Lab ISS | the project cautiousExplorer with Actors

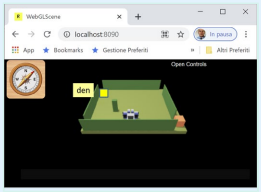
Introduction

This case-study deals with the design and development of proactive/reactive software systems based on the concept of Actor, as introduced in LabIss2021 | wshttp support with ActorBasicJava observers.

Requirements

Design and build a software system that allow the robot described in ***VirtualRobot2021.html*** to exibit the following behaviour:

- the robot lives in a rectangular room, delimited by walls that includes one or more devices (e.g. sonar) able to detect the presence of obstacles, including the robot itself;
- the robot has a **den** for refuge, located in the position shown in the picture



- the robot works as an *explorer of the room*. Starting from its **den**, the goal of the robot is to create a map of the room that records the position of the fixed obstacles. The presence of mobile obstacles in the room is (at the moment) excluded;
- since the robot is *'cautious'*, it returns immediately to the **den** as soon as it finds an obstacle. It also stops for a while (e.g. 2 seconds), when it 'feels' that the sonar has detected its presence.

Delivery

The customer requires to receive at **12 noon on April 6** a file named

```
cognome_nome_cea.pdf
```

including a (synthetic) description of the project (preceded by a proper analysis of the problem) based on components of type ActorBasicJava and a reference to a *first working prototype* (even with limited capabilities) of the system.

Meeting

A SPRINT-review phase with the customer is planned (via Teams) at **5.15 pm on April 6**.

Requirement analysis

The constumer requires to design and buid a software system that is able to drive a *robot*

to walk in a rectangular room in order to explore the sorrounding reality. The consumer also specifies that the robot lives in a rectangular room.

The interview with the constumer clarified the meaning of the used nouns that are the following:

- **robot**: device already owned by the consumer and able to walk by receiving commands. The robot is fully described in VirtualRobot2021.html, a document provided by the consumer;
- **rectangular room**: the room in wich the robot lives and walks, delimited by **walls**;
- **den**: the **home** of the robot (a sort of landmark), in other words the place in the room where he starts the **exploration sessions** and goes back when finished; the den must be located near a wall as stated in the requirements:
- **obstacles**: entities external to the robot that can stay into the environment; at the moment the obstacles are permanent and each of them occupies a portion of the environment in which the robot cannot pass;
- **sonar**: another entity external to the robot that is a device able to detect the presence of the robot;
- **explorer**: the *role* of the robot that represents his aim to find the obstacles around the den;
- **position**: an identifiable place (we could also call it *point*) of the environment that can only be occupied by one entity (obstacles, devices or robot) at a time;
- **cautious**: the property of the robot to return immediately at the den when he encounter an obstacle; in addition to this, the robot must stop for a while when it feels that a sonar has detected his presence and then return to the home;
- **map**: a support to the robot that make it able to remember details about the room like the position of the obstacles;

After the analysis of the nouns, we introduced a new one:

- **exploration session**: it consists in these steps:
  1. robot start moving from the den;
  2. robot explore environment moving in it (**round trip**);
  3. robot finds an obstacle by colliding with it or feels that a sonar has detected him;
  4. after collision, robot immediately return ad the den (**return trip**); in the case of sonar detenction, the robot first waits for a while and after returns to the den.

About the meaning of the verbs used in the interaction with the consumer:

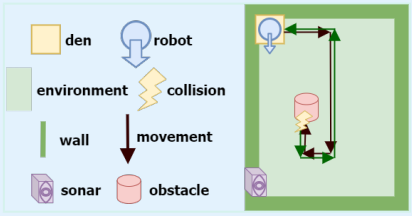
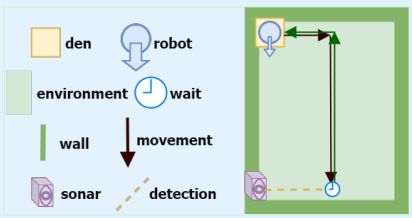
- **lives**: the action of the robot to stay and move only in the closed environment;
- **detect**: the action of a sonar to notice the presence of the robot; the presence will be notify in the way specificied in the document VirtualRobot2021.html; when a sonar detect the robot, optionally he return to his den;
- **move**: the action of the robot to walk into the environment thanks to the mean of locomotion it is equipped with; consumer specified that the movements of the robot can be random or oganized with an organized logic;
- **find**: the action of the robot to collide with an obstacle in order to identify its position.
- **return**: the action of the robot to move towards the den; the consumer as not specify

details of the route that he should take.

