

Réalisation d'application(2020)

iteration 4

Groupe 2

24 AVRIL 2020

1 7.42 TIME OUT

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

```
1     public void timeOut(){
2         if(pastRooms.size()==20) {
3             gameView.show("Time ouuuuuuuutt\n");
4             interpretCommandString("quit");
5         }
6     }
```

2 7.43.45 looked door/trap door

ayant fait un fichier csv pour la description des rooms on a juste rajouté une méthode pour dire c'est quoi l'état de la porte en lisant le fichier csv, état 1 pour trap door ,2 pour looked room 0 si y'a rien

```
1         if (currentRoom.getStateExit(nextRoom) == 0 ||
2             currentRoom.getStateExit(nextRoom) == 1 ) {
3             goRoom(nextRoom);
4             if (beam1()) {
5                 cpt++;
6                 if (cpt == 1) {
7                     checkpoint = nextRoom;
8                     gameView.show("beamer charged you
9                         can use it in the next room");
10                } else if (cpt >= 2) {
11                    gameView.show("beamer can be used\n"
12                        );
13                    used = true;
14                }
15            }
16        }
17        else {
18            if (!key()) {
19                gameView.show("\nloooocked rooom right
20                    here find a key to open it\n");
21            } else {
22                goRoom(nextRoom);
23            }
24        }
25    }
```

```

20         }
21     }

1     if(pastRoom.getStateExit(currentRoom)==0) {
2         goBack(pastRoom);
3         if(beam1()){
4             cpt++;
5             if(cpt==1) {
6                 checkpoint=pastRoom;
7                 gameView.show("beamer charged you
                                can use it in the next room");
8             }else if(cpt>=2){
9                 gameView.show("beamer can be used\n"
                                );
10                used=true;
11            }
12        }
13    }
14    else {
15        if(!key()) {
16            gameView.show("\nyou don't have a key to
                            back to this room use look to find a
                            key \n");
17        }else{
18            gameView.show("\nyou opened the door \n"
                            );
19            goBack(pastRoom);
20        }
21    }

```

3 7.44 beamer

pour cette fonctionnalité on a utiliser a compteur qui reviens a zero quand on utilise le beamer,pour utiliser le beamer faut le prendre comme un item en utilisant look et ensuite quand il est prêt a être utiliser faut taper la commande beam

```

1     private boolean beam1(){
2         for(int i=0;i<p1.getItems().size();i++){
3             if(p1.getItems().get(i).getName().equals("beamer
4                 ")){
5                 return true;
6             }
7         }
8         return false;
9     }
10    case BEAM:
11        if(used) {
12            goBack(checkpoint);
13            gameView.show("beamer used\n");
14            used=false;
15            cpt=0;

```

```

16         }
17         else
18             gameView.show("beamer uncharged\n");
19         break;

```

4 7.46 Transporter room

on a créé deux classe TrasporterRoom et RoomRandomizer ; TrasporterRoom hérite de Room comme stipulé dans le chapitre 9 et utilise RoomRandomizer qui utilise elle même la classe Random .

```

1
2 import java.util.HashMap;
3 import java.util.Random;
4 public class TransporterRoom extends Room
5 {
6     private RoomRandomizer randomRoom;
7     private HashMap <String, Room> myRooms;
8
9     public TransporterRoom (final String description, final
10         HashMap <String, Room> Rooms)
11     {
12         super(description);
13         this.myRooms = Rooms;
14     }
15
16     @Override
17     public Room getExit (final String direction)
18     {
19         randomRoom = new RoomRandomizer(myRooms);
20         return randomRoom.randomizeRooms();
21     }
22 }
23
24
25
26 import java.util.HashMap;
27 import java.util.Random;
28 public class RoomRandomizer
29 {
30     private HashMap <String, Room> myRooms;
31     private Random randomize;
32
33     public RoomRandomizer (final HashMap <String, Room>
34         myRooms)
35     {
36         this.myRooms = myRooms;
37         this.randomize = new Random();
38     }
39
40     public Room randomizeRooms()

```

```

41     {
42         Room[] roomTab = myRooms.values().toArray (new Room
43             [0]);
44         return roomTab[randomize.nextInt (myRooms.size())];
45     }
46 }

```

5 7.47.0 abstract Command

Beaucoup de modifications ont été effectuées pour cette partie, préparez-vous à voir beaucoup de code.

