Pràctica 1, agent ascensor

Martí Amengual Torrens <marti.amengual.torrens@gmail.com>,
Pau Rul·lan Ferragut <paurullan@gmail.com>

13 de juliol de 2012

Resum

La instal·lació d'un ascensor que no respongui sols als estímuls mecànics sinó que decideixi un comportament òptim és una bona iniciació a l'estudi dels agents intel·ligents. En aquesta pràctica es duu a terme l'avaluació de les clàusules que determinen com un ascensor es comportarà de manera que els seus ocupants tinguin el millor servei possible. Es proposa una implementació en un llenguatge de caràcter general i una interfície gràfica amigable.

Índex

1 Introducció

Consideris el cas d'un ascensor senzill que pot percebre la següent informació del seu entorn:

- En quin pis es troba aturat.
- A quins pisos volen anar els ocupants.
- En quins pisos hi ha persones que volen entrar a l'ascensor. Es considera que hi ha dos botons per cridar l'ascensor, un de pujada i un de baixada.
- L'estat de la porta (oberta o tancada).

A més, l'ascensor es capaç de realitzar les següents accions:

1. Pujar un pis, a no ser que estigui al darrer pis.
2. Baixar un pis, a no ser que estigui a la planta baixa.
3. Obrir la porta.
4. Tancar la porta.
5. Esperar δ segons per tal que els usuaris puguin entrar i sortir.

Amb aquestes dades s'ha de dissenyar un agent capaç de controlar l'ascensor de forma eficient. No serà eficient, per exemple, canviar el sentit de l'ascensor quan estigui pujant si encara hi ha algú adins que vol pujar a un pis superior o si hi ha algú a fora que hagi sol·licitat entrar a l'ascensor des d'algun pis superior.

2 Avaluació del problema

Per tal de resoldre la pràctica hem dissenyat un agent estímul-resposta (d'ara endavant *agent ER*) amb estat intern. L'agent rep unes percepcions de l'ambient, on a partir d'aquets estímuls i el seu estat intern és capaç d'escollir l'acció més eficient.

Per tal de que l'agent ascensor sigui eficient no ha estat suficient resoldre'l a través d'un simple agent ER sinó que hem hagut d'afegir un estat intern. Aquesta informació ajudarà a l'agent a decidir quina acció toca executar per tal de satisfer les necessitats dels usuaris. A més, amb l'estat intern, se'ns permetrà fer-ho minimitzar els canvis de sentit per així proporcionar la millor experiència possible als usuaris.

A continuació es descriuen amb més detall cada una d'aquestes característiques enunciades: percepcions, accions, estat intern, estat de l'ambient.

2.1 Percepcions

Les percepcions de l'agent ascensor són les senyals o estímuls que percep de l'ambient on treballa. El nostre agent ascensor podrà percebre:

- El pis on es troba situat.
- Una llista de les peticions de sortida de les persones que es troben a l'interior.
- Una llista de les peticions d'entrada per tal de pujar de pis.
- Una llista de les peticions d'entrada per tal de baixar de pis.
- Si la porta està oberta o tancada.

Si analitzam les percepcions podem deduir que l'ascensor només podrà canviar de pis mentre la porta estigui tancada. Si la porta esta oberta l'agent ascensor no podrà pujar ni baixar de pis. Com que en tot moment l'agent sap en quin pis esta situat podrà deduir fàcilment quin tipus de peticions té dels pisos superiors i inferiors.

El problema ve de que aquestes percepcions no proporcionen suficient informació com per decidir l'acció i necessitam saber les darreres accions i moviments de l'ascensor. Per aquest motiu s'utilitza un estat intern que es descriu en el proper subapartat.

2.2 Estat intern

Com s'ha mencionat abans, únicament amb les percepcions no es suficient decidir quina acció convé aplicar l'agent ja que podrà ser diferent depenent de si l'ascensor es trobà en sentit de pujada o baixada. Per aquest motiu hem definit un estat intern caracteritzat pel sentit de l'ascensor (pujada o baixada) i per l'última acció realitzada.

No ens es suficient que l'estat intern sigui exclusivament la darrera acció realitzada per l'agent perquè aquesta no es pot determinar el sentit del mateix. Per exemple, en el cas que en algun determinat pis l'ascensor s'ha de parar per obrir la porta, esperar δ segons i tancar la porta, l'última acció de tancar la porta, no ens determina si l'ascensor estava baixant o pujant de pis. Com que no ho sabem l'ascensor podria actuar de manera ineficient. Per tant, hem de saber en tot moment amb quin sentit es troba l'ascensor.

