UPPSALA UNIVERSITY   Advanced programming
Artur Kulmizev   March 2021

Assignment 4

In this lab you will design and write an object-oriented program from scratch. The purpose is to build a text-based adventure game. You will also need to create a very simple dialogue system and read from a game description file. You can read more about text-based games at Wikipedia (`https://en.wikipedia.org/wiki/Interactive_fiction`). If you want to try out a game, there is one in Emacs called Dunnet, that you can start by typing the command `M-x dunnet`.

Your task is to write code for such a game, following all specifications below. One of the most important aspects in the lab is your object-oriented design, i.e. which classes you use and how the relations between them look. It is also important to carefully consider which data structures (e.g. list, deque, dictionary) to use for different tasks.

Read the full description before you start coding, so that you understand the full task, and the requirements on your solutions.

1 Organization

For this lab you can self-organize groups of 1–3 students. Fill your group in in Studium once you have decided on it.

2 The game

Your task: implement a text-based adventure game where the player walks around in a house, and can take and use items that are placed in the house. The house should have several rooms with doors in between them, which can be open or closed. The player can only move between rooms with an open door between them. Items of different types should also be distributed among the rooms. Some items are stationary, and nothing can be done with them. Others are movable, and the player can take them, carry around and release them. A third type of items are movable with an action that can be performed when the player holds them. For instance a key could be used to unlock doors.

The game is played by using simple text commands, like `go N`, to go to a room north of the current location, and `take cup` to pick up a cup. An example of how the game can look when played can be found in Appendix A.

2.1 Description of the game world

The game world, i.e. the house should be described in a text file. Rooms, doors, items and starting position should be defined in this file.

A house specification text file should include the following parts:

- Specification of rooms starts with the word `room` followed by the name of the room, which should not contain spaces.
- Specification of doors starts with the word `door`, and has three parts:
  - Direction: `dir1-dir2`, where the direction from the first room is followed by a hyphen, followed by the direction from the other room
  - The name of the first room
  - The name of the second room

  The directions can be any string at all, but I suggest using cardinal directions, like "N" for north. If one wants several doors in the same directions, it is possible to use indexes like "N1", "N2". If one wants a house where it is easy to get lost, it is possible to use non-logical direction pairs like "N-E".
- Which room the player starts in is given by the word `start` followed by a room name.
- Items are specified starting with the word `item`, followed by at least three parts:
  - The name of the item (any string without spaces)
  - The name of the room where the item is
  - The type of the item:
    - `STATIONARY`: the item cannot be lifted
    - `MOVE`: the item can be lifted, but cannot be used for any task
    - `USE`: the item can be lifted and used for some action.
  - If the item is of type `USE`, this should be followed by the action that can be performed if the player holds the item.
- Empty lines, and lines starting with "#" (comments) should be ignored.

All strings should be without spaces, but underscores can be used if you want to have multi word identifiers. The above is a starting point for the file. Depending on your individual extensions to the game, you may need to extend the format slightly.

Figure 1 contains an example configuration, corresponding to the image in Figure 2. You may use this as a starting point for your own house description, if you wish. In principle you should be able to run your code also with this file, possibly without the locked door and the unlock operation for the key, though, which is optional.

In your code for reading the house description files, you can assume that the correct format is used, no extra error handling is needed for that. The house description file should be given as an argument when starting your game:

```
python textGame.py gameConfiguration.txt
```

```
#Rooms
room Hall
room Kitchen
room Storage
room Bedroom

#Doors
door N-S open Hall Bedroom
door N-S closed Storage Kitchen
door E-W locked Hall Storage
door E-W open Bedroom Kitchen

#Items
item box Storage STATIONARY
item hour_glass Kitchen MOVE
item key Bedroom USE unlock

#Start position
start Hall
```

Figure 1: Example house specification file

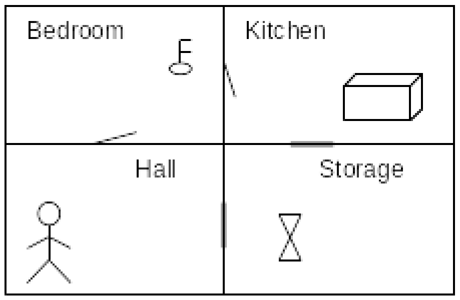

Figure 2: Example game world

2.2 Commands

Your game will include the following commands:

- `go DIR`: Let's the player move to the room in direction `DIR`, if there is an open door in that direction.
- `take ITEM`: Let's the player take the item `ITEM` if it is in the same room as the player, and `ITEM` is not a `STATIONARY` item. The item should then be held by the player, and no longer available in the room.
- `release ITEM`: Let's the player release item `ITEM`, if it is being held. The item will then be in the room the player is in, and should no longer be held by the player.
- `open DIR`: Opens the door in direction `DIR` in the current room, if there is such a door, and if that door is closed.
- `show`: Describes the room the player is currently in, i.e. gives its name, lists the doors, and the available items, if any.
- `commands`: Lists all available commands in the game.
- `holding`: Lists all the items the player is currently holding.
- `quit`: Ends the game.

Beside these pre-defined actions, you should also add at least two of your own commands, that requires the player to hold a specific item. An example can be the command "unlock DIR" that requires the player to hold a key, and that unlocks a door in direction DIR (this would require doors to also have a locked status). You are free to choose these actions and items on your own, but here are some suggestions:

- `unlock DIR` – requires a key to unlock a door
- `read ITEM` – reads some hidden text in an item like a book or note, for instance a clue to the player. The text to be displayed can be added to the house description.
- `drink/eat ITEM` – Probably does not really have any effect, but it is possible to restrict some other actions for which extra energy is needed.
- `open ITEM` – it is possible to have items that can contain other items, for instance a box. These items would become visible after typing `OPEN`. This would require extensions to the house description.
- `keycode DIR CODE` – one can imagine doors having keycodes in order to pass them. Requires modifications to the house description.
- ...

