Mission 16: Rookie Training

Start date: 24 October 2017
Due: 31 October 2017, 23:59

IMPORTANT WARNING:

Because we provide you with the flexibility in choosing the approach by which you solve the problems in this mission, we require that you submit well commented/annotated programs. Describe your approach to the programming questions, and then comment/annotate the relevant parts that implements your idea. If you fail to do so, you risk having marks deducted by your tutor.

To better prepare everyone for the upcoming confrontation, Executive Officer Kenneth has been placed in charge of the combat training of all the initiates onboard the ship. The XO reminds all trainees that the assault will be done as a large group, thus coordination would be of the utmost importance. The purpose of this training would be for everyone to decide upon their own strategy for attack, allowing the XO to plan a more comprehensive assault strategy.

Leading all the initiates to the ship's Simulation Room, Kenneth explains that all training would be done here, which allows for separate simulated battlefields to be set up for each trainee, without cross-interference. By modifying the structure of the simulation, you can also train in different scenarios. XO Kenneth wishes all of you the best of luck, and motions you to proceed.

Task 1:

This task is worth 4 marks.

The following shows a portion of the definition used by the Simulation Room to create ServiceBots:

```
function ServiceBot(name, initLoc, inertia){
    Person.call(this, name, initLoc);
    this.__inertia = inertia;
}
ServiceBot.Inherits(Person);
ServiceBot.prototype.__act = function(){
    Person.prototype.__act.call(this);
    if (math_floor(math_random()*this.__inertia) === 0) {
        var randomLoc = this.__pickRandomDir();
        if (!is_empty_list(randomLoc)) {
```

```
            this.moveTo(head(randomLoc));
        } else {
            ;
        }
    } else {
        ;
    }
}
function MakeAndInstallBot(name, initLoc, inertia){
    var newBot = new ServiceBot(name, initLoc, inertia);
    newBot.__install(newBot.getLocation());
    newBot.take(list(MakeAndInstallKeycard(newBot.getLocation())));
    return newBot;
}
var bot1 = MakeAndInstallBot("b1", bot1_place, 2);
var bot2 = MakeAndInstallBot("b2", bot2_place, 3);
```

After creating the simulation engine and beginning the simulation using `engine.__start();`, observe how `bot1` and `bot2` move around by clicking Next Action. Once you're familiarised with the system, call `engine.__runRounds(10);` to simulate 10 rounds, each denoted by a `---Tick---` output message in the console.

(a) Which bot is the most restless?

(b) Describe the mechanism that determines who is more "restless".

(c) Theoretically speaking, how often will you, on average, see both of them making a move at the same time? Is this your observation if you wait for `---Tick---` to appear 10 times? Why or why not?

The Training Begins

The Cubic Mothership: Training brief by XO Kenneth

Having analysed the structure of the Death Cube from afar, our Advance Scouts have determined that it has at least 4 levels, each with at least 4 rows and 4 columns of rooms. One of the rooms holds the force field generator, the destruction of which being the aim of the assault. You should thus all train in simulations consisting of `4 * 4 * 4` rooms.

As the actual location of the generator room and the connectivity of the Death Cube are unknown, you should not hope to memorize any route, rather, you shold adopt a strategy that will be able to find the room no matter where it is located.

Your task, then, is to find the room and obtain the means to enter it.
Also, as we are preparing for a group assault, refrain from moving more than one room at a time, as you may cause distractions to your comrades in the actual battle.

Best of luck, all of you. You may proceed.

Task 2:

This task is worth 6 marks.

Due to its importance, it is likely that the generator room will be protected in some way. Further analysis of the reports, however, unearths a photograph of a service robot gaining access by tapping a card on the door frame. It would also appear that there are a few of these cards carried by bots, and that any of them can be used to gain access.

NOTE: The `is_instance_of` function will return true when applied to an object and its corresponding type. For example, you can check if an object is a `Keycard` if it answers `true` to `is_instance_of(object, Keycard)`.

We will simulate the generator room using a `ProtectedRoom`. The `ProtectedRoom` is like a normal `Room`, except that only a `Person` holding a `KeyCard` will be allowed to enter.

Time to get yourself in that room. Implement a class that extends `Player` such that, in every turn, you will do all of the following:

- attack a ServiceBot in the same room as you, if you have a charged Weapon,
- pick up a Keycard in the same room as you, and
- enter a ProtectedRoom if it is beside your current Room and you have a Keycard.

NOTE: As you will not be alone in the actual battle, you **MUST** check if it is a `ServiceBot` before attacking, even during this training!

You will be assigned a few weapons during the final battle. To train for that, you will start by learning to use one weapon. To make an attack on, for example, bot1 using a lightsaber, you will need to do the following:

```
var lightsaber = [find your own lightsaber];
var targets = list(bot1);
this.use(lightsaber, targets);
```

As the weapons you will be provided with are yet unknown, XO Kenneth suggests that you search your possessions for a usable weapon when you need to attack using the `is_instance_of` function. You can determine if the weapon is ready by checking calling its `isCharging` method.

Remember that you can only move to a new room once per round.

Submit your definitions of a class that extends `Player`.