2.3 Accions

Explicant les percepcions i l'estat intern de l'agent gairebé ja hem definit les accions de l'ascensor. En tot cas les accions que pot escollir l'agent són aquestes cinc:

1. Pujar un pis, a no ser que l'ascensor estigui al darrer pis.
2. Baixar un pis, a no ser que l'ascensor estigui a la planta baixa.
3. Obrir la porta.
4. Tancar la porta.
5. Esperar δ segons per tal de que els usuaris de l'ascensor puguin sortir i entrar de l'ascensor.

2.4 Estat de l'ambient

L'ambient de l'agent ascensor estarà definit per diferents estats. Aquests estats ho determinaran les diferents combinacions de l'estat de l'ascensor i les peticions que realitzin els usuaris. L'estat de l'ascensor vendrà determinat per el pis on es troba l'ascensor i per com es trobin les portes d'aquest (obertes o tancades). Les peticions que realitzin els usuaris per sortir o entrar a l'ascensor complementaran el seu estat i d'aquesta manera definiran els diferents possibles estats de l'ambient.

3 Regles de l'agent

Per a definir les regles de l'agent ens ajudarem de les següents funcions/variables:

- pujada / baixada: representarà el sentit de l'ascensor a través de l'estat intern.
- esperat_δ_segons: representarà si l'última acció ha estat esperar δ segons per tal de que els usuaris puguin entrar i sortir de l'ascensor.
- obert / tancat: representarà l'estat de la porta.
- peticio_sortida(actual): representarà si hi ha una petició de sortir de l'ascensor en el pis actual.
- peticio_pujada(actual): representarà si hi ha una petició de pujar en el pis actual.
- peticio_baixada(actual): representarà si hi ha una petició de baixar en el pis actual.
- peticions_inferiors: representarà si existeixen peticions als pisos inferiors, tant siguin de sortida, pujada i baixada.
- peticions_superiors: representarà si existeixen peticions als pisos superiors, tant siguin de sortida, pujada i baixada.

Les regles de l'agent que hem dissenyat són les següents:

1. obert i esperat_δ_segons $\Rightarrow$ tancar
2. obert $\Rightarrow$ esperar_delta_segons
3. tancat i peticio_sortida(actual) $\Rightarrow$ obrir
4. tancat i pujada i peticio_pujada(actual) $\Rightarrow$ obrir
5. tancat i baixada i peticio_baixada(actual) $\Rightarrow$ obrir
6. tancat i pujada i peticions_superiors $\Rightarrow$ pujar
7. tancat i baixada i peticions_inferiors $\Rightarrow$ baixar
8. tancat i pujada i peticions_inferiors $\Rightarrow$ baixar
9. tancat i baixada i peticions_superiors $\Rightarrow$ pujar
10. tancat i pujada i peticio_baixada(actual) $\Rightarrow$ obrir
11. tancat i baixada i peticio_pujada(actual) $\Rightarrow$ obrir

4 Funcionament dels moderns i actuals ascensors

Actualment podem diferenciar dos tipus d'ascensor segons l'algorisme de funcionament, un amb un sol botó de cridada exterior a cada pis i un amb dos botons de cridada exterior a cada pis; un botó per ascendir de pis i un botó per descendir de pis.

Com que l'ascensor dissenyat a la pràctica es a través de dos botons de cridada exterior per cada pis, anem a centrar-nos en aquest tipus. L'algorisme que utilitzen els ascensors reals segueixen els mateixos principis utilitzats a la pràctica per tal de gestionar les peticions internes i externes de l'ascensor. A continuació s'exposa el seu mode de funcionament i així podem comprovar que és el mateix que el que s'utilitza a la pràctica.

Funcionament en pujada: l'ascensor es va aturant a tots els pisos marcats des de l'interior i també en els pisos on hi ha una petició marcada de pujada, però no en els que només estigui marcat la petició de baixada. Un cop arribat al pis més elevat on hi hagi hagut peticions de pujada es canvia de sentit per tal de funcionar en mode baixada.

Funcionament en baixada: l'ascensor es va aturant a tots els pisos marcats des de l'interior i també en els pisos on hi ha una petició marcada de baixada, però no en els que només estigui marcat la petició de pujada.