Notre but est de décentraliser la classe `GameModel` qui est surchargée par le traitement des commandes d'utilisateur. On aimerait avoir une classe par commande, tout d'abord la classe `Command` bascule vers une classe abstraite. J'ai enlevé les définitions de getteur et de setteur ainsi que les commentaires pour avoir plus de lisibilité. Ce qui compte ici c'est que nous avons une méthode abstraite "execute" qui lance le traitement de cette commande.

```

1  public abstract class Command
2  {
3      private String secondWord;
4
5      public Command()
6      public String getSecondWord()
7      public boolean hasSecondWord()
8      public void setSecondWord(String secondWord)
9
10     public abstract void execute(Player player,
11         GameModel gameModel, GameView gameView);
12
13 }
14

```

Ensuite passons à la classe `CommandWords`, de même je ne vous montre pas les méthodes `getCommandList` ou `isCommand`, ils ont déjà été présentés dans l'itération précédente.

Nous avons une nouvelle `HashMap` constituée de clé `String` et de valeur `Command`, c'est ici que nous stockerons les classes de traitement de commande. Comme vous pouvez le voir, à chaque mot-clé correspond une classe associée, par exemple, le mot-clé "go" est associé à la classe de commande "GoCommand" et ainsi de suite.

Enfin la méthode `get` retourne une classe héritée de la classe abstraite `Command`, en fonction du mot-clé donné en paramètre.

Listing 1: `CommandWords`

```

1  public class CommandWords
2  {
3
4      private HashMap<String, CommandWord> validCommands;
5      private HashMap<String, Command> commands;
6

```

```

7     public CommandWords()
8     {
9         validCommands = new HashMap<String, CommandWord>();
10        for (CommandWord command : CommandWord.values()) {
11            if (command != CommandWord.UNKNOWN){
12                validCommands.put(command.toString(),
13                                command);
14            }
15        }
16
17        commands = new HashMap<String, Command>();
18        commands.put("go", new GoCommand());
19        commands.put("quit", new QuitCommand());
20        commands.put("sos", new HelpCommand());
21        commands.put("look", new LookCommand());
22        commands.put("eat", new EatCommand());
23        commands.put("back", new BackCommand());
24        commands.put("test", new TestCommand());
25        commands.put("take", new TakeCommand());
26        commands.put("drop", new DropCommand());
27        commands.put("mine", new MineCommand());
28        commands.put("beam", new BeamCommand());
29    }
30
31    public boolean isCommand(String aString)
32
33    public String getCommandList()
34
35    public Command get(String word)
36    {
37        return (Command)commands.get(word);
38    }
39 }

```

Puis, dans la classe Parser, une légère modification est nécessaire car la classe Command est devenue abstraite, nous avons besoin d'utiliser la méthode get pour avoir la bonne classe en retour de fonction.

Listing 2: Parser

```

1     public Command getCommand(String inputLine)
2     {
3         String word1;
4         String word2;
5
6         StringTokenizer tokenizer = new StringTokenizer(
7             inputLine);
8         if (tokenizer.hasMoreTokens())
9             word1 = tokenizer.nextToken();
10        else
11            word1 = null;
12        if (tokenizer.hasMoreTokens())
13            word2 = tokenizer.nextToken();
14        else

```

```

14         word2 = null;
15         Command command = commands.get(word1);
16         if(command != null) {
17             command.setSecondWord(word2);
18         }
19         return command;
20     }

```

Beaucoup de fonctions ont disparue, ils ont été transféré à leur classe associé, `interpretCommandString` n'a pas changé, en fonction du `String` donné en argument, il va demander au parser de lui retourner la bonne classe associé. Enfin `processCommand` va vérifier la validité de la commande, si elle n'est pas null, elle lance sa méthode "execute".

Certains attributs implémenté par ma camarade ont été transféré dans la classe `player` (`cpt`, `used`, `checkpoint`) par exemple.

