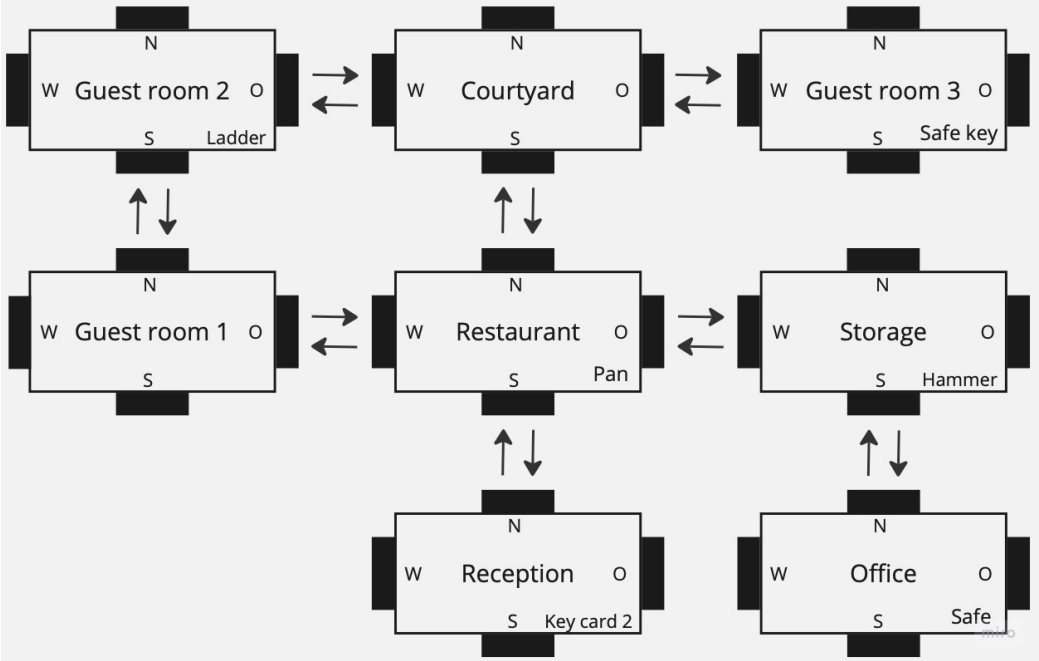
Abandoned Hotel Dokumentation

Spielidee

- In diesem Spiel geht es um ein verlassenes Hotel in dem ein Geldschatz vermutet wird.
- Dieser befindet sich im Safe des Büros des Hotels.
- Das verlassene Hotel hat verschiedene Räume. Einige dieser Räume sind verschlossen und müssen mit entsprechenden Gegenständen geöffnet werden (z.B. Schlüsselkarte, Hammer)
- Der Spieler startet in der Rezeption und muss zuerst im Hotel die Zahlenkombination des Safes finden und danach mit dieser zum Büro gehen.
- Der Spieler gewinnt, wenn er in das Büro kommt und die Zahlenkombination dabei hat.

Spielplan

Das Hotel hat 8 Räume die gemäss dem Plan unten mit Türen verbunden sind. In einigen Räumen gibt es Gegenstände die man mitnehmen kann. Diese sind rechts unten bei jedem Raum angegeben.



2.2 Wann hat man gewonnen?

Der Spieler hat gewonnen wenn er in die Office kommt und eine Leiter und die Zahlenkombination dabei hat.

Jeder Raum hat ein boolean Attribut *target* welches angibt ob es sich bei dem Raum um einen Gewinnerraum handelt. Bei meinem Spiel muss man allerdings auch noch überprüfen ob der Spieler beim Betreten des Raums die Leiter und die Zahlenkombination dabei hat. Dies wird in der Methode *goRoom(Command command)* der Klasse *Game* folgendermassen überprüft;

```
if (currentRoom().isTarget() &&
    takenItems.contains(ladder) &&
    takenItems.contains(combination)) {
      System.out.println("You have reached " + currentRoom().shortDescription() +
                         " with a ladder and the key combination.");
      System.out.println("You won the game.");
      return true;
}
```

Meine Methode *goRoom* gibt als Rückgabewert an, ob das Spiel gewonnen wurde. Das Attribut *takenItems* ist eine Liste aller Gegenstände (siehe unter 2.4) die der Benutzer genommen hat.