The robot is able to receive command in *cril language* coded into JSON strings that can be **encapsulated in HTTP POST messages** or **sent on a websocket** as specified in VirtualRobot2021.html. Then the software system to be designed must use these mechanism to carry out and manage the exploration of the robot. The consumer has not provided details about:

- the number of the *exploration sessions*;
- when the application should stop;
- the route of the exploration;
- the route of the return trip;
- the length of the time the robot must stop when a sonar has detected him;

User Stories

First user story The user places the robot in the **den** facing south and then he starts the system. The system start the exploration session by moving the robot randomly (dark red movements into the right image). The **user can't stop the execution of a single exploration session** and the system must proceed autonomously without user interactions. **When the robot collide with an obstacle, the system must send the correct commands in order to move the robot back to the den**. The movements of the robot must be the same of the round trip but inverted like in the right image (dark green movements). At the single exploration session, the robot must be positioned in the den ready for another session.	
Second user story The user places the robot in the **den** facing south and then he starts the system. The system start the exploration session by moving the robot randomly (dark red movements into the right image). The **user can't stop the execution of a single exploration session** and the system must proceed autonomously without user interactions. **When the sonar detect the robot, the system must send the correct commands in order to stop the robot waiting for a while and after move the robot back to the den**. The movements of the robot must be the same of the round trip but inverted like in the right image (dark green movements).	

At the single exploration session, the robot must be positioned in the den ready for another session.	

In these user stories we supposed that the return trip must be the same of the round but in future we could make this mechanism smarter using the map (e.g. calculating the trip using the map). About the exploration instead, we think that an efficient way is to proceed by making a sort of *waves*. We will delve into this topic in the problem analysis.

Problem analysis

Relevant aspects

1. From the requirement analysis it emerged that the system that is about to be implemented must communicate with the robot via HTTP or via websocket using commands coded in JSON String. In order to do this it need to create a **distributed system** with two main components:
   - the **robot** provided by the consumer;
   - an **application** able to send the command to the robot and manage the responses.

   The application will be called ***cautiousExplorerActors***.

2. As anticipated, the robot and the application can communicate:
   - sending messages to the port 8090 with **HTTP Request/Response**;
   - sending messages to the port 8091 using **websocket**.

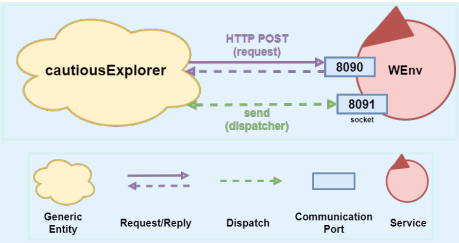
   In addition to, as specified, the commands must be coded in **JSON string**.

The communication is not always *request/response* and, in addition to this, collisions and sonar informations can't be received via HTTP: **it means that we must use websocket** in order to satisfy the requirements. Then, we have an **operative abstraction-gap** even if there are lots of libraries for websocket in Java. Furthermore, in order to separate the parts of the software system, we decided to use ***actor model programming*** so we could have another operative abstraction-gap. Fontunately, the consumer has provided a library that allows us to easily use actor model in Java (wssupportAsActorJava.html). The last operative abstraction-gap is given by the **construction of the map** of the room.

Logical Architecture

In the relevant aspects it was said that system has two main components then logical architecture is the following:

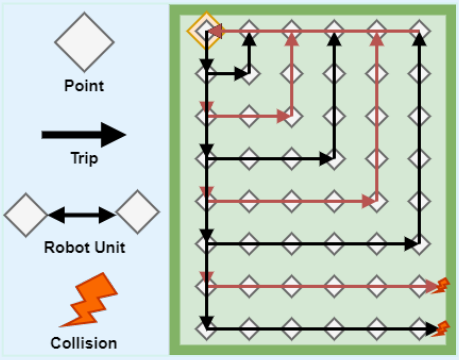
	For now **cautiousExplorerActors** it's a generic component (entity) and it will be clarified in progect phase even if we can say that it is composed by some actors. **WEnv** is the environment of the robot able to receive commands via HTTP or websocket.