Listing 3: GameModel

```

1  public class GameModel extends Observable
2  {
3      private Player p1;
4      private Parser parser;
5      private HashMap<String,Room> rooms;
6      private Stack<Room> pastRooms;
7      private GameView gameView;
8      private TransporterRoom tr ;
9
10     public GameModel()
11     public Player getP1()
12     public int getPastRoomsSize()
13     public void addGameView(GameView gm)
14     public Room getCurrentRoom()
15     private void createRooms()
16     public void notifyChange()
17     public String getImageLinkString()
18     public String getWelcomeString()
19     public void timeOut(){
20     public void addPastRoom(Room r){
21     public Stack<Room> getPastRooms()
22     public String getGoodByeString()
23     public String getHelpString()
24     public String getExitString()
25     public String getLocationInfo()
26     public String getCommandString()
27
28     public void interpretCommandString(String userInput)
29     {
30         Command command = parser.getCommand(userInput);
31         processCommand(command);
32     }
33
34     public void play() {
35         gameView.printWelcome();
36     }

```

```

37
38     private boolean processCommand(Command command)
39     {
40
41         if(command == null){
42             gameView.show("I don't know what you
43                             mean...\n");
44             return false;
45         }else{
46             command.execute(p1, this ,gameView);
47         }
48         return true;
49     }
50
51 }

```

En ce qui concerne la classe player, vous avez le transfert des attributs cpt, checkpoint, used et max_weight depuis le GameModel. Les nouvelles méthodes apparues étant juste des getter et des setter, je vous mets pas le code, leur déclaration est suffisamment explicite.

Listing 4: Player

```

1     import java.util.ArrayList;
2     public class Player {
3         private String name;
4         private Room currentRoom;
5         private double weight ;
6         private ArrayList<Item> items;
7         private int cpt;
8         private Room checkpoint;
9         private boolean used=false;
10        private double max_weight = 6.0;
11
12        public Player(String name, Room currentRoom)
13        public double getMaxWeight()
14        public void setMaxWeight(double w)
15        public void setUsed(boolean v)
16        public boolean getUsed()
17        public void setCheckpoint(Room r)
18        public Room getCheckpoint()
19        public void incCpt()
20        public void setCpt(int v)
21        public int getCpt()
22        public String getName()
23        public Room getCurrentRoom()
24        public void setCurrentRoom(Room currentRoom)
25        public ArrayList<Item> getItems()
26        public void setItems(ArrayList<Item> items)
27        public double getWeight()
28        public void setWeight(double weight)
29        public boolean key(){
30            for(int i=0;i<getItems().size();i++){
31                if(getItems().get(i).getName().equals("key"))

```

```

        ){
            return true;
        }
    }
    return false;
}

public boolean beam1(){
    for(int i=0;i<getItems().size();i++){
        if(getItems().get(i).getName().equals("
            beamer")){
                return true;
            }
        }
    }
    return false;
}
}

```

Enfin passons au commandes, voici donc la liste des classes de command. J'ai enlever les commentaires car cela gêne la lecture, ces classe n'ont pas subit de grande transformation, ils reste dans l'esprit des fonctions que je vous ai présenté dans les iterations précédente. Néanmoins, si vous désirez voir ces détails, voici le lien.

<https://github.com/bk211/zuul-bad-ultimate-super-champion-directorcut-edition>

Listing 5: Liste de commande

```

1
2 BackCommand.java
3 BeamCommand.java
4 DropCommand.java
5 EatCommand.java
6 GoCommand.java
7 HelpCommand.java
8 LookCommand.java
9 MineCommand.java
10 QuitCommand.java
11 TakeCommand.java
12 TestCommand.java

```

Listing 6: Backcommand

```

1 public class BackCommand extends Command
2 {
3
4     public BackCommand()
5
6     public void goBack(Room lastRoom, GameModel gm, Player
        player, GameView gameView)
7     {
8
9         player.setCurrentRoom(lastRoom);
10        if(player.getCurrentRoom().getImageLinkString() !=
            null){