2.3 Befehl “back”

Um den Befehl “back” implementieren zu können, muss man sich statt des aktuellen Raums (*currentRoom*) in der Klasse Game den gesamten Pfad des Benutzers merken. Dies mache ich in dem Attribut *path* des Typs *Stack<Room>.* Durch die Verwendung eines Stapels kann ich mit *push(room)* das nächste erreichte Zimmer in der Methode *goRoom(…)* hinzufügen. Mit *pop()* kann ich ein Zimmer zurückgehen. Dies habe ich mit der Methode *goBack()* in der Klasse *Game* implementiert. Zuvor habe ich in der Klasse *CommandWords* noch das “back” Kommandowort hinzugefügt.

```
private void goBack() {
  if (path.size() == 1) {
    System.out.println("You cannot go back.");
  } else {
    path.pop();
    printDescription();
  }
}
```

2.4+2.5 Mehrere Gegenstände im Raum

Meine Klasse Item sieht folgendermassen aus:

```
public class Item {
  private String name;
  private String description;

  public Item(String name, String description) {
    this.name = name;
    this.description = description;
  }
  …
}
```

Meine Gegenstände haben einen Namen *name* und eine Beschreibung *description.*
Zur Klasse *Room* habe ich ein Attribute *items* hinzugefügt in welchem alle Gegenstände die sich in diesem Raum befinden abgespeichert werden. Mit den Methoden *addItem*, *removeItem*, und *containsItem* kann ein Gegenstand hinzugefügt, weggenommen, und überprüft werden. Ich habe auch noch eine neue Methode *descriptionWithItems()* geschrieben über welche ausgedruckt werden kann, welche Gegenstände im Raum sind.

```
public class Room {
  private String description;
  private HashMap<String, Room> exits;
  private boolean target;
  private List<Item> items;

  public Room(String description, boolean target) {
    this.description = description;
    this.exits = new HashMap<>();
    this.target = target;
    this.items = new ArrayList<>();
  }
  …
  public boolean containsItem(Item item) {
    return items.contains(item);
  }
  public void addItem(Item item) {
    items.add(item);


  public void removeItem(Item item) {
    items.remove(item);
  }
  …
  public String descriptionWithItems() {
    StringBuilder stringBuilder = new StringBuilder(description);
    if (items.size() > 0) {
      stringBuilder.append(": “);
      stringBuilder.append(itemString());
    }
    return stringBuilder.toString();
  }
  private String itemString() {
    List<String> descriptions = items.stream().map(i -> i.toString()).collect(Collectors.toList());
    return String.join(", ", descriptions);
  }
  private String exitString() {
    return String.join(", ", exits.keySet());
  }
}
```

2.6 Gewicht

Ich habe zur Klasse *Item* ein Attribut *weight* hinzugefügt welches für jeden Gegenstand das Gewicht (in Gramm) angibt. Über einen Getter *getWeight()* kann das Gewicht eines Gegenstands abgefragt werden.

```
public class Item {
  ...
  private double weight;
  ...
```

```
    public Item(…, double weight) {
      …
      this.weight = weight;
    }
    public double getWeight() {
      return weight;
    }
}
```

2.7 Map

Zunächst habe ich zur Klasse Game ein Attribut *allRooms* hinzugefügt welches eine Liste aller Räume speichert:

```
public class Game {
    …
    private Room reception, office, restaurant, storage, courtyard, guest1, guest2, guest3;
    private List<Room> allRooms;
    …
    public Game() {
      …
      allRooms = Arrays.asList(reception, office, restaurant, storage, courtyard, guest1,
                                    guest2, guest3);
      …
    }
    …
}
```

Danach habe ich die Klasse *CommandWords* um ein neues Kommando “map” erweitert. Schliesslich musste noch die Methode *processCommand(Command command)* der Klasse *Game* ergänzt werden:

```
private boolean processCommand(Command command) {
    …
    String commandWord = command.getCommandWord();
    if (commandWord.equals("help")) {
      printHelp();
    } else if (commandWord.equals("go")) {
    …
    } else if (commandWord.equals("map")) {
     printMap();
    }
    return false;
}
```

Die Methode *printMap()* druckt die Räume, wie gefordert aus. *printDescription()* druckt dabei den aktuellen Raum (und weitere Informationen wie die Exits und die Gegenstände die im Raum sind):

```
private void printMap() {
    System.out.println("Rooms and their items:");
    for (Room room: allRooms) {
      System.out.println("   " + room.descriptionWithItems());
    }
    printDescription();
```

```
}
```

3.1+3.2 Der Spieler kann mehrere Gegenstände bei sich tragen

Die Klasse *Game* wird um ein Attribut *takenItems* erweitert, welches die Gegenstände in einer Liste enthält, die der Benutzer aufgesammelt hat. Über die Methode *printDescription()* wird nach jeder Eingabe des Benutzers nicht nur der aktuelle Raum angezeigt, sondern auch die Liste aller eingesammelten Gegenstände.

```
public class Game {
  …
  private List<Item> takenItems;
  …
  public Game() {
    …
    takenItems = new ArrayList<>();
  }
  …
  private void printDescription() {
     System.out.println(currentRoom().longDescription());
     if (takenItems.size() > 0) {
       List<String> descriptions = takenItems.stream()
                                          .map(i -> i.toString()).collect(Collectors.toList());
       System.out.println("You carry: " + String.join(", ", descriptions));
     }
  }
}
```