About interaction instead:

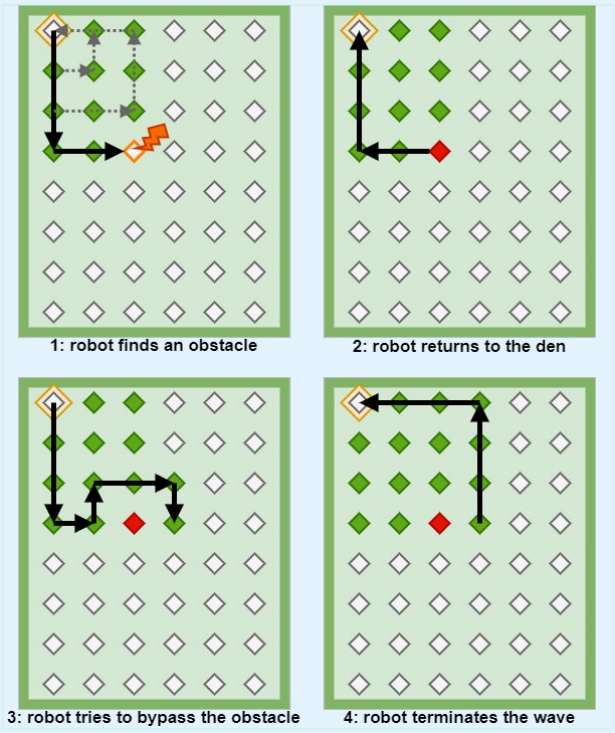
- the use of **HTTP** seems very suitable and very portable thanks to the existence of lots of API;
- the use of **websocket** could be much more efficent decreasing the overhead costs of HTTP messages and in addition to the usage of this component allows asynchronous interation in a natural way. In order to satisfy the requirements, websocket is needed for the asynchronous interaction.

Exploration Logic



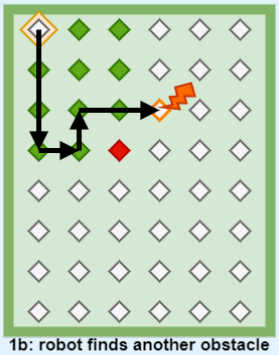
As said in the requirement anlysis, the consumer did not specify how the robot should move to explorate the room. The cril language let us to specify the time of a single move of the robot than, if we set a constant time (**single move time**) for all forward move, it's possible represent the room into a cartesian plan. The distance of the points in the plan is called **Robot Unit** (RU) and it is the distance travelled by the robot in a *single move time*.
We decide that an efficient way it to proceed by making a sort of *waves* like in the image (**explorarion wave**).

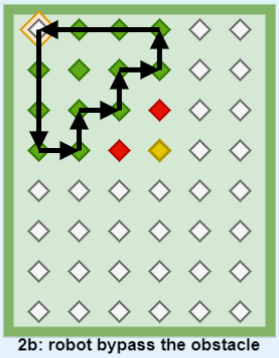
Collision Coping

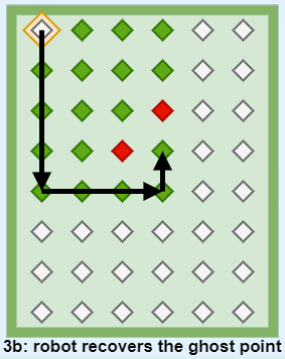


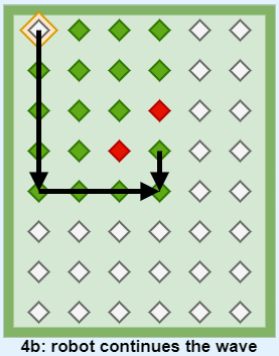
Case a: single obstacle

Suppose that the robot is making an exploration wave and at some point the robot collide with an obstacle *(1)*: then the robot mark the point ad occupied by an obstacle and immediately returns to the den *(2)*. After that, the robot must complete the exploration wave so he has to *bypass* the obstacle and he can make it passing around him *(3)* and then ending the wave *(4)*. If at the point 3 the robot finds another obstacle, he should do the same thing and try to bypass also this new one.