```



```

11         gameView.showImage(player.getCurrentRoom().
12             getImageLinkString());
13     }
14     gm.notifyChange();
15 }
16
17 public void execute(Player player, GameModel gameModel,
18     GameView gameView){
19     if(hasSecondWord()) {
20         gameView.show("Back what?\n");
21     }
22     else if(gameModel.getPastRooms().empty()){
23         // si la pile est vide
24         gameView.show("No record of last visited room");
25     }else{
26         // la commande est valide
27         // Try to leave current room.
28
29         Room pastRoom = gameModel.getPastRooms().pop();
30         Room currentRoom = player.getCurrentRoom();
31         if(pastRoom.getStateExit(currentRoom)==0) {
32             goBack(pastRoom, gameModel, player, gameView
33                 );
34             if (player.beam1()) {
35                 player.incCpt();
36                 if (player.getCpt() == 1) {
37                     player.setCheckpoint(pastRoom);
38                     gameView.show("beamer charged you
39                         can use it in the next room");
40                 } else if (player.getCpt() >= 2) {
41                     gameView.show("beamer can be used\n"
42                         );
43                     player.setUsed(true);
44                 }
45             }
46         }
47     }
48     else {
49         if(!player.key()) {
50             gameView.show("\nyou don't have a key to
51                 back to this room use look to find a
52                 key \n");
53         }else{
54             gameView.show("\nyou opened the door \n"
55                 );
56
57             goBack(pastRoom, gameModel, player,
58                 gameView);
59         }
60     }
61 }
62
63 }
64
65 }

```

Listing 7: Beamcommand

```

1  public class BeamCommand extends Command
2  {
3      public BeamCommand()
4      {
5      }
6
7      public void goBack(Room lastRoom, GameModel gm,
8          Player player, GameView gameView)
9      {
10         player.setCurrentRoom(lastRoom);
11         if(player.getCurrentRoom().getImageLinkString()
12             != null){
13             gameView.showImage(player.getCurrentRoom().
14                 getImageLinkString());
15         }
16         gm.notifyChange();
17     }
18
19     public void execute(Player player, GameModel
20         gameModel, GameView gameView){
21
22         if(player.getUsed()){
23             goBack(player.getCheckpoint(), gameModel,
24                 player, gameView);
25             gameView.show("beamer used\n");
26             player.setUsed(false);
27             player.setCpt(0);
28         }
29         else
30             gameView.show("beamer uncharged\n");
31     }
32 }

```

Listing 8: Dropcommand

```

1  import java.util.ArrayList;
2  public class DropCommand extends Command
3  {
4      public DropCommand()
5      {
6      }
7
8      public void execute(Player player, GameModel
9          gameModel, GameView gameView){
10         if(!hasSecondWord()) {
11             // if there is no second word, we don't know
12             // where to go...
13             gameView.show("drop what ? \n");
14             return;
15         }
16         String itemName = getSecondWord();
17         ArrayList<Item> pItem = player.getItems();

```

```

16         for(int i=0;i<pItem.size();i++){
17             if(pItem.get(i).getName().equals(itemName))
18                 {
19                     gameView.show("i dropped the " + itemName
20                                     + "\n");
21
22                     ArrayList<Item> roomListItems = player.
23                         getCurrentRoom().getItems();
24                     roomListItems.add(pItem.get(i));
25                     player.getCurrentRoom().setItems(
26                         roomListItems);
27
28                     double total_weight = player.getWeight()
29                         - pItem.get(i).getWeight();
30                     player.setWeight(total_weight);
31
32                     ArrayList<Item> newStateOfList = player
33                         .getItems();
34                     newStateOfList.remove(i);
35                     player.setItems(newStateOfList);
36                 }
37             else
38                 gameView.show("i already don't have this
39                               item \n");
40         }
41     }
42 }

```

Listing 9: Eatcommand

```

1 public class EatCommand extends Command
2 {
3
4     public EatCommand()
5     {
6     }
7
8     public void execute(Player player, GameModel
9         gameModel,GameView gameView){
10         for(int i=0;i<player.getCurrentRoom().getItems().
11             size();i++) {
12             if (player.getCurrentRoom().getItems().get(i)
13                 .getName().equals("magic_cookie")) {
14                 gameView.show("magic cookie eaaten \n");
15                 player.setMaxWeight(player.getMaxWeight()
16                     + 3);
17                 gameView.show("now your weight capacity
18                             is " + Double.toString(player.
19                                 getMaxWeight()) + "\n");
20             } else {
21                 gameView.show("noooo magic cookie \n ");
22             }
23         }
24     }
25 }

