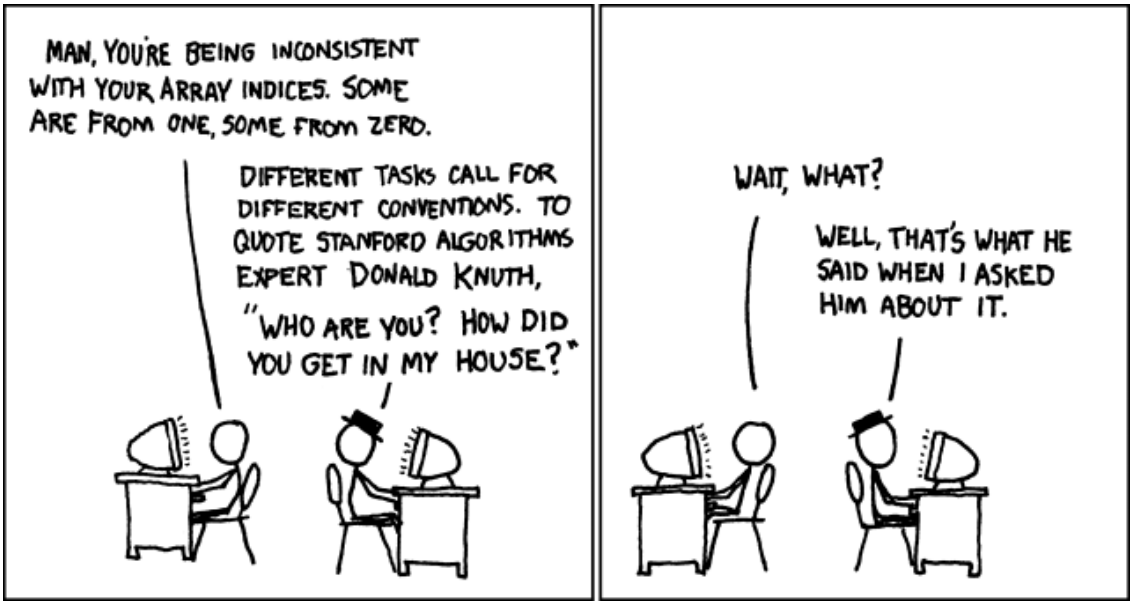
Assignment 10: Agents and boardgames

ETH Zurich

Hand-out: 27. November 2009
Due: 18. December 2009



Goals

- Test your understanding of agents and event driven programming.
- Implement more of the boardgame project.

1 Observing the temperature

This is an old exam question. Given is the class *TEMPERATURE_SENSOR*. Assume that there is a hardware component that updates the *value* query by calling the command *set_value* whenever the temperature changes.

To do

Implement the task with EiffelStudio. Download the files for *TEMPERATURE_SENSOR* and *HEATING_CONTROLLER* from
http://se.ethz.ch/teaching/2009-H/eprog-0001/exercises/observable_temp.zip.

1. Write a class *OBSERVABLE_TEMPERATURE_SENSOR* that inherits from the given class *TEMPERATURE_SENSOR*. Your implementation should allow observers to register agents. These agents are called with the new temperature value as an argument whenever the *set_value* feature of *OBSERVABLE_TEMPERATURE_ SENSOR* is invoked by the hardware. You may use any publish-subscribe mechanism shown in the course that satisfies the following conditions:

   - An observer can observe any number of publishers.
   - A publisher does not know its observers, but only the agents to call when the temperature changes.

2. Complete the implementation of the observer class *HEATING_CONTROLLER* using your implementation of *OBSERVABLE_TEMPERATURE_SENSOR*. There should be a feature *set_sensor* that does three things:

   (a) it should set *sensor* to the sensor passed as an argument

   (b) it should register the procedure *adjust_heating* to be called whenever a change to the value of the new sensor happens

   (c) if the procedure *adjust_heating* in class *HEATING_CONTROLLER* was already subscribed to the previous sensor, then it should unsubscribe it from this old sensor.

   You may add new features or redefine features from class *ANY* to complete the implementation.

To hand in

Submit your answers to your assistant.

2 Reviewing reference passing

This is a pen and paper exercise.

To do

Analyze the following code and write down on paper the result of the four print instructions (in order).

```
class
  APPLICATION

create
  make

feature {NONE} -- Initialization

  make
      -- Run application.
    local
      obj_1, obj_2: STRING
    do
      create obj_1.make_from_string ("test")
      obj_2 := obj_1
      io.new_line
      print (obj_2)
      obj_2.append_string ("_foo")
      io.new_line
      print (obj_1)
      io.new_line
      do_bar (obj_1)
      io.new_line
      print (obj_1)
    end

feature -- Basic Operations

  do_bar (obj_arg: STRING)
      -- Argument passing behavior test.
    local
      obj_3: STRING
    do
      create obj_3.make_from_string ("_loc")
      obj_3 := obj_arg
      obj_arg.append ("_again?")
      print (obj_3)
    end
end
```

To hand in

Submit your answers to your assistant.