Un cop arribat al pis més inferior on hi hagi hagut alguna petició es canvia de sentit per tal de funcionar en mode pujada.

A més, també podem trobar-nos ascensors més complexos formats per més d'una unitat. Aquests ascensors tenen un sistema de coordinació. Aquest sistema de coordinació processa l'estat de cada unitat de l'ascensor decidint en tot moment quina d'ella es més avantatjosa per tal de servir cada petició disminuint així el temps d'espera per parts dels usuaris.

Com hem pogut observar, els funcionament dels ascensors reals és el mateix que l'utilitzat a la pràctica, treballant d'una manera intel·ligent i eficient.

5 Manual d'usuari

Un cop arrancada l'aplicació se'ns presentarà una finestra gràfica amb un conjunt d'accions per dur a terme:

1. Anar a un pis des de l'interior de l'ascensor
2. Sol·litar pujar des de la planta baixa fins a l'antepenúltim pis
3. Sol·litar baixar des del primer fins al darrer pis
4. Iniciar una demostració de comportament
5. Tancar l'aplicació

A la figura 1 s'assanyalen els botons distints que es corresponen a les accions possibles.

La figura 2 representa l'estat en funcionament del programa. Es pot veure una interacció, com tenim varis estímuls en marxa i una animació a l'ascensor que ens indica el seu estat intern.

S'ha adjuntat un petit conjunt de demostracions per facilitar l'avaluació de la interfície gràfica, automatitzant jocs de prova com un ascensor que puja varis pisos o un que ha de resoldre una situació de vàries peticions. Aquestes demostracions són semblants als casos d'ús que es varen usar per verificar el comportament lògic de l'ascensor. D'aquesta manera l'usuari pot ràpidament confirmar les possibilitats gràfiques i estudiar apartats com les animacions o els temps $delta$ d'apertura de portes.

