RÉALISATION D'APPLICATION (2020)

Itération 4

Groupe 2

03 DÉCEMBRE 2018

1 7.42 TIME OUT

1.1 7.42.0

On appel la fonction dans la classe GameView dans la methode update on code la methode dans GameModel Au lieu d'utiliser le temps on compte le nombre de rooms

```
public void timeOut(){
    if(pastRooms.size()==20) {
        gameView.show("Time ouuuuuuuutt\n");
        interpretCommandString("quit");
    }
}
```

2 7.43

Il est relativement facile d'implémenter cette fonctionnalité, en effet, depuis notre fichier **RoomData.csv**, chaque salle de jeu possède l'information de sortie, vous pouvez le voir comme un graph orienté.
Par exemple, nous pouvont rendre la salle vestiaire comme une salle couloir à sens unique, vous pouvez acceder à la salle vestiaire via la porte de la salle de repos, mais la salle de vestiaire n'a qu'une seul porte qui mène à l'aile droite, enfin l'aile droite ne possède pas de porte qui permet d'y retourner.

```
vestiaire,in the change room,img/hallway.gif
south,aile_droite
NONE
aile_droite,in the right wing,img/outside.gif
west,hall,east,cuisine,south,dock2
NONE
```

3 7.44 beamer

Pour cette fonctionnalité, j'ai d'abord rajouter le beamer dans une salle.

```
cargo1,in the cargo1,img/courtyard.gif
north,cargo2,south,aile_gauche,east,rest
beamer,there is a beamer on the ground,1.0
```

Ensuite, j'ai rajouté dans la class CommandWord les instructions equip, charge, fire et teleport qui correspondent à l'équipement du beamer, recharge du beamer, lancement du beamer et teleportation vers la salle marquée. Voici les nouvelles fonctions du GameModel.

```
    private void equip(Command command){
        if(!command.hasSecondWord()) {
        // if there is no second word, we don't know what to equip...
            gameView.show("equip what ?\n");
            return;
        }
        String itemName = command.getSecondWord();
        ArrayList<Item> roomItem = p1.getCurrentRoom().getItems();
        Boolean found = false;
        for(int i=0;i<roomItem.size();i++){
            if(roomItem.get(i).getName().equals(itemName) &&
                    itemName.equals("beamer")) {
                p1.setBeamer(roomItem.get(i));
                found = true;

                p1.getCurrentRoom().getItems().remove(i);
                p1.getCurrentRoom().setItems(p1.getCurrentRoom().getItems());
            }
        }

        if(found){
            gameView.show("Equip succes \n");
        }else{
            gameView.show("Equipment not found\n");
        }
    }

    private void charge(){
        if(p1.charge()){
            gameView.show("charge succes\n");
        }else{
            gameView.show("charge failed\n");
        }
    }
    private void fire(){
        if(p1.fire()){
            gameView.show("fire succes\n");
            p1.setBeamerLocation(p1.getCurrentRoom());;
        }else{
            gameView.show("fire failed\n");
        }
```

```
}
private void teleport(){
    Room targetRoom = p1.getBeamerRoom();
    if(targetRoom != null){//if valide fired beamer
        goRoom(targetRoom);
    }else{
        gameView.show("Beamer location not found, please fire the beamer\n")
    }
}
```

Par la suite, la classe Player a une nouvelle attribut beamer, qui est une nouvelle class

```
public void setBeamer(Item b){
    this.beamer = new Beamer(b);
}

public Boolean charge(){
    if(beamer.getStatus()){
        beamer.setStatus(false);
        beamer.setCharge(beamer.getCharge() + 1);
        return true;
    }
    return false;
}

public Boolean fire(){
    if(beamer.getCharge() >= 2){
        return true;
    }

    return false;
}

public void setBeamerLocation(Room r){
    beamer.setLocation(r);
    beamer.setCharge(0);//reset charge counter
}

public Boolean teleport(){
    return true;
}

public Room getBeamerRoom(){
    return beamer.getLocation();
}
```

Enfin, voici la class beamer en question, il peut hériter d'une class Item.

```
public class Beamer extends Item {
    Boolean ready;
    Boolean status;
    int charge;
    Room location;

    public Beamer(){
        this.ready = false;
    }

    public Beamer(Item item){
        this.name = item.name;
        this.description = item.description;
        this.weight = item.weight;
        this.charge = 0;
        this.ready = true;
        this.status = false;
        this.location = null;
    }


    public void fire(Room loc){
        this.location = loc;
        this.status = false;
        this.charge = 0;
    }

    public Boolean getStatus(){
        return status;
    }
    public Boolean ready(){
        return ready;
    }
    public int getCharge(){
        return charge;
    }

    public void setStatus(Boolean v){
        status = v;
    }

    public void setCharge(int v){
        charge = v;
    }

    public void setLocation(Room r){
        this.location = r;
    }

    public Room getLocation(){
```

```
        return location;
    }
}
```

Il y a aussi quelque ajout dans GameModel que je n'ai pas cité, ils s'agissent que des ajouts mineur de type copier+coller dans la boucle switch de traitement de commande.

4 7.44 Locked door

Génial, je dois refaire l'exercice 7.43
J'ai donc modifier le fichier RoomData pour qu'en plus des sortie, une valeur booleen soit demandé pour representer le fait que la porte soit vérrouillée ou non. La class Room a désormais une nouvelle attribut door, ainsi qu'une méthode setDoor pour rajouter une porte.

```
public class Room
{
    private String description;
    private String imageLink;
    private HashMap<String, Room> exits;
    private ArrayList<Item> items;
    private ArrayList<Door> doors;
    ...etc...
public void setDoor(String exitName,Boolean v){
    doors.add(new Door(exitName, v));
}

public Door getDoor(String name){
    for(Door door : doors){
        if(door.getExitName().equals(name)){
            return door;
        }
    }
    return null;
}

public boolean isOpen(String name){
    Door d = getDoor(name);
    if(d != null){
        return getDoor(name).getStatus();
    }
    else{
        return true;
    }
}
```

Enfin, dans la méthode goRoom du GameModel, nous devont désormais vérifier si la porte est ouverte ou non.

```
String direction = command.getSecondWord();

// Try to leave current room.
Room nextRoom = p1.getCurrentRoom().getExit(direction);

if (nextRoom == null) {
    gameView.show("There is no door!\n");
}
else if(p1.getCurrentRoom().isOpen(direction)){
    goRoom(nextRoom);
}
else {
    gameView.show("The door is locked!\n");
}
```