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

Mission 17:
Rookie Training

Issued: 17 October 2011

Due: 24 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

This mission follows from Discussion Group Exercises 6. Adventure games can be an interesting waste of time for many computer lovers. In order not to waste time, it is important to complete the exercises in DG5 and understand how the system works prior to attempting this mission.

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.

To better prepare everyone for the upcoming confrontation, Executive Officer Zi Han 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, Zi Han 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 Zi Han wishes all of you the best of luck, and motions you to proceed.

Task 1: (4 marks)

Our Simulation Room creates Service Bots using the following definition:

```
(define (make-service-bot name birthplace inertia)
  (let ((person (make-person name birthplace)))
    ( (message)
      (case message
        ((bot?) ( (self) #t))
        ((act)
         ( (self)
           ((get-method person 'act) self)
           (when (= (random inertia) 0)
             (ask self 'go (pick-random (ask (ask self 'place) 'exits))))))
        (else (get-method person message))))))

(define (make&install-bot name birthplace inertia)
  (let ((bot (make-service-bot name birthplace inertia)))
    (ask (register bot) 'install)
    (ask bot 'take (make&install-key-card (ask bot 'place)))
    bot))

(define bot1 (make&install-bot 'b1 bot1-place 2))
(define bot2 (make&install-bot 'b2 bot2-place 3))
```

After loading the simulation using `(start-mission #f)`, observe how `bot1` and `bot2` move around by clicking Next Action. Once you're familiarised with the system, call `(run-clock 10)` to simulate 10 rounds, each denoted by `---Tick---`.

(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 Zi Han

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 $4 \times 4 \times 4$ 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. Try not to wander aimlessly, as it will increase your chances of being killed by enemies. 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: (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: Every object that is of `obj-type` will answer `#t` to `(is-a object 'obj-type?)`. For example, you can check if a thing is a `key-card` if it answers `#t` to `(is-a thing 'key-card?)`.

We will simulate the generator-room using a `protected-room`. The `protected-room` is like a normal `place`, except that only a `person` holding a `key-card` will be allowed to enter.

Time to get yourself in that room. Implement `make-player` such that, in every turn, you will do all of the following:

- attack a service bot in the same room as you, if you have a charged weapon,
- pick up a key-card in the same room as you, and
- enter a protected-room if it is beside your current room and you have a key-card.

NOTE: As you will not be alone in the actual battle, you **MUST** check if it is a `service-bot` 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

```
(let ((lightsaber <find your own charged lightsaber>))
  (ask self 'use lightsaber bot1))
```

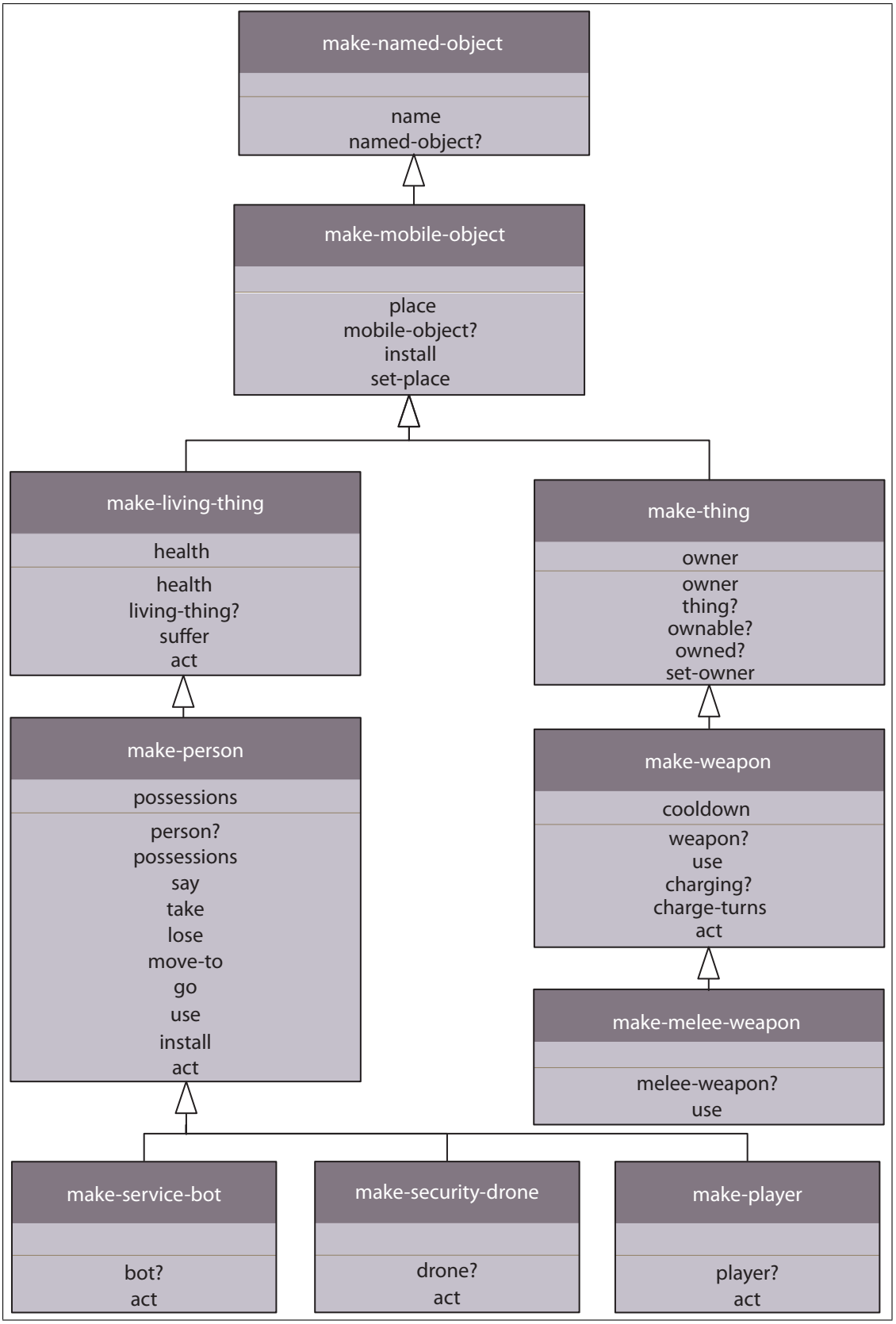
As the weapons you will be provided with are yet unknown, XO Zi Han suggests that you search your possessions for a usable weapon when you need to attack. All weapons will have a `weapon?` method that answers `true`, and you can determine if the weapon is ready by checking if it's not still `charging?`.

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

Submit your definitions of `make-player`. To test your strategy, uncomment `(start-mission)` at the bottom of the template. Remember to comment out the support code for Task 1.

HINT: You might want to look at `other-people-at-place` in the appendix.

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...