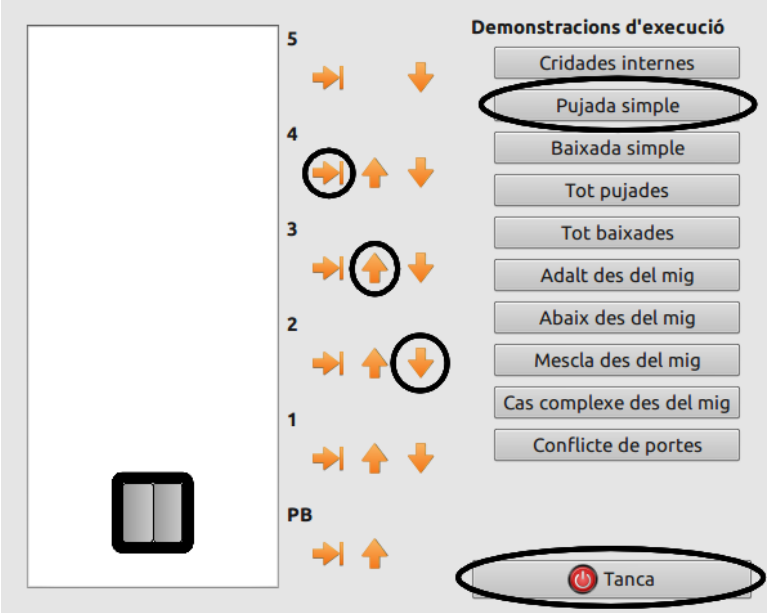

Figura 1: *Captura de pantalla de la interfície gràfica*

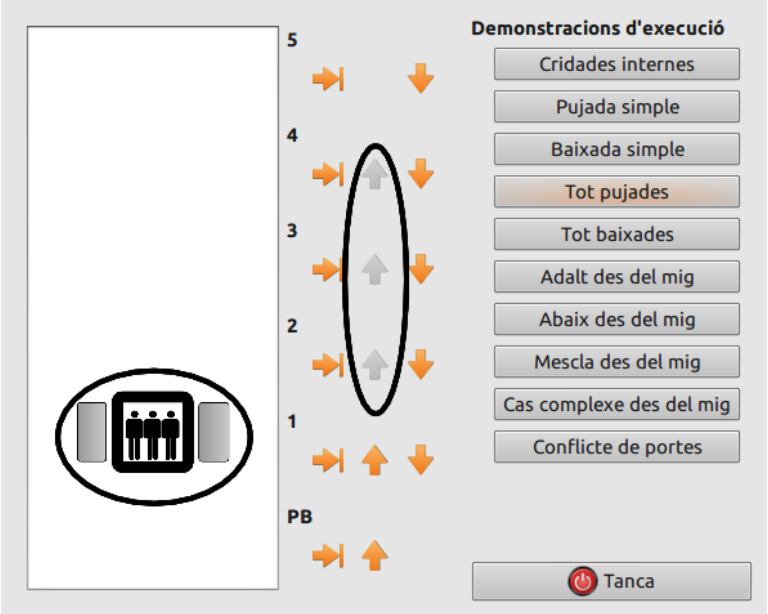

Figura 2: *Captura de pantalla de la interfície gràfica*

6 Conclusions

La pràctica ha sigut tot un repte sobretot en l'aspecte de la presentació. Malgrat la traducció de les clàusules lògiques i l'esquema de comportament no va resultar massa complicada la creació d'una interfície gràfica atractiva ens va portar més hores de les esperades. De tota manera esperam haver guanyat flexibilitat en aquest aspecte de tal manera que en futures pràctiques poguem lliurar resultats del mateix nivell de qualitat amb molt manco temps.

A Implementació

Per la implementació de la pràctica s'ha optat pel llenguatge de propòsit general Python i les biblioteques gràfiques QT mitjançant els *bindings* PyQt4. El disseny de la pràctica s'ha fet de tal manera que tenim quatre fitxers distints, quadasqun amb els seus objectius concrets:

elevator.py Cos lògic de l'agent ascensor

test.py Jocs de proves de l'agent ascensor

igu.py Classes de la interfície gràfica

main.py Programa principal

Agent ascensor - elevator.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from __future__ import print_function

import PyQt4.QtCore

__version__ = "0.0.1"

class Elevator(PyQt4.QtCore.QObject):

    def __init__(self, n_floors=5, initial_floor=0):
        """
        Les accions pending són les que falten per processar. Les new són les
        que la interfície ha llançat i encara s'han d'afegir a la llista
        pending. El motiu de separació d'aquests conjunts és per no tenir
        problemes de concurrència i simular que l'ascensor no pot canviar
        d'opinió en segons quins moments

        El format de les accions són (tipus, pis objectiu), on els tipus són
        "up", "down", "open".

        open_floor_log és una cua on afegim els pisos que hem obert la
        porta. Molt útil per fer els unittest.
        """
        super(Elevator, self).__init__()

        self.pending = set()
        self.new = set()

        self.direction = "up"

        assert(n_floors > 0 )
        assert(initial_floor < n_floors)

        self.floor = initial_floor
        self.n_floors = n_floors
        self.bottom_floor = 0
        self.top_floor = n_floors-1

        self.open_floor_log = []
        self.last = "init"


    def emit_update(self):
        self.emit(PyQt4.QtCore.SIGNAL("update()"))

```

```
48
49      def update_pending(self):
50          self.pending.update(self.new)
51          self.new = set()
52
53      def add_list(self, cases):
54          """La llista d'estímuls d'entrada s'afegeixen al conjunt d'estímuls
55          pendents a actualitzar. Necessari pels unittest."""
56          self.new.update(cases)
57
58      @property
59      def opposite(self):
60          if self.direction == "up":
61              return "down"
62          else:
63              return "up"
64
65      def flip_direction(self):
66          self.direction = self.opposite
67
68
69      def present(self, action):
70          return action in self.pending or action in self.new
71
72      def next(self):
73          """
74          Funció clau de l'algorisme de l'ascensor que indica l'acció a fer.
75          obrir, pujar, baixar o esperar i obrir per canviar de sentit. (open,
76          up, down, wait, open_switch).
77
78          Important que en els casos que obrim cal afegir a la llista de log de
79          portes obertes.
80          """
81          if not self.pending:
82              return "wait"
83          if (self.floor, "open") in self.pending or (self.floor, self.direction) in self.pe
84              return self.calc_open()
85          if self.direction == "up":
86              return self.calc_up()
87          elif self.direction == "down":
88              return self.calc_down()
89
90      def calc_open(self):
91          """
92          L'ascensor té una petició en el pis acutal, per tan s'ha de saber si
93          s'ha d'obrir la porta, si s'ha d'obrir i canviar de sentit o si ha de
94          seguir amb la direcció que anava.
95          """
96
```

