RÉALISATION D'APPLICATION (2020)

Itération 3

Groupe 2

03 DÉCEMBRE 2018

1 7.28.1 TEST COMMAND

creer une nouvelle commande test acceptant un second mot representant un nom de fichier, et executant toutes les commandes lues dans ce fichier de texte.

```
public void test_file(Command command)
{
    if(!command.hasSecondWord()) {
        // if there is no second word, we don't know where to go...
        gameView.show("test what ?");
        return;
    }
    String file = command.getSecondWord();
    try {
        Scanner scanner = new Scanner(new File(file));
        while (scanner.hasNextLine()) {
            interpretCommandString(scanner.nextLine());
        }
        scanner.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

ainsi que ajout de Test dans commandWord et dans processCommand

```
private boolean processCommand(Command command)
{

    if(command.isUnknown()) {
        gameView.show("I don't know what you mean...\n");
        return false;
    }
    CommandWord commandWord = command.getCommandWord();
    switch (commandWord){
        case HELP:
            gameView.printHelp();
            break;
        case GO:
            goRoom(command);
            break;
        case BACK:
            goBack(command);
            break;
        case LOOK:
            look();
            break;
        case EAT:
            goRoom(command);
            break;
        case TEST:
            test_file(command);
            break;
        case TAKE:
            take(command);
            break;
        case DROP:
            drop(command);
            break;
        case MINE:
            mine();
            break;
        case QUIT:
            if (confirmQuit(command) == true){
                gameView.disable();
            }
            break;
        default:
            break;

    }
    return true;
}
```

2 7.28.2

Créer 2 fichiers de commandes : 1) parcours idéal pour gagner 2) exploration de toutes les possibilités du jeu
1)ideal_run.txt , le joueur prends la cle du parking et sort du jeu en allant vers les escaliers situé à l'autre bout de la salle

```
look
take key
go east
go up
go east
go east
go south
```

1)explore_all.txt , le joueur explore toutes les salles du jeu, test de back, commandes invalides

```
back
look
take
take key
drop
drop key
go east
go south
back
go north
go north
go south
go east
go north
back
go south
go south
back
back
go east
go south
go south
back
go east
go south
not valid command
foo bar
go go
```

3 7.29 et 7.31

ajout de classe player et ajout d'une array list pour la collecte des items et ajout du champs name dans items ansi que le fichier csv contenant les informations sur les rooms,leur items et leur sorties

```
import java.util.ArrayList;
public class Player {
    private String name;
    private Room currentRoom;
    private  double weight ;
    private ArrayList<Item> items;




    public Player(String name, Room currentRoom) {
        this.name = name;
        this.currentRoom = currentRoom;
        this.weight = 0;
        this.items = new ArrayList<Item>();
    }

    public Room getCurrentRoom() {
        return currentRoom;
    }

    public void setCurrentRoom(Room currentRoom) {
        this.currentRoom = currentRoom;
    }

    public ArrayList<Item> getItems() {
        return items;
    }
    public void setItems(ArrayList<Item> items) {
        this.items = items;
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }
}
```

csv file

```
parking,in the parking,img/castle.gif
east,aile_gauche
pomme,Il y a une pomme par terre, 4.2
aile_gauche,in the left wing,img/courtyard.gif
north,cargo1,south,dock1,up,hall
magic_cookie,increase your weight capacity, 0.0
cargo1,in the cargo1,NULL
north,cargo2,south,aile_gauche,east,rest
NONE
cargo2,in the cargo2,NULL
south,cargo1
pomme,Il y a une pomme par terre,0
dock1,in the dock1,NULL
north,aile_gauche
cle,Il y a une cle par terre, 1
dock2,in the dock2,NULL
north,aile_droite
NONE
```

```
reactor,in the reactor room,NULL
sud,rest
NONE
rest,in the rest room,NULL
north,reactor,west,cargo1,south,hall,east,vestiaire
NONE
vestiaire,in the change room,NULL
west,rest,south,aile_droite
NONE
aile_droite,in the right wing,NULL
north,vestiaire,west,hall,south,cuisine,east,dock2
NONE
cuisine,in the kitchen,NULL
west,aile_droite,south,escalier
NONE
escalier,in the staires,NULL
north,cuisine
NONE
hall,in the Hall,NULL
north,rest,west,aile_gauche,south,commandement,east,aile_droite,
NONE
commandement,in the head quarter,NULL
north,hall
NONE
```

4 7.30/7.31/7.32

ajout des méthodes a exécuter quand on exécute take et drop et gérer le poids max ainsi que le nombre d'item max