1b: robot finds another obstacle

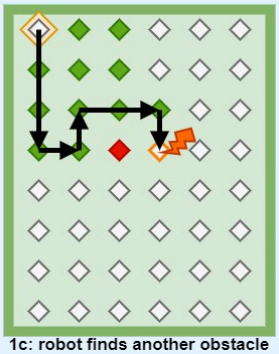

2b: robot bypass the obstacle

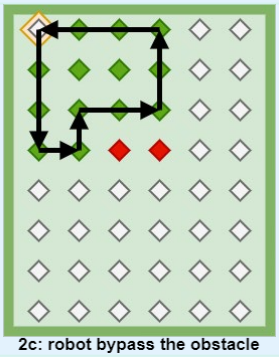

3b: robot recovers the ghost point


4b: robot continues the wave

Case b: two diagonal obstacles

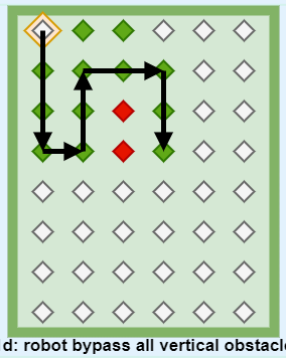
It is possible that in the points *(3)* of the first image the robot finds another obstacle in the position specified in the left image *(1b)*. The new one obtsacle must also be bypassed with the same way (after that the robot is returned to the den). However, if the obstacle is in this position there is a *lost point* of the wave (yellow diamond in *(2b)*, we called it **ghost point**). Robot can recever this point at the next wave with a small deviation *(3b)* and after normally continues the wave *(4b)*.

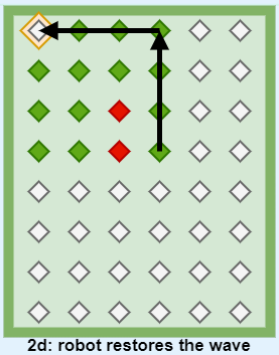

1c: robot finds another obstacle


2c: robot bypass the obstacle

Case c: two horizontal obstacles

It is also possible that in the points *(3)* of the first image the robot finds another obstacle in the position specified in the left image *(1c)*. The new one obtsacle must also be bypassed (after that the robot is returned to the den) in this simple way in order to terminate the current wave *(2c)*.

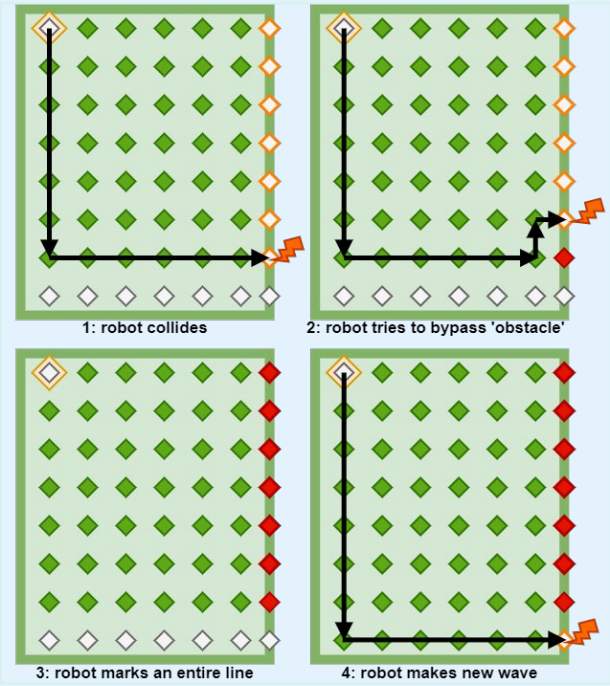

1d: robot bypass all vertical obstacles


2d: robot restores the wave

Case d: two vertical obstacles

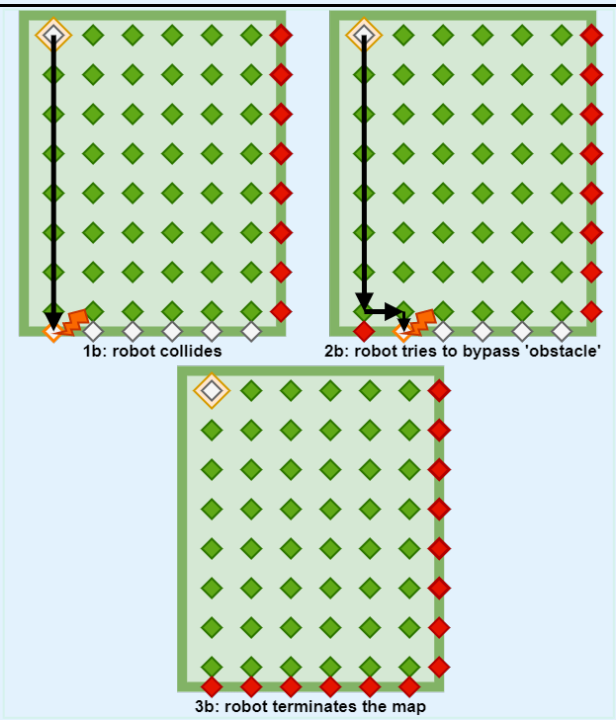
Suppose that the robot has encountered a new one obstacle that is vertical to another found in the previous one wave. In this case he has to bypass the obstacles like in *(1d)* and after restores the wave *(2d)*. Notice that in order to apply this solution it's needed at least one free point above the two obstacles. Otherwise the case is more complicated and we well talk about it later.