```

```

17     }
18 }
19 }

```

Listing 10: Gocommand

```

1  public class GoCommand extends Command
2  {
3      public GoCommand()
4      {
5      }
6
7      public void goRoom(Room nextRoom, GameModel gm, Player
           player, GameView gameView)
8      {
9          gm.addPastRoom(player.getCurrentRoom());
10         player.setCurrentRoom(nextRoom);
11         if(player.getCurrentRoom().getImageLinkString() !=
           null){
12             gameView.showImage(player.getCurrentRoom().
                 getImageLinkString());
13         }
14         gm.notifyChange();
15     }
16
17     public void execute(Player player, GameModel gameModel,
           GameView gameView){
18
19         if(!hasSecondWord()) {
20             // if there is no second word, we don't know
                where to go...
21             gameView.show("Go where?");
22             return;
23         }
24
25         String direction = getSecondWord();
26
27         // Try to leave current room.
28         Room currentRoom = player.getCurrentRoom();
29         Room nextRoom = player.getCurrentRoom().getExit(
           direction);
30
31         if (nextRoom == null) {
32             gameView.show("There is no door!\n");
33         }
34         else {
35             if (currentRoom.getStateExit(nextRoom) == 0 ||
                 currentRoom.getStateExit(nextRoom) == 1 ) {
36                 goRoom(nextRoom, gameModel, player,
                     gameView);
37                 if (player.beam1()) {
38                     player.incCpt();
39                     if (player.getCpt() == 1) {
40                         player.setCheckpoint(nextRoom);

```

```

41         gameView.show("beamer charged you
42             can use it in the next room");
43     } else if (player.getCpt() >= 2) {
44         gameView.show("beamer can be used\n"
45             );
46         player.setUsed(true);
47     }
48 }
49 else {
50     if(!player.key()) {
51         gameView.show("\nloocked rooom right
52             here find a key to open it\n");
53     }else{
54         goRoom(nextRoom, gameModel, player,
55             gameView);
56     }
57 }
58 }
59 }
60 }
61 }
62 }

```

Listing 11: Helpcommand

```

1  public class HelpCommand extends Command
2  {
3      public HelpCommand()
4      {
5      }
6
7      public void execute(Player player, GameModel gameModel,
8          GameView gameView){
9          gameView.printHelp();
10     }
11 }
12 }

```

Listing 12: Lookcommand

```

1  public class LookCommand extends Command
2  {
3      public LookCommand()
4      {
5      }
6
7      public void execute(Player player, GameModel gameModel,
8          GameView gameView){
9          gameView.show(player.getCurrentRoom().
10             getLongDescription());

```

```

9     }
10 }

```

Listing 13: Minecommand

```

1  public class MineCommand extends Command
2  {
3
4      public MineCommand()
5      {
6      }
7
8      public void execute(Player player, GameModel
          gameModel, GameView gameView){
9
10         if(player.getItems().size()==0)
11             gameView.show("i am poor i only have \n");
12         for(Item i:player.getItems())
13             gameView.show(i.getName() + "\n");
14         gameView.show(Double.toString(player.getWeight()
15             ) + "\n");
16     }
17 }

```

Listing 14: Quitcommand

```

1  public class QuitCommand extends Command
2  {
3
4      public QuitCommand()
5      {
6      }
7
8      public void execute(Player player, GameModel gameModel,
          GameView gameView){
9          if(hasSecondWord()) {
10             gameView.show("Quit what?\n");
11         }
12         else {
13             gameView.disable();
14         }
15     }
16
17 }
18 }

```

Listing 15: Takecommand

```

1  public class TakeCommand extends Command
2  {
3
4      public TakeCommand()
5      {
6      }

