Réalisation d'application(2020) iteration 4

Groupe 2

24 AVRIL 2020

1 7.42 TIME OUT

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

```
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||
                   currentRoom.getStateExit(nextRoom) == 1 ) {
2                  goRoom(nextRoom);
3                  if (beam1()) {
4                      cpt++;
5                      if (cpt == 1) {
6                          checkpoint = nextRoom;
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             }
15             else {
16                 if(!key()) {
17                     gameView.show("\nlooocked rooom right
                           here find a key to open it\n");
18                 }else{
19                     goRoom(nextRoom);
```

```
                }
            }

            if(pastRoom.getStateExit(currentRoom)==0) {
                goBack(pastRoom);
                if(beam1()){
                    cpt++;
                    if(cpt==1) {
                        checkpoint=pastRoom;
                        gameView.show("beamer charged you
                            can use it in the next room");
                    }else if(cpt>=2){
                        gameView.show("beamer can be used\n"
                            );
                        used=true;
                    }
                }
            }
            else {
                if(!key()) {
                    gameView.show("\nyou don't have a key to
                         back to this room use look to find a
                         key \n");
                }else{
                    gameView.show("\nyou opened the door \n"
                        );
                    goBack(pastRoom);
                }
            }
```

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

```
    private boolean beam1(){
        for(int i=0;i<p1.getItems().size();i++){
            if(p1.getItems().get(i).getName().equals("beamer
                ")){
                return true;
            }
        }
        return false;
    }

    case BEAM:
                    if(used) {
                        goBack(checkpoint);
                        gameView.show("beamer used\n");
                        used=false;
                        cpt=0;
```

```
                    }
                    else
                        gameView.show("beamer uncharged\n");
                    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 .

```

import java.util.HashMap;
import java.util.Random;
public class TransporterRoom extends Room
{
    private  RoomRandomizer randomRoom;
    private HashMap <String, Room> myRooms;

    public TransporterRoom (final String description, final
        HashMap <String, Room> Rooms)
    {
        super(description);
        this.myRooms = Rooms;

    }

    @Override
    public Room getExit (final String direction)
    {
        randomRoom = new RoomRandomizer(myRooms);
        return randomRoom.randomizeRooms();

    }
}



import java.util.HashMap;
import java.util.Random;
public class RoomRandomizer
{
    private HashMap <String, Room> myRooms;
    private Random randomize;

    public RoomRandomizer (final HashMap <String, Room>
        myRooms)
    {
        this.myRooms = myRooms;
        this.randomize = new Random();
    }

    public Room randomizeRooms()
```

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

5 7.47.0 abstract Command

Beaucoup de modifications ont été effectué pour cette partie, préparer vous à voir beaucoup de code.
Notre but est de décentraliser la classe GameModel qui est surchargé par le traitement des commandes d'utilisateur, On aimerais avoir une classe par commande, tout d'abord la class Command bascule vers une class abtraite. J'ai enlevé les definitions 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
2   public abstract class Command
3   {
4       private String secondWord;
5
6       public Command()
7       public String getSecondWord()
8       public boolean hasSecondWord()
9       public void setSecondWord(String secondWord)
10
11      public abstract void execute(Player player,
12          GameModel gameModel,GameView gameView);
13
14  }
```

Ensuite passons à la classe CommandeWords, de meme je ne vous montre pas les méthodes getCommandList ou isCommand, ils ont déja été présenté dans l'iteration prédecedente.
Nous avons une nouvelle hashMap constitué 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 mots clé correspond une classe associé, 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 hérité de la classe abstraite Command, en fonction du mot clé donné en paramètre.

