National University of Singapore
School of Computing
CS1101S: Programming Methodology (Scheme)
Semester I, 2011/2012

Mission 18:
Advanced Training

Issued: 19 October 2011

Due: 26 October 2011, before 23:59

Readings:

- SICP: Chapter 3, Section 3.1, 3.2, 3.3
- Concrete Abstractions, Chapter 14
- Lecture notes on Object-Oriented Programming

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 code. Describe your approach to the programming questions, and then annotate the blocks of relevant code that implements your idea. If you fail to do so, you risk having marks deducted by your tutor.

The Training Continues

Guardians of the Generator

Training Brief by XO Zi Han

One of our Advance Scouts has been able to enter the Death Cube under disguise. She has been able to observe the service bots up close, and it would appear they bear no weapons, only tools for repair and construction. However, when she attempted to disable the service bots, armed security drones started streaming out of the walls of the room. Not only was the supply of drones continuous, they appeared able to spawn more quickly as time passed. Luckily, our Scout managed to escape, and is now recuperating.

Using this hard-won information, you should now train to defeat the potential enemies. Again, I must caution you against aimless wandering while searching for the service bots, and risking separation from your team by moving across too many rooms at once.

Our fleet fast approaches the Cube. I urge all of you to hasten your training. May the Force be with us all.

Mission Start

This mission consists of **two** tasks.

Unless requirements overlap, do not remove the requirements of past Tasks and Missions.

Task 1: (3 marks)

Instead of wandering around aimlessly, you will need to have a better method of finding `service-bots` and `security-drones`. Every turn, you should

- remember the room you are in,
- attack any bots or drones in your room, and
- move towards an unvisited room if there is one, or a random one otherwise

Make appropriate changes to `make-player` to simulate this behaviour. Mark your changes using comments (for example, `; M18 T1`).

Task 2: (3 marks)

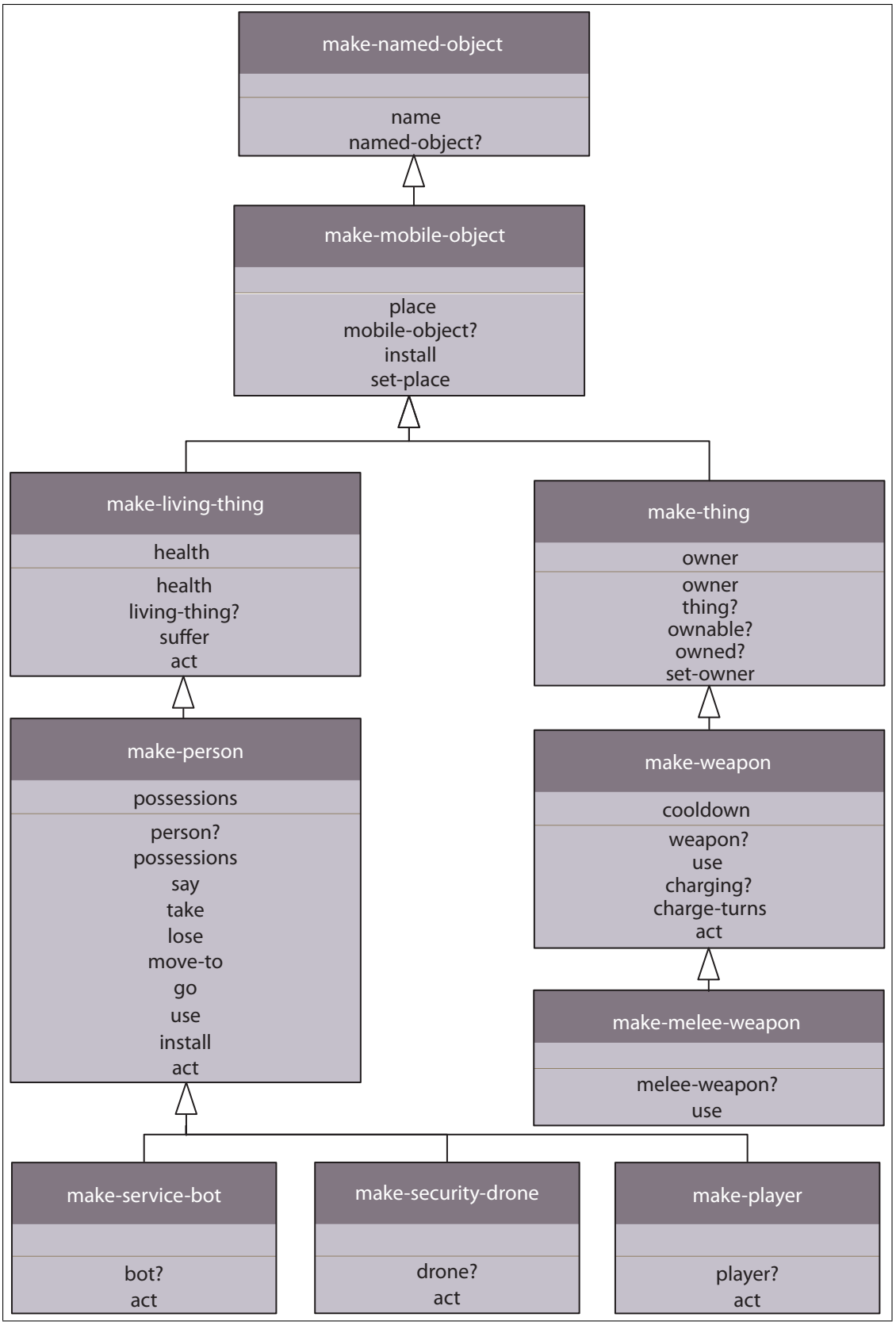
The moment a service-bot is destroyed, an alarm will sound and a security drone will spawn at that location equipped with a `zapray`. Each turn, the security drone(s) will zap unauthorised people carrying cards in their rooms and pick up any key-cards lying on the floor. It is thus **very** important that you find the generator-room quickly.