3 The final project: programming a boardgame (part 4)

To do

In this task, that constitutes the final project, you will extend the implementation of the boardgame system. Here is the previous, already implemented specification.

Current game specification

The idea is to program a prototype of a board-game[1]. It comes with a board, divided into 40 squares, a pair of six-sided dice, and can accommodate 2 to 6 players. It works like this: all players start from the first square. One at the time, players take a turn. This includes rolling the dice and advance their respective tokens on the board. When all players are done with their turn, it is called a round. Players have now money. Each player starts with an amount of 7 CHF. There are the following special squares:

- Squares 5, 15, 25, 35 are "Bad Investment"squares. The effect on a player that lands on them is to subtract 5 CHF from that player's current owned amount of money, if the player can afford it, otherwise as much as it is possible is subtracted. The amount of money a player owns cannot go below zero.
- Squares 10, 20, 30, 40 are "Lottery Win"squares. The effect on a player that lands on them is too add 10 CHF to that player's amount.

The winner will be the player that has more money after the first player advances beyond the 40th square.

New game specification

You can either use our pre-defined specification detailed below (Snakes and Ladders), or provide your own. In the latter case, please ask your assistant for approval before starting to code. At the beginning of the spring semester you will have a chance to showcase your game if you wish to do so.

Snakes and Ladders

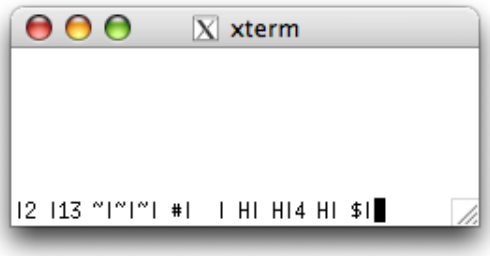
The idea is to add more special squares and a console-based GUI to the implementation as follows:

- Squares 2, 3, 4, 12, 13, 14, 22, 23, 24, 32, 33, 34 are "Snakes"squares. Each snake is represented by three snake characters (see below) and covers three consecutive squares. The first square is the tail, the second is the body and the third is the head. The effect for a player landing on a snake square is the following: when a player lands on the head of a snake (that is, on square 4, 14, 24 or 34), he is immediately sent back to the tail of the snake (that is, respectively, square 2, 12, 22, 32). If the player lands on the snake's body (squares 3, 13, 23, 33 respectively) he wins 5 CHF. Nothing happens when the player lands on the snake's tail.
- Squares 7, 8, 9, 17, 18, 19, 27, 28, 29, 37, 38, 39 are "Ladders "squares. Each ladder is represented by three ladder characters (see below) and covers three consecutive squares. The first square is the bottom, the second is the body and the third is the top. The effect for a player landing on a ladder square is the following: when a player lands on the

[1] We draw inspiration from a case study in the excellent book by Craig Larman: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)

bottom of a ladder (that is, on square 7, 17, 27 or 37), he is immediately sent forward to the top of the ladder (that is, respectively, square 9, 19, 29, 39). If the player lands on the ladder's body (squares 8, 18, 28, 38 respectively) he loses 5 CHF. Nothing happens when the player lands on the top of the ladder.

- Keep a game trace by drawing the board on the command line or by writing a text file, or both. You may use the following characters: "~"for snakes, "H"for ladders, "$"for lottery, "#"for bad investments. A partial board representation, purely indicative, is shown in the following figure. Pipe characters "|"are used to separate squares, and numbers to identify



players. Notice that more players can be on a square.

- The winner will be the player with most money after the first player advances exactly to the 40th square. If a player happens to go beyond the 40th square, then it has to continue backwards. For example, if a player is on square 39 and rolls a 4, it will end up on square 37. Square 37 is a ladder bottom, so the player will be pushed forward to square 39 again. Ties (multiple winners) are possible.

More sample specifications

Here we provide some suggestions for additions to the game:

- Add a more sophisticated GUI to the basic game to make it more appealing. Use the EiffelVision 2 library.
- Extend the game to a Monopoly clone, with or without a GUI.
- Provide a way for the players to battle against each other when they occupy the same square. The duel can be solved by rolling a die and can result in a gain or loss of money for the players involved.
- Program the famous game "Mensch, ärgere dich nicht". Similar games are Ludo and Pachisi.
- Unleash your imagination and provide your own extension! Remember, before starting to code, to ask your assistant for approval.

Hints

- The classes provided in the solution for assignments 6 or 8 can provide a reasonable starting point, if you wish. You can download them at
http://se.ethz.ch/teaching/2009-H/eprog-0001/exercises/boardgame.zip and
http://se.ethz.ch/teaching/2009-H/eprog-0001/exercises/boardgame2.zip.

- In case you chose the snakes and ladders sample specification, note that to be able to move a player backwards because of the snakes, you may have to add a reference to the previous square in addition to the reference to the next square.
- The EiffelVision 2 documentation can be found at http://docs.eiffel.com/book/solutions/eiffelvision-2.

To hand in

Submit the code of your classes.