```

```

7
8     public void execute(Player player, GameModel
9         gameModel, GameView gameView){
10         if(!hasSecondWord()) {
11             // if there is no second word, we don't know
12             // where to go...
13             gameView.show("take what ?\n");
14             return;
15         }
16         String itemName = getSecondWord();
17         ArrayList<Item> roomItem = player.getCurrentRoom
18             ().getItems();
19         for(int i=0;i<roomItem.size();i++){
20             if(roomItem.get(i).getName().equals(itemName)
21                 && !itemName.equals("magic_cookie")) {
22                 if(player.getWeight()>player.
23                     getMaxWeight() || player.getItems().
24                     size() > 5) {
25                     gameView.show("it's enough for me ?\n");
26                     return;
27                 }
28             }
29
30             gameView.show("i took the " + itemName+
31                 "\n");
32
33             ArrayList<Item> newStateOfList = player
34                 .getItems();
35             newStateOfList.add(roomItem.get(i));
36             double total_weight = player.getWeight()
37                 + roomItem.get(i).getWeight();
38             player.setWeight(total_weight);
39             player.setItems(newStateOfList);
40
41             ArrayList<Item> roomListItems = player.
42                 getCurrentRoom().getItems();
43             roomListItems.remove(i);
44             player.getCurrentRoom().setItems(
45                 roomListItems);
46
47         }
48     }
49 }
50

```

Listing 16: Testcommand

```

1     public class TestCommand extends Command
2     {
3         public TestCommand()
4         {
5

```

```

6
7      public void execute(Player player, GameModel
8          gameModel, GameView gameView){
9
10         if(!hasSecondWord()) {
11             // if there is no second word, we don't know
12             // where to go...
13             gameView.show("test what ?");
14             return;
15         }
16         String file = getSecondWord();
17         try {
18             Scanner scanner = new Scanner(new File(file)
19                 );
20             while (scanner.hasNextLine()) {
21                 gameModel.interpretCommandString(scanner
22                     .nextLine());
23             }
24             scanner.close();
25         } catch (FileNotFoundException e) {
26             e.printStackTrace();
27         }
28     }
29 }

```

6 7.47.1

J'ai crée 4 paquet pour le mise en paquet, la première est le package pkg_commands qui regroupe les différentes commandes de jeu utilisé par le joueur comme "go", "back", "eat".

Listing 17: pkg_commands

```

1      BackCommand.java
2      EatCommand.java
3      LookCommand.java
4      TakeCommand.java
5      BeamCommand.java
6      GoCommand.java
7      MineCommand.java
8      TestCommand.java
9      DropCommand.java
10     HelpCommand.java
11     QuitCommand.java

```

Le paquet pkg_data stocke toutes les classes support qui permet de représenter une information de jeu, il y a donc la class Item, Player ainsi que les 3 classes en rapport avec Room.

Listing 18: pkg_data

```

1      Item.java
2      Player.java

```



```

3      Room.java
4      RoomRandomizer.java
5      TransporterRoom.java

```

Puis arrive le paquet `pkg_tools` qui contient les classes "outils", ils sont utilisés par les classe supérieur pour traiter ou stocker certains donnée, Par exemple le parser pour la conversion instruction String vers instruction Command à proprement parler, la classe `RoomFileReader` qui permet de lire le fichier `RoomData.csv`, qui stocke la configuration des salles de jeu.

Listing 19: `pkg_tools`

```

1      Command.java
2      CommandWord.java
3      CommandWords.java
4      Parser.java
5      RoomFileReader.java

```

Enfin le paquet `mainStruct` est composé des 2 grande composante du modele MVC ainsi que la classe `UserInterface`. Ce sont des classes d'entrée au niveau supérieur pour le programme.

Listing 20: `pkg_commands`

```

1      GameModel.java
2      GameView.java
3      UserInterface.java

```

7 7.53 7.54

BlueJ a été abandonné dès le début du projet car il n'apportait aucun bénéfice sur le developement du projet, car l'interface BlueJ était lent au niveau de la compilation et pénible à utiliser car il ne présente pas de foncitionnalité interessante si ce n'est pédagogique pour un débutant en Java , pour exécuter le programme par exemple, il fallait charger blueJ, lancer une nouvelle instance de Game puis lancer la méthode `play()`. Dès le début, un fichier `Makefile` était présent pour gérer d'éventuel dépendance. Je vous mets quand même le code de la classe `Game` ici.