Since you are already memorising the rooms you visit, we can modify that to aid in finding our generator-room. The generator room will be a `protected-room`, and would contain a `generator`. Now, in each turn, you will

- find out if the generator-room is nearby (beside you), and either
  - remember its location if you don't have a `key-card` or
  - enter the generator-room if you have a `key-card`. Also, you should
- avoid entering any `protected-rooms` if you don't have a `key-card`, and
- once you have a `key-card` and know where the generator-room is, you should head to the generator-room. You MUST with every move you make, make progress towards the generator room i.e. you should not utilise any scheme which involves moving around randomly - instead the path that you take should be deterministic and progressive in arriving at the generator room.

Make appropriate changes to `make-player` to simulate this behaviour. Mark your changes using comments (for example, `; M18 T2`).

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



Appendix: Useful Procedures and Methods

General Procedures		
Name	**Parameters**	**Description**
`other-people-at-place`	`person:Person`, `place:Place`	Retrieves all people at `place`, except `person`.
`pick-random`	`lst:list`	Picks a random item in `lst`.
`random-neighbour`	`place:Place`	Picks a random adjacent room.

`split-list`	`pred:procedure`, `lst:list`	Equivalent to
		`(cons`
		`(filter pred lst)`
		`(filter (negate pred) lst))`

Place Methods: `(ask place method . args)`		
`neighbours`		Retrieves a list of adjacent rooms
`exits`		Retrieves a list of directions leading out of the room
`things`		Retrives a list of things inside, whether living or not, owned or unowned
`neighbour-towards`	`dir:direction`	Retrieves the neighbour in that direction, or `#f` if there is none
`accept-person?`	`person:Person`	Checks if `person` can enter
Living-thing Methods: `(ask living-thing method . args)`		
`health`		Retrieves current Health Points
Person Methods: `(ask person method . args)`		
`possessions`		Retrieves list of items owned
`say`	`sentence:list`	Says the sentence
`take` / `pick`	`item:Thing`, `item...`	Takes all specified items
`lose` / `drop`	`item:Thing`, `item...`	Loses all specified items
`use`	`item:Thing`, `...`	Uses the item. See the items table for more details
`move-to`	`room:Place`	Moves to `room`. Must be adjacent to current location
`go`	`dir:direction`	Moves in that direction
`act`		Called every clock cycle

Weapon Manuals	
`(ask weapon 'charging?)`	Check if a weapon is charging
`(ask weapon 'charge-turns)`	Check remaining charging time
`(ask weapon 'max-damage target)`	Determines max. damage on a target
`(ask weapon 'min-damage target)`	Determines min. damage on a target
Using Weapons:	
`(ask self 'use` **`melee-weapon`** `target)`	Use `melee-weapon` to attack selected `target` in the same room.

To be continued...