2.3 Interaction

The interface for the game is text-based. The game gives the player information of how the current room looks like, and the player types commands, as specified above. See also the example in Appendix A.

When the game starts it should describe the room where the player starts. The game should also provide feedback on each command the player gives. Some commands, like `show` are requests for information, which should then be printed. For the other commands, if they are successful, a confirmation of the action should be printed (e.g "you have taken the book"), or else an error message should be printed (e.g "there is no book"). For the command `go` a description of the new room should be printed if successful. If the player types in something that is not a correct command, an error message should be given. The game should not crash if the player makes an error in typing some command.

You do not have to worry about having correct grammar or a flow in the language. As long as everything is understandable, that is enough. Texts like "The action open was performed on door N1" or "You took a scissors" will be accepted.

3 Object orientation

The main purpose of this lab, except programming practice in general, is to let you practice object-oriented design, along with all remaining concepts your learned throughout the course. However, the game should be completely object-oriented. That means that all code, except a small main program should be part of a class.

Part of the task is to carefully think about your object-oriented design, which classes you should have, and how they should be connected. An extra demand on your code is that there should be *at least one instance of inheritance*. As a rule of thumb, try to make sure that each class has exactly one responsibility and try to have as few dependencies as possible between classes, so that not all classes knows about each other.

Beside the code you should also hand in a high-level UML class diagram for your project. Each class and the relations between them should be part of the diagram, but you do not need to specify methods and variables for your classes. It is recommended to work with your code and diagram in parallel. A good work flow is to start by sketching a diagram, which is then updated while you do the coding, and possibly realize that some of your thoughts were not perfect from the beginning. Carefully check that the code and diagram are consistent before you hand them in.

4 Requirements for your code

Your game should fulfill the following requirements:

- The game should work as outlined in this text
- All commands described above, plus two own commands should be implemented
- The house should fulfill:
  - There should be at least six rooms (you may etiher extend the specification file described in this document, or create a new house from scratch)
  - There should be both open and closed doors when the game starts
  - There should be at least one item of type `STATIONARY`, one item of type `MOVE` and two items of type `USE`, one for each additional action you implement
- The game should not crash, but give an error message if the player types something strange
- The game should be completely object-oriented, with no code outside classes, except a short main program
- Inheritance should be used at least once

5 Report

You should also hand in a report of about 1 A4 page. It should contain a discussion of your object-oriented design, and can bring up the following questions.

- *Was it easy to solve the problems that came up, using the design you choose?*
- *Was there something in your design that made some part of the implementation difficult?*
- *Are there some choices in the design that you regret and wish you had solved differently?*
- *Was there something in your original design that you realized that you needed to change once you started coding?*
- *Is there something about your design that you are specifically satisfied with?*

One example of something that can be either easy or difficult depending on your design is that if you open a door from one direction, it should also be open from the other direction. Also bring up your choices of data structures. Choose 2–3 examples of data structures you choose, and describe reasons, pros and cons with this choice. For instance, how did you keep track of all the different rooms, using a list, deque, dictionary or other structure, and why? Note that your program will be small, so the time complexity of operations will not have any notable impact, but imagine that you scale this game up to be very large. Would your data strucutres still be a good choice then?

In addition, briefly describe the two actions you choose to add to your game.

What to hand in

All of you should hand in the following:

- All your code
- A UML diagram (could be drawn by hand and scanned/photoed, or drawn using some software tool)
- Your house description file
- A printout of a run of your game that illustrates the required and your own commands, and some error messages (like in Appendix A)
- Your report

All files should be handed in via Studium. The main deadline is April 2, and the resubmission deadline May 15.

Appendix A: example game run

Below is an example of a test run of a text-based game. This game is consistent with the example world described above. It contains the command `unlock` which requires a key. Commands typed by the player are indicated as starting with a prompt symbol >. View this as an example. The exact formulations of answers may be different in your game.

```
> python textGame.py gameConfiguration.txt

Welcome to the house game!
You are in the Hall.
There are doors towards E, N

Please type commands in order to play the game!

 > go E

The door is locked

> go N

You are in the Bedroom.
Here are doors towards S, E
Here are the following items:
key

> take key

You took key

> show

You are in the Bedroom.
Here are doors towards S, E

> go E

You are in the Kitchen.
Here are doors towards S, W
Here are the following items:
hour_glass

> take hour_glass

You took hour_glass

> go S
```

```
The door is closed

> open S

Opened the door

> go S

You are in the Storage.
Here are doors towards N, W
Here are the following items:
box

> take box

It is not possible to take box

> go W

The door is locked

> unlock W

The door is unlocked

> open W

Opened the door

> go W

You are in the Hall.
There are doors towards E, N

> go E

You are in the Storage.
Here are doors towards N, W
Here are the following items:
box

> holding

You are holding:
key
hour_glass

> release hour_glass
```

```
hour_glass released

> show

You are in the Storage.
Here are doors towards N, W
Here are the following items:
box
hour_glass

> holding

You are holding:
key

> release box

You do not hold box

> take cup

There is no cup

> go S

There is no door

> jump

I do not understand

> open

What should I open

> quit

Good bye and thanks for playing!
```