```
public void take(Command command)
    {
        if(!command.hasSecondWord()) {
            // if there is no second word, we don't know where to go...
            gameView.show("take what ?\n");
            return;
        }
        String itemName = command.getSecondWord();
        ArrayList<Item> roomItem = p1.getCurrentRoom().getItems();
        for(int i=0;i<roomItem.size();i++){
            if(roomItem.get(i).getName().equals(itemName) && !itemName.equals("magic_cooki
                if(p1.getWeight()>max_weight || p1.getItems().size() > 5) {
                    gameView.show("it's enough for me ?\n");
                    return;
                }

                gameView.show("i took the " + itemName+ "\n");
                ArrayList<Item> newStateOfList  = p1.getItems();
                newStateOfList.add(roomItem.get(i));
                double total_weight = p1.getWeight() + roomItem.get(i).getWeight();
                p1.setWeight(total_weight);
                p1.setItems(newStateOfList);


                ArrayList<Item> roomListItems  = p1.getCurrentRoom().getItems();
                roomListItems.remove(i);
                p1.getCurrentRoom().setItems(roomListItems);



            }
            else
                gameView.show("there is no item who have this name in this room \n");
        }
    }
    public void drop(Command command)
    {
        if(!command.hasSecondWord()) {
            // if there is no second word, we don't know where to go...
            gameView.show("drop what ? \n");
            return;
        }
        String itemName = command.getSecondWord();
        ArrayList<Item> pItem = p1.getItems();
        for(int i=0;i<pItem.size();i++){
            if(pItem.get(i).getName().equals(itemName)) {

                gameView.show("i droped the " + itemName +"\n");

                ArrayList<Item> roomListItems  = p1.getCurrentRoom().getItems();
                roomListItems.add(pItem.get(i));
                p1.getCurrentRoom().setItems(roomListItems);

                double total_weight = p1.getWeight() - pItem.get(i).getWeight();
                p1.setWeight(total_weight);

                ArrayList<Item> newStateOfList  = p1.getItems();
                newStateOfList.remove(i);
                p1.setItems(newStateOfList);


            }
            else
                gameView.show("i already don't have this item \n");
        }
    }
```

pour ajouter les commandes c'est le même procédé que d'habitude

5 7.33 / magic cookie

rajouter la commande eat et faire en sorte qu'elle fonctionne que quand y'a un item magic cookie et aussi qu'elle fanctionne pas avec take comme vu précédemment

```
    private void eat(){
        Item s = new Item("magic_cookie","increase your weight capacity", 0.0);
        for(int i=0;i<p1.getCurrentRoom().getItems().size();i++) {
            if (p1.getCurrentRoom().getItems().get(i).getName().equals("magic_cookie")) {
                gameView.show("magic cookie eaaten \n");
                max_weight += 3.0;
                gameView.show("now your weight capacity is " + Double.toString(max_weight)
            } else {
                gameView.show("noooo magic cookie \n ");
            }
        }
    }
```

6 7.35 enums

ajout de la classe commandWord

```
public enum CommandWord
{
    GO,QUIT,HELP,LOOK,EAT,BACK,TEST,TAKE,DROP,MINE,UNKNOWN;
}
```

dans la classe commandWords

```
    public CommandWords()
    {
        validCommands = new HashMap<String, CommandWord>();
        validCommands.put("go", CommandWord.GO);
        validCommands.put("quit", CommandWord.QUIT);
        validCommands.put("help", CommandWord.HELP);
        validCommands.put("look", CommandWord.LOOK);
        validCommands.put("eat",  CommandWord.EAT);
        validCommands.put("back", CommandWord.BACK);
        validCommands.put("test", CommandWord.TEST);
        validCommands.put("take", CommandWord.TAKE);
        validCommands.put("drop", CommandWord.DROP);
        validCommands.put("mine", CommandWord.MINE);

    }
        public CommandWord getCommandWord(String commandWord)
    {
        CommandWord command = validCommands.get(commandWord);
        if(command != null) {
            return command;
        }
        else {
            return CommandWord.UNKNOWN;
        }
    }

    public boolean isCommand(String aString)
    {
        return validCommands.containsKey(aString);
    }

    /**
     * return a String that containg all valid command word
     * @return the result String
    */
    public String getCommandList(){
        String result = "";
        for (String command : validCommands.keySet()) {
            result += command + " ";
        }
        return result;
    }
```

dans la classe command

```
    public CommandWord getCommandWord()
    {
        return commandWord;
    }
```

dans game model

```
        private boolean processCommand(Command command)
        {

            if(command.isUnknown()) {
                gameView.show("I don't know what you mean...\n");
                return false;
            }
            CommandWord commandWord = command.getCommandWord();
            switch (commandWord){
                case HELP:
                    gameView.printHelp();
                    break;
                case GO:
                    goRoom(command);
                    break;
                case BACK:
                    goBack(command);
                    break;
                case LOOK:
                    look();
                    break;
                case EAT:
                    goRoom(command);
                    break;
                case TEST:
                    test_file(command);
                    break;
                case TAKE:
                    take(command);
                    break;
                case DROP:
                    drop(command);
                    break;
                case MINE:
                    mine();
                    break;
                case QUIT:
                    if (confirmQuit(command) == true){
                        gameView.disable();
                    }
                    break;
                default:
                    break;

            }
            return true;
        }
```