItem){
    this.aCurrentWeight-=pItem.getWeight();
    this.aCurrentRoom.addItem(pItem.getName(),pItem);
    this.aItemList.removeItem(pItem.getName());
}
/**
 * Return the Player's weight
 */
public int getCurrentWeight(){
    return this.aCurrentWeight;
}
/**
 * Return the Player's max weight
 */
public int getMaxWeight(){
    return this.aMaxWeight;
}
/**
 * Set Player's weight
 */
public void setCurrentWeight(final int pCW){
    this.aCurrentWeight=pCW;
}
/**
 * Set Player's max weight
 */
public void setMaxWeight(final int pMW){
    this.aMaxWeight=pMW;
}
/**
 * Return the Player's current room
 */
public Room getCurrentRoom(){
    return this.aCurrentRoom;
}
/**
 * Set the Player's current room
 */
public void setCurrentRoom(final Room pRoom){
    this.aCurrentRoom=pRoom;
}
/**
 * @return Stack which contains previous rooms
 */
public Stack getStackRooms(){
    return this.aStackRooms;
}
/**
 * @return The last Room registered.
 */
public Object peekStackRooms(){
```

```java
/**
 * @return The last Room registered.
 */
public Object peekStackRooms(){
    return this.aStackRooms.peek();
}
/**
 * Remove the last room registered.
 */
public void popStackRooms(){
    this.aStackRooms.pop();
}
```

```java
/**
 * Add a Room onto the top of aStackRooms.
 * @param pA Room to be added to aStackRooms.
 */
public void pushRoom(final Room pA){
    this.aStackRooms.push(pA);
}
```

```java
private void goRoom(final Command pCommand)
{
    if(!pCommand.hasSecondWord()) {
        // if there is no second word, we don't know where to go...
        this.aGui.println("Go where?");
        return;
    }

    String vDirection = pCommand.getSecondWord();

    // Try to leave current room.
    Room vNextRoom = this.aPlayer.getCurrentRoom().getExit(vDirection);

    if (vNextRoom == null)
        this.aGui.println("There is no door!");
    else {
        this.aPlayer.pushRoom(this.aPlayer.getCurrentRoom());
        this.aPlayer.setCurrentRoom(vNextRoom);
        this.aGui.println(this.aPlayer.getCurrentRoom().getLongDescription());
        if(this.aPlayer.getCurrentRoom().getImageName() != null)
            this.aGui.showImage(this.aPlayer.getCurrentRoom().getImageName());
    }
}
```

Ex7.30

```
/**
 * Take an item
 */
private void take(final Command pCommand){
    if(!pCommand.hasSecondWord()) {
        this.aGui.println("Take what?");
        return;
    }

    String vItemName = pCommand.getSecondWord().substring(0,1).toUpperCase()+pCommand.getSecondWord().substring(1).toLowerCase();

    if (!this.aPlayer.getCurrentRoom().getItemListString().toLowerCase().contains(vItemName.toLowerCase()))
        this.aGui.println("This Item doesn't exist");
    else {
        if(this.aPlayer.getCurrentWeight()+this.aPlayer.getCurrentRoom().getItem(vItemName).getWeight() > this.aPlayer.getMaxWeight()){
            this.aGui.println("You can no longer carry objects, your inventory is too heavy .");
        }
        else{
            this.aPlayer.takeItem(this.aPlayer.getCurrentRoom().getItem(vItemName));
            this.aGui.println("Item "+vItemName+" successfully taken.");
        }
    }
}
/**
 * Drop item.
 */
private void drop(final Command pCommand){
    if(!pCommand.hasSecondWord()) {
        // if there is no second word, we don't know what to take...
        this.aGui.println("Drop what?");
        return;
    }

    String vItemName = pCommand.getSecondWord().substring(0,1).toUpperCase()+pCommand.getSecondWord().substring(1).toLowerCase();

    if (!this.aPlayer.getItemListString().toLowerCase().contains(vItemName.toLowerCase()))
        this.aGui.println("This Item doesn't exist");
    else {
        this.aPlayer.dropItem(this.aPlayer.getItem(vItemName));
        this.aGui.println("Item "+vItemName+" successfully dropped.");

    }
}
```

Ex7.31

```java
private HashMap<String, Item> aItemList;

/**
 * Constructeur par defaut
 */
public ItemList()
{
    this.aItemList=new HashMap<String, Item>();

}

/**
 * @Return The item associated with pItem
 * @param The item's name
 */
public Item getItem(final String pItem){
    return this.aItemList.get(pItem);
}
/**
 *
 * @Return The String associated with pItem
 * @param The item
 */
public String getItemName(final Item pItem){
    return this.aItemList.get(pItem).getName();
}
/**
 * Return a string with the Romm's items.
 */
public String getItemListString(){
    String vReturnString = "";
    Set<String> vKeys= this.aItemList.keySet();
    for(String vItem : vKeys){
        vReturnString +=" "+ vItem;
    }
    return vReturnString;
}
/**
 * Add item into the hashmap
 */
public void addItem(final String pNItem,final Item pItem){

    this.aItemList.put(pNItem, pItem);
}
/**
 * Remove an Item into the Room
 * @param pNItem String with which the specified Item is to be deleted.
 */
public void removeItem(final String pNItem){
    this.aItemList.remove(pNItem);
}
```

Ex7.32

Voir plus haut

Ex7.33

```
/**
 * Show your items.
 */
private void items(){
    this.aGui.println("You are carrying :"+this.aPlayer.getItemListString()+" "+this.aPlayer.getCurrentWeight()+ " g in total.");
}
```

Ex7.34

```
/**
 * Eat
 */
private void eat(final Command pCommand){
    if(!pCommand.hasSecondWord()) {

        this.aGui.println("Eat what?");
        return;
    }

    String vItemName = pCommand.getSecondWord().substring(0,1).toUpperCase()+pCommand.getSecondWord().substring(1).toLowerCase();

    if (!this.aPlayer.getItemListString().toLowerCase().contains(vItemName.toLowerCase()))
        this.aGui.println("You don't have this");
    else {
        switch(vItemName){
            case "Champignon":
            this.aPlayer.useItem(this.aPlayer.getItem("Champignon"));
            this.aPlayer.setMaxWeight(this.aPlayer.getMaxWeight()*2);
            this.aGui.println("Champignon successfully eaten."+ '\n'+ "You can carry more items!");
            break;

            default:
            this.aGui.println("Item "+vItemName+" is not edible.");
            break;
        }

    }
}
```