Boundary Detection



Case a: right boundary side detection

At a certain wave, the robot will collide with boundary without knowing it's it *(1)*. Then robot several tries to bypass all *fictitious* obstacle represented by the right side of the boundary *(2)* until he marks an entire line *(3)* terminating the current wave. When the start the new wave, the robot newly collides with the boudary but at this time he returns to den and then starts the next way without trying to bypass the 'obstacle'.

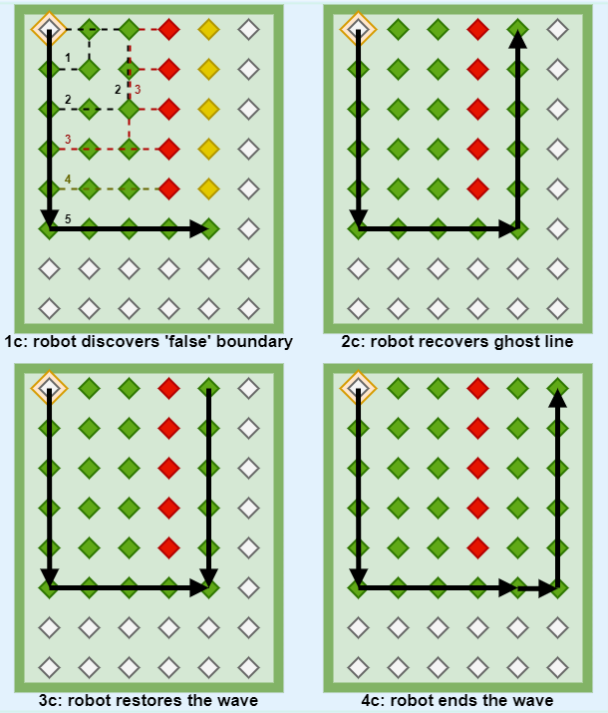


Case b: south boundary side detection

When the robot has done the last wave into the room, he start the *last* wave and collides with the south side of the boundary *(1b)*. After that, he tries to bypass the *fictitous* obstacle repesented by the same side and makes it several times *(2b)*. When he marks all the side, he ends the exploration *(3c)*.

Case c: false right side

It is possible that there are lots of obstacle that form a vertical line *deluding* the robot that it is the right side of the boundary. In this case it's also possible to have some **ghost lines** *(1c)* that must be recovered *(2c)*. It could be very difficult to recover these lines but simplifying the robot should



recovers theese with a strong logic and after he must restores the current wave *(3c)* and terminates it *(4c)*. Pay attention that if the situation is the one on the left, the robot can not return to the den like a normal wave but he has to bypass the line formed by the obstacles.

Remarks

We observe that:

- The specification of the exact 'nature' of our ***cautiousExplorerActors*** software is left to the designer. However, we can say here that is it **not a database, or a function or an object** and that it is composed by some actors.
- To make our ***cautiousExplorerActors*** software **as much as possibile independent** from the undelying communication protocols, the designer could make reference to proper ***design pattern***, e.g. **Adapter**, **Bridge**, **Facade**.
- It is **not** quite easy to define **what the robot has to do** the meet the requirements and memory map is needed in order to do an efficient exploration. However we make a cumulative algorithm for the exploration:

```
let us define emum direction {UP,DOWN,LEFT,RIGHT};
let us define map room_map;
let us define int wave;
the robot start in the DEN position, direction=DOWN, wave = 0;
do
  1) wave++;
  do
    1) send the robot the request to do the next step of the wave;
    2) if collision or sonar information are detected then cople it;
    3) propertly update room_map;
  while the wave is not finished;
while robot does not detect bot the south and the right side;
```

Test plans

Project

Testing

Deployment

Maintenance

By student
Name: Luca Marchegiani
Email: luca.marchegiani3@studio.unibo.it
Git Repo: https://github.com/LM-96/MarchegianiLuca