OOSD1 Tutorial/Practical for Week 1 Semester 2

This tutorial is aimed at helping you and you group to get started with the assignment.

Preliminaries

Firstly you should have assigned yourself to a group and if you have yet to find one take the time now to talk with a tutor. week work in groups of 3 or 4.

If you have not already done so, then its time to download the assignment material, which can be found on Blackboard under:

Learning Material -> Assignment

Finally, there is a a short video describing how to access all the provided material and how to get the provided Netbeans project up and running here:

https://www.youtube.com/watch?v=ckGPRvt29LM&t=3s

you should watch this now, if you have not already done so.

At this point you should have a working Netbeans project that the Zombie frontend can connect to.

Loading the world

One of the key elements of the assignment is to find a suitable representation of for the game world, such that the player can issue commands, e.g. to move rooms or pickup an item, and the server to interpret these and respond, updating the world as necessary, e.g. to remove a picked up item from the current room and place it in the players inventory.

To start with you need to use the class WorldLoader to load the world, performed by the constructor, and then call its methods to find out about the world, i.e what rooms there are, what items there are, and so on. These

additions should be made to the supplied starter project, initially with changes to the *main* method in the class *ZombieStarter*.

To create an instance of the *WorldLoader* you need to simply call the default constructor adding the following within the *try* { … } block within the *main* method:

```
WorldLoader worldLoader = new WorldLoader();
```

Remembering you will need to import the package *world*:

```
import world.*;
```

At this point the world will have been loaded into the object *worldLoader*. This object contains everything you will need to build an interactive version of the world for your assignment. For example, the method:

```
java.lang.String                getInfo()
```

returns a string that should be displayed when the player issues the command *"info"*. All the methods of WorldLoader are documented in the supplied HTML Javadocs, which you should open now by clicking on the index.html found in the directory *oosd1-assignment-javadocs*, created when you unzipped the Javadocs.zip file.

There are many ways to proceed from here, but one possible approach is to create a world class, which will be the backbone of your game, it will keep track of the different rooms, it will know which room the player is currently in and so on.

Creating the World

So assuming you are going to create a world class, go ahead and do this now, remember it won't need a main method.

What, if any arguments will its constructor take?

Well one thing that might be useful is to store the *info* string, returned by the *getInfo*() method. Go ahead and add an *info* argument to your world's constructor, remembering that you will need an attribute to store it into and so on.

Now you need to create an instance of your world class in the main method and pass in the string returned by getInfo().

What other things will the world keep a track of? Here are some ideas:

- A list of rooms?
- The current room that the player is in?
- The players inventory?
- Others?

Many of these properties initial values can be derived from the *WorldLoader* class, for example the method *iterator()* returns an object that can be looped over to access the different rooms in the world.

Of course, this is what much of the assignment is concerned with and we will leave those aspects for you to consider over the next few weeks. For now we will focus on implementing the command *"info"* now that we have the string that should be displayed stored in our world class.

Implementing the "info" command

The class *ZombieServer*, also provided as a library, interacts with the frontend and to play the game, in particular, it makes calls to an instance of the class *ZombieBot*, which in turns interprets player commands and returns HTML strings to be displayed by the frontend.

If you look at the bottom the *try* { … } block, in the *main* method, you will see the following code:

```
ZombieServer zs = new ZombieServer(
        // get host address, rather than using 127.0.0.1, as this
        // will then be displayed when server waits for connection
        // which allows the address to then be typed into client.
        ip.getHostAddress(), //"127.0.0.1",
        8085,
        new ZombieBot());
```

Notice the creation of an instance of the *ZombieBot*. Currently the supplied implementation of the *ZombieBot* class simply provides default implementations for methods required by *ZombieServer*. It is your task to provide implementations for each of the methods, such that the game can be played. Again all the documentation for *ZombieBot* can be found in the Javadocs.

One such method is *processCmd()*, which is called by *ZombieServer* whenever it receives a command, entered by the player, from the frontend. It has the following prototype:

```
java.util.List<java.lang.String> processCmd(java.lang.String cmd)
```

You should open up the *ZombieBot.java* file in the Netbeans project and scroll down until you find the implementation of the *processCmd()* method. This method will look something like:

```
@Override
public List<String> processCmd(String cmd) {
    ArrayList<String> result = new ArrayList<>();
    String[] cmds = cmd.split(" "); // split cmd by space

    switch(cmds[0]) {
       case "info":
          result.add("handle info command");
           break;
       /* rest of switch */
     }

     return result;
}
```

The argument cmd is the command entered by the player to be interpreted and corresponding actions, implied by the command, implemented by *processCmd*(). The variable *result* stores a list of strings that will be returned to the frontend to be displayed back to the player. result is retuned back to *ZombieServer*.

The main part of *processCmd*() is the switch statement that performs different actions, depending on the command passed in. Currently we want implement the command "info". In this case what should *result* contain? It needs to be the string that we passed in to the constructor of our *world* class! This means we

need a way of referencing our world class from within *ZombieBot*. The simplest solution to this is to place in the world instance to the constructor of *ZombieBot*. To this end you should extend you *ZombieBot* constructor to take your world class as a parameter and store it as an attribute. You need to update the creation of the *ZombieBot* instance, in the main method, to look something like:

```
ZombieServer zs = new ZombieServer(
        // get host address, rather than using 127.0.0.1, as this
        // will then be displayed when server waits for connection
        // which allows the address to then be typed into client.
        ip.getHostAddress(), //"127.0.0.1",
        8085,
        new ZombieBot(world));
```

Where *world* is an instance of the *World* class you created as part of the previous section.

Finally, all that remains to implement the "info" command is to replace the line:

```
        result.add("handle info command");
```

With a call to a method in your world class that returns the info string, e.g. something like:

```
        result.add(world.getInfo());
```

(If you have not yet implemented a method to return the info string, now would be a good time to do it ☺)

Now try running the game, connect the frontend to you modified server and once running enter the command "info" and you should see the string:

```
Welcome to OOSD1 Zombies

Zombies are real, no matter what you mum told you! UWE's campus has been invaded
and you must kill the zombies and get to the exit room, to save yourself and the
University. Good luck!
```

Displayed in the game window.

What next?

Now you have a world object its time to implement other aspects of the game. For example, one of the key aspects of the world is rooms, the world is made up of rooms, each room can have zero or more entrances that lead to other rooms. As a next step I would recommend adding rooms to your world, using the following steps:

- Walk over the rooms (return by *iterator()* from WorldLoader) adding each one to your world. Add the room's name as you create it.
- Each room has a list of entrances, add these entrances to your world. You need to be careful here as an entrance points to a room and so it is important to create all rooms first, hence the first step. Each entrance should have a corresponding direction and room that it leads to.
- Now update the rooms in your world to contain a list of its entrances.
- If you correctly implemented the above steps you should now be able to implement a basic "move" command, allowing the player to move around the world.

Once you have achieved the ability to move around the world, then you can add the extras, e.g. "look" command, picking up and dropping items, and finally zombies!