```
97          upper_floors = range(self.floor+1, self.n_floors)
98          lower_floors = range(self.floor-1, self.bottom_floor-1, -1)
99
100         if (self.floor, self.direction) in self.pending:
101             return "open"
102         if (self.floor, "open") in self.pending:
103             if (self.floor, self.opposite) in self.pending:
104                 if self.direction == "up":
105                     for i in upper_floors:
106                         if (i, "open") in self.pending or \
107                            (i, "down") in self.pending or \
108                            (i, "up") in self.pending:
109                             return "open"
110                     return "open_switch"
111                 else:
112                     for i in lower_floors:
113                         if (i, "open") in self.pending or \
114                            (i, "down") in self.pending or \
115                            (i, "up") in self.pending:
116                             return "open"
117                       return "open_switch"
118             else:
119                 return "open"
120         else: #Nomes tenim l'opposite seleccionat
121             if self.direction == "up":
122                 return self.calc_up()
123             else:
124                 return self.calc_down()
125
126     def calc_up(self):
127         """
128         L'ascensor està pujant, càlcul de l'acció a fer. Casos seguir pujant,
129         pujar fins a poder canviar de sentit (més proper), canviar sentit i
130         baixar fins tenir algú per baixar (proper), baixar fins recollir algú
131         per pujar (més llunyà).
132         """
133         upper_floors = range(self.floor+1, self.n_floors)
134         lower_floors = range(self.floor-1, self.bottom_floor-1, -1)
135         up_to_floors = range(self.bottom_floor, self.floor)
136         for i in upper_floors:
137             if (i, "open") in self.pending or (i, "up") in self.pending:
138                 return "up"
139         for i in upper_floors:
140             if (i, "down") in self.pending:
141                 return "up"
142         # canvi de sentit
143         if (self.floor, "down") in self.pending:
144             return "open_switch"
145         # casos de trucades des de més avall
```

```
146             for i in lower_floors:
147                 if (i, "down") in self.pending:
148                     return "down"
149             for i in up_to_floors:
150                 if (i, "up") in self.pending or (i, "open") in self.pending:
151                     return "down"
152 
153 
154     def calc_down(self):
155         """
156         L'ascensor està baixant, càlcul de l'acció. Casos de baixada: seguir
157         baixant, baixar fins recollir un de pujada (més llunyà), pujar per
158         recollir un de pujada (més proper), pujar fins recollir un de baixar
159         (més llunyà),
160         """
161         upper_floors = range(self.floor+1, self.n_floors)
162         lower_floors = range(self.floor-1, self.bottom_floor-1, -1)
163         up_to_floors = range(self.bottom_floor-1, self.floor-1)
164         down_to_floors = range(self.n_floors, self.floor, -1)
165 
166         for i in lower_floors:
167             if (i, "open") in self.pending or (i, "down") in self.pending:
168                 return "down"
169         for i in lower_floors:
170             if (i, "up") in self.pending:
171                 return "down"
172         # canvi de sentit
173         if (self.floor, "up") in self.pending:
174             return "open_switch"
175         for i in upper_floors:
176             if (i, "up") in self.pending:
177                 return "up"
178         for i in down_to_floors:
179             if (i, "down") in self.pending or (i, "open") in self.pending:
180                 return "up"
181 
182     def do(self):
183         """Obté el valor de la següent acció, l'executa i la lleva de la llista
184         d'estímuls pendents a respondre. En cas que es processi una acció en el
185         pis local cal confirmar que llevam els possibles (local, "open").
186         """
187         if self.last == "init":
188             action = "wait"
189             return action
190         if self.last == "open":
191             action = "close"
192             return action
193         if not self.pending:
194             return "wait"
```