Listing 1: CommandeWords

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

Puis, dans la classe Parser, une légère modification est necessaire 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(
                inputLine);
7           if(tokenizer.hasMoreTokens())
8               word1 = tokenizer.nextToken();
9           else
10              word1 = null;
11          if(tokenizer.hasMoreTokens())
12              word2 = tokenizer.nextToken();
13          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              Command command = parser.getCommand(userInput);
30              processCommand(command);
31
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
                            mean...\n");
43                      return false;
44                  }else{
45                      command.execute(p1, this ,gameView);
46                  }
47
48                  return true;
49          }
50
51      }
```

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

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")
```

```
                    ){
32                      return true;
33                  }
34              }
35              return false;
36          }
37
38          public boolean beam1(){
39              for(int i=0;i<getItems().size();i++){
40                  if(getItems().get(i).getName().equals("
                        beamer")){
41                      return true;
42                  }
43              }
44              return false;
45          }
46
47      }
```

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

Listing 7: Beamcommand

```
1  public class BeamCommand extends Command
2  {
3      public BeamCommand()
4      {
5      }
6
7      public void goBack(Room lastRoom, GameModel gm,
           Player player, GameView gameView)
8      {
9
10         player.setCurrentRoom(lastRoom);
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(player.getUsed()){
20             goBack(player.getCheckpoint(), gameModel,
                   player, gameView);
21             gameView.show("beamer used\n");
22             player.setUsed(false);
23             player.setCpt(0);
24         }
25         else
26             gameView.show("beamer uncharged\n");
27     }
28 }
```

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
            gameModel,GameView gameView){
9           if(!hasSecondWord()) {
10              // if there is no second word, we don't know
                    where to go...
11              gameView.show("drop what ? \n");
12              return;
13          }
14          String itemName = getSecondWord();
15          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 droped the " + itemName
                            +"\n");
20
21                      ArrayList<Item> roomListItems  = player.
                            getCurrentRoom().getItems();
22                      roomListItems.add(pItem.get(i));
23                      player.getCurrentRoom().setItems(
                            roomListItems);
24
25                      double total_weight = player.getWeight()
                             - pItem.get(i).getWeight();
26                      player.setWeight(total_weight);
27
28                      ArrayList<Item> newStateOfList  = player
                            .getItems();
29                      newStateOfList.remove(i);
30                      player.setItems(newStateOfList);
31
32
33                  }
34                  else
35                      gameView.show("i already don't have this
                            item \n");
36              }
37          }
38      }
```

Listing 9: Eatcommand

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

```
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
                                can use it in the next room");
42                      } else if (player.getCpt() >= 2) {
43                          gameView.show("beamer can be used\n"
                                );
44                          player.setUsed(true);
45                      }
46                  }
47
48              }
49              else {
50                  if(!player.key()) {
51                      gameView.show("\nlooocked rooom right
                            here find a key to open it\n");
52                  }else{
53                      goRoom(nextRoom,  gameModel, player,
                            gameView);
54                  }
55              }
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,
            GameView gameView){
8           gameView.printHelp();
9       }
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,
            GameView gameView){
8           gameView.show(player.getCurrentRoom().
                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()
                    ) + "\n");
15
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
                gameModel,GameView gameView){
9               if(!hasSecondWord()) {
10                  // if there is no second word, we don't know
                        where to go...
11                  gameView.show("take what ?\n");
12                  return;
13              }
14              String itemName = getSecondWord();
15              ArrayList<Item> roomItem = player.getCurrentRoom
                    ().getItems();
16              for(int i=0;i<roomItem.size();i++){
17                  if(roomItem.get(i).getName().equals(itemName
                        ) && !itemName.equals("magic_cookie")) {
18                      if(player.getWeight()>player.
                            getMaxWeight() || player.getItems().
                            size() > 5) {
19                          gameView.show("it's enough for me ?\
                                n");
20                          return;
21                      }
22
23                      gameView.show("i took the " + itemName+
                            "\n");
24
25                      ArrayList<Item> newStateOfList  = player
                            .getItems();
26                      newStateOfList.add(roomItem.get(i));
27                      double total_weight = player.getWeight()
                             + roomItem.get(i).getWeight();
28                      player.setWeight(total_weight);
29                      player.setItems(newStateOfList);
30
31                      ArrayList<Item> roomListItems  = player.
                            getCurrentRoom().getItems();
32                      roomListItems.remove(i);
33                      player.getCurrentRoom().setItems(
                            roomListItems);
34
35                  }
36                  else
37                      gameView.show("there is no item who have
                            this name in this room \n");
38              }
39          }
40      }
```

Listing 16: Testcommand

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

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

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 reprensenter 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 bénefice 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 foncitonalité interessente 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: pkg_commands

```
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      }
```