Listing 21: `Game`

```

1      public class Game
2      {
3          private GameModel gameModel;
4          private GameView gameView;
5
6          public Game()
7          {
8              gameModel = new GameModel();
9
10             gameView = new GameView(gameModel);
11             gameModel.addObserver(gameView);
12             gameModel.addGameView(gameView);
13
14         }

```

```

15
16     public void play()
17     {
18         gameModel.play();
19     }
20
21     public static void main(String[] args) {
22         Game g = new Game();
23         g.play();
24     }
25
26 }

```

8 58

La commande utilisé est la suivante

Listing 22: Make jar

```

1 jar cvmf MANIFEST.MF Game.jar Game.class pkg_commands/
  BackCommand.class pkg_commands/BeamCommand.class
  pkg_commands/DropCommand.class pkg_commands/EatCommand.
  class pkg_commands/GoCommand.class pkg_commands/
  HelpCommand.class pkg_commands/LookCommand.class
  pkg_commands/MineCommand.class pkg_commands/QuitCommand.
  class pkg_commands/TakeCommand.class pkg_commands/
  TestCommand.class pkg_data/Item.class pkg_data/Player.
  class pkg_data/Room.class pkg_mainStruct/GameModel.class
  pkg_mainStruct/GameView.class pkg_mainStruct/
  UserInterface\$.class pkg_mainStruct/UserInterface.class
  pkg_tools/Command.class pkg_tools/CommandWord.class
  pkg_tools/CommandWords.class pkg_tools/Parser.class
  pkg_tools/RoomFileReader.class img/* cmd.txt explore_all.
  txt RoomData.csv ideal_run.txt

```

9 59

Concernant la sauvegarde, le systeme est similaire à celui de la commande test. L'idée est donc de sauvegarder toutes les entrées utilisateur, dans notre programme, l'entrée utilisateur est saisi dans l'interface graphic, qui est ensuite envoyé dans le GameModel sous forme de String, Donc au niveau du GameModel, j'ai rajouté une ArrayList qui va stocker ces commandes.

A chaque fois qu'une commande est saisi, une copie est ajouté dans ce tableau de stockage

Listing 23: GameModel

```

1 public class GameModel extends Observable
2 {
3     private ArrayList<String> userMove;
4
5     public GameModel()

```

```

6      {
7          pastRooms = new Stack<Room>();
8          createRooms();
9          this.parser = new Parser();
10         userMove = new ArrayList<String>();
11     }
12
13     public void interpretCommandString(String userInput){
14         Command command = parser.getCommand(userInput);
15         saveCommandString(userInput);
16         processCommand(command);
17     }
18
19
20     private void saveCommandString(String userInput){
21         userMove.add(userInput);
22     }
23     public ArrayList<String> getUserHistory(){
24         return userMove;
25     }

```

Ensuite, j'ai créé une nouvelle commande "save" qui fait une sauvegarde l'état du jeu, il suffit donc d'ajouter cette ligne dans CommandWords, et CommandWord

Listing 24: CommandWords Et CommandWord

```

1     SAVE("save");
2     commands.put("save", new SaveCommand());

```

Nous avons donc une nouvelle classe de commande que j'ai appelé SaveCommand.java, il va vérifier si l'utilisateur a donné un nom de fichier, si oui, il effectue la sauvegarde des commandes dans ce fichier.

Listing 25: SaveCommand

```

1     package pkg_commands;
2     import pkg_mainStruct.GameModel;
3     import pkg_mainStruct.GameView;
4
5     import java.io.FileWriter;
6     import java.io.IOException;
7     import java.util.ArrayList;
8
9     import pkg_data.Player;
10    import pkg_tools.Command;
11
12    public class SaveCommand extends Command
13    {
14        public SaveCommand(){
15
16
17            public void execute(Player player, GameModel
18                gameModel,GameView gameView){
19
20                if(!hasSecondWord()) {
21                    // if there is no second word, we don't know
22                    where to go...

```

```

20         gameView.show("save what ?");
21         return;
22     }
23     String filename = getSecondWord();
24     ArrayList<String> userHistory = gameModel.
        getUserHistory();
25     userHistory.remove(userHistory.size() -1); //
        remove last instruction since it's the save
        instruction
26
27     try {
28         FileWriter writer = new FileWriter(filename
        );
29         for (String str : userHistory) {
30             writer.write(str+"\n");
31         }
32         writer.close();
33
34     } catch (IOException e) {
35         gameView.show("Error with the given file\n")
        ;
36     }
37
38 }
39
40 }

```