Nun wird ein neues Kommandowort “take” in der Klasse *CommandWords* definiert und die Methode *processCommand(…)* der Klasse *Game* entsprechend erweitert:

```
private boolean processCommand(Command command) {
  …
  String commandWord = command.getCommandWord();
  if (commandWord.equals("help")) {
    printHelp();
  } else if (commandWord.equals("go")) {
  …
  } else if (commandWord.equals("take")) {
    takeItem(command);
  }
  return false;
}
```

In den Methoden *takeItem(…)* wird implementiert was passiert wenn der Benutzer das neue “take” Kommando wählt. Wird ein Gegenstand aufgenommen, so muss dieser auch im Raum vorhanden sein. Dann wird der Gegenstand aus dem Raum entfernt und zu der Liste *takenItems* hinzugefügt.

```
sprivate void takeItem(Command command) {
  if (!command.hasSecondWord()) {
    System.out.println("Take what?”);
  } else {
    String itemName = command.getSecondWord();
    Item item = findItem(itemName);
```

```
    if (item == null) {
      System.out.println("Unknown item " + itemName);
    } else if (currentRoom().containsItem(item)) {
      currentRoom().removeItem(item);
      takenItems.add(item);
      printDescription();
    } else {
      System.out.println("Room does not contain item " + itemName);
    }
  }
}
```

3.2 Mit Maximalgewicht

Um ein Maximalgewicht überprüfen zu können, muss man wissen, wie gross das Gewicht aller Gegenstände ist die bereits aufgenommen wurden. Ist das Gewicht des neuen Gegenstands plus das Gewichts aller bereits aufgenommenen Gegenstände grösser als das Maximalgewicht, dann kann der Benutzer den neuen Gegenstand nicht aufnehmen.

Im Attribut *takenWeight* der Klasse *Game* wird das Gewicht aller Gegenstände in *takenItems* abgespeichert. Dieses Attribut wird angepasst wenn sich der Inhalt von takenItems ändert. Bevor ein Gegenstand genommen werden kann, wird überprüft, ob dies möglich ist. Im Code unten wird 8000.0 Gramm als Maximalgewicht erlaubt.

```
public class Game {
  …
  private List<Item> takenItems;
  private double takenWeight = 0.0;
  …
  private void takeItem(Command command) {
    …
    String itemName = command.getSecondWord();
    Item item = findItem(itemName);
    if (item == null) {
      System.out.println("Unknown item " + itemName);
    } else if (currentRoom().containsItem(item)) {
      if (takenWeight + item.getWeight() > 8000.0) {
        System.out.println("Cannot take " + item.getName() + ". Total weight too heavy.");
      } else {
        currentRoom().removeItem(item);
        takenItems.add(item);
        takenWeight = takenWeight + item.getWeight();
        printDescription();
      }
    } else {
       System.out.println("Room does not contain item " + itemName);
    }
  }
  …
}
```

3.3 Gegenstände wieder ablegen

Gegenstände können über ein neues Kommando “drop” abgelegt werden. Dieses neue Kommando wird in *CommandWords* eingetragen. Danach wird *processCommand(…)* der Klasse *Game* erweitert:

```
private boolean processCommand(Command command) {
  …
  String commandWord = command.getCommandWord();
  if (commandWord.equals("help")) {
    printHelp();
  } else if (commandWord.equals("go")) {
  …
  } else if (commandWord.equals("drop")) {
    dropItem(command);
  }
  return false;
}
```

Methode *dropItem(…)* legt einen Gegenstand ab. Beim Ablegen eines Gegenstandes muss dieser in der Liste *takenItems* vorliegen, er wird aus dieser Liste entfernt und zu dem aktuellen Raum hinzugefügt. Dabei wird das Gewicht entsprechend angepasst.

```
private void dropItem(Command command) {
  if (!command.hasSecondWord()) {
    System.out.println("Drop what?");
  } else {
    String itemName = command.getSecondWord();
    Item item = findItem(itemName);
    if (item == null) {
      System.out.println("Unknown item " + itemName);
    } else if (takenItems.contains(item)) {
      takenItems.remove(item);
      takenWeight = takenWeight - item.getWeight();
      currentRoom().addItem(item);
      printDescription();
    } else {
      System.out.println(itemName + " not in possession");
    }
  }
}
```