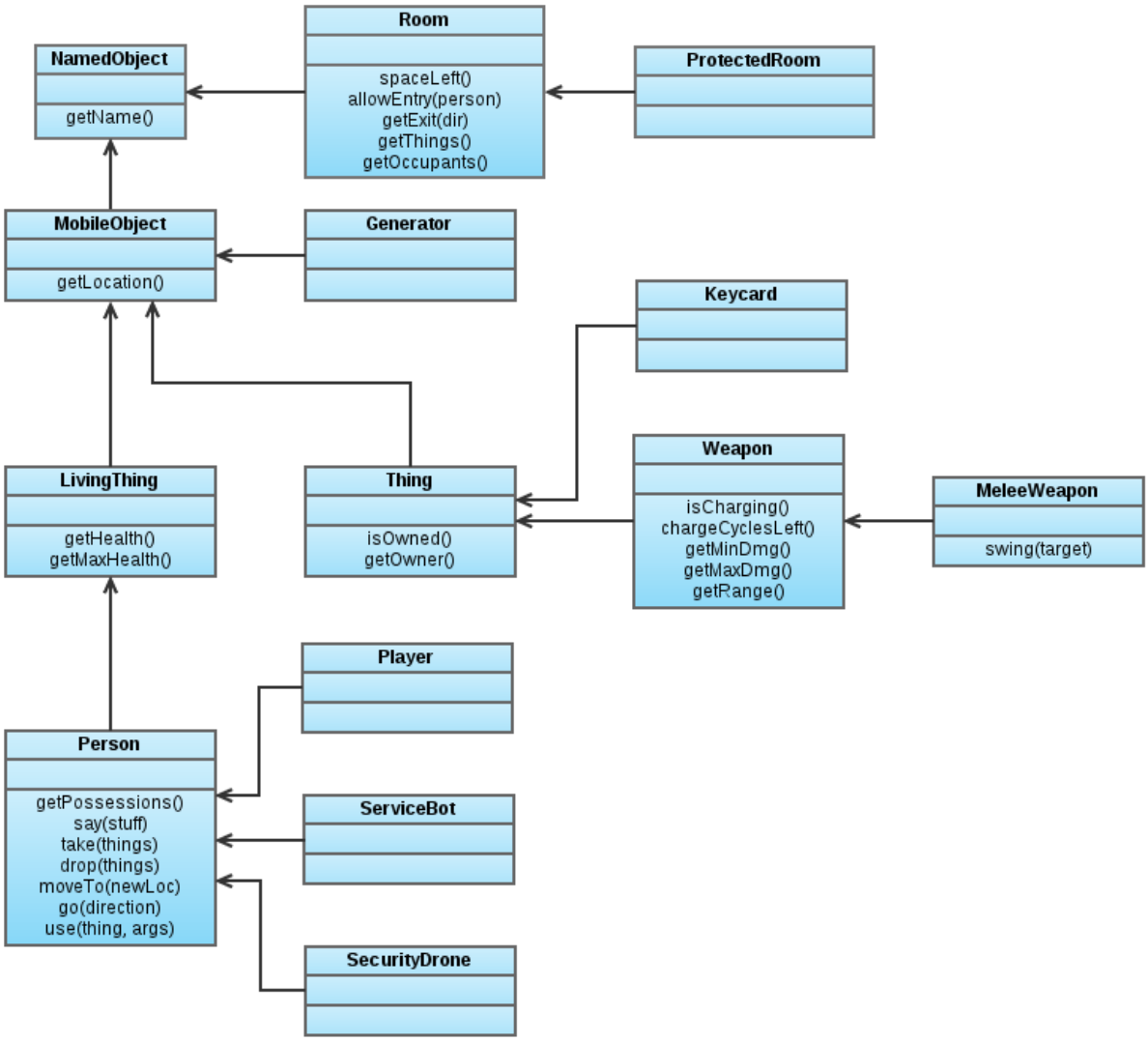
HINT: You should check the appendix for other available methods that might be useful.

Submission

To submit your work to the Academy, place your solutions into the box that says “Program” on the mission page, click “Save”, then click “Finalize Submission”. Note that submission is

final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.

Appendix: A simplified representation of our object hierachy is as follows:



Note that if you choose to explore the the simulation libraries, you will find many private methods and properties not intended for use in your solutions. These private methods and properties are indicated by being prefixed with two underbars (e.g. `LivingThing.prototype.__act`). You will be penalized for using any of these properties.

Certain methods return results of different types. Use `is_object` or `is_instance_of` appropriately for such cases.

Appendix: Classes and Methods Reference

Room's Methods:		
`spaceLeft`		Returns the remaining space in the room for occupants
`getThings`		Retrieves a list of unowned things inside the room
`getOccupants`		Retrieves a list of all living things inside the room
`getExit`	`dir:string`	Retrieves the room in that direction, or `false` if there is none
`getExits`		Retrieves the list of directions leading out of the room
`getNeighbours`		Retrieves the list of rooms adjacent to the room
`allowEntry`	`person:Person`	Checks if `person` can enter

Valid strings for `getExit`: ("north", "south", "east", "west", "up", "down")

LivingThing's Methods:	
`getHealth`	Retrieves current Health Points
`getMaxHealth`	Retrieves maximum Health Points

Person's Methods:		
`getPossessions`		Retrieves list of items owned
`say`	`stuff:string`	Says stuff
`take`	`things:list of things`	Takes all specified items
`drop`	`things:list of things`	Drops all specified items
`use`	`thing:Thing, ...`	Uses the item. See the items table for more details
`moveTo`	`newLoc:Room`	Moves to `newLoc`. Must be adjacent to current locatio
`go`	`dir:string`	Moves in that direction

Valid strings for `go`: ("north", "south", "east", "west", "up", "down")

Weapon's Methods	
`isCharging`	Check if a weapon is charging
`chargeCyclesLeft`	Check remaining charging time
`getMaxDmg`	Checks max damage
`getMinDmg`	Checks min damage

Weapon's Methods	
`getRange`	Checks weapon range

Using Weapons:	
`this.use(melee_weapon, list(target))`	Use `melee_weapon` to attack selected `target` in the same room