```
195
196             action = self.next()
197             if action == "open_switch":
198                 action = "open"
199                 self.flip_direction()
200             if action == "open":
201                 self.open_floor_log.append(self.floor)
202                 open_door = (self.floor, "open")
203                 get_person = (self.floor, self.direction)
204                 if open_door in self.pending:
205                     self.pending.remove(open_door)
206                 if get_person in self.pending:
207                     self.pending.remove(get_person)
208             elif action == "up":
209                 self.floor += 1
210                 assert(self.floor < self.n_floors)
211                 self.direction = "up"
212             elif action == "down":
213                 self.floor -= 1
214                 assert(self.floor >= 0)
215                 self.direction = "down"
216             return action
217
218         def floor_actions(self):
219             """Calcula a una tupla els pisos a que s'ha d'obrir la porta en
220             ordre. Útil pels unittest. Pot incloure repeticions com «(1, 2, 3, 2,
221             1)». Afecta a l'estat de l'ascensor.
222             """
223             self.update_pending()
224             while self.pending:
225                 self.last = self.do()
226             return tuple(self.open_floor_log)
227
228         def action(self):
229             self.update_pending()
230             self.last = self.do()
231             self.emit_update()
232
233
234         def __str__(self):
235             s = "new: %s \n" % self.new
236             s += "pending: %s\n" % self.pending
237             s += "pis: %d\n" % self.floor
238             s += "direcció: %s\n" % self.direction
239             s += "darrera: %s" % self.last
240             return s
241
242
243     if __name__ == '__main__':
```

```
244     pass
```

Jocs de proves lògics - test.py

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4   """
5
6   Unittest de la pràctica de l'ascensor. La intenció és poder tenir tot el
7   conjunt de funcionalitats de la pràctica sense haver de fer cap interfície
8   gràfica. Amb els unittest també podrem confirmar casos extrems, com d'ascensors
9   que passen buits per anar a buscar més gent a pisos més avall, sense passar
10  pena de rompre el codi.
11
12  La codificació dels casos consisteix en dos vectors, un dels estímuls externs i
13  un que simbolitza els pisos on s'obren les portes i per ordre. Cal fixar-se que
14  el vector d'estímuls es refereix a l'estat en un moment determinat i deixarem
15  de banda temes de concurrència. Aquest vector també suposa que no tenim accions
16  repetides. També és important veure que l'ordre d'apertura de les portes si és
17  important i poden haver pisos repetits.
18
19  Usarem els símbols «d», per l'apertura (altgr+i), «b» per baixar (altgr+u) i «a»
20  per pujar (altgr+shift+u).
21  """
22
23  from __future__ import print_function
24
25  import unittest
26
27  import elevator
28
29  __version__ = "0.0.1"
30
31  class TestElevator(unittest.TestCase):
32      """Classe pròpia pels tests de l'ascensor"""
33
34      def setUp(self):
35          """ Creació de l'ascensor. Determinam 6 pisos i que l'ascensor comenci
36          al zero."""
37          self.elevator = elevator.Elevator(6, 0)
38
39      def test_one(self):
40          """
41          Cas senzill de tot trucades internes.
42          (0, 1d, 2d, 3d, 4d, 5d) -> (1, 2, 3, 4, 5)
43          """
44          cases = [(1,"open"), (2,"open"), (3,"open"), (4,"open"), (5,"open")]
```

```
45          results = (1, 2, 3, 4, 5)
46          self.elevator.add_list(cases)
47          self.assertEqual(self.elevator.floor_actions(), results)
48
49      def test_two(self):
50          """
51          Cas senzill de pujar
52          (1, 2a, 3d) -> (2, 3)
53          """
54          cases = [(2, "up"), (3, "open")]
55          results = (2, 3)
56          self.elevator.floor = 1
57          self.elevator.add_list(cases)
58          self.assertEqual(self.elevator.floor_actions(), results)
59
60      def test_three(self):
61          """
62          Cas senzill de baixada
63          (5, 4b, 3d) -> (4, 3)
64          """
65          cases = [(4, "down"), (3, "open")]
66          results = (4, 3)
67          self.elevator.floor = 5
68          self.elevator.add_list(cases)
69          self.assertEqual(self.elevator.floor_actions(), results)
70
71      def test_four(self):
72          """
73          Cas senzill de pujar
74          (4, 3a) -> (3)
75          """
76          cases = [(3, "open")]
77          results = (3,)
78          self.elevator.floor = 4
79          self.elevator.add_list(cases)
80          self.assertEqual(self.elevator.floor_actions(), results)
81
82      def test_five(self):
83          """
84          Cas senzill de pujar
85          (4, 2a) -> (2)
86          """
87          cases = [(2, "open")]
88          results = (2,)
89          self.elevator.floor = 4
90          self.elevator.add_list(cases)
91          self.assertEqual(self.elevator.floor_actions(), results)
92
93
```

```
94      def test_up_all(self):
95          """
96          Cas senzill de tot pujades.
97          (0, 1a, 2a, 3a, 4a, 5a) -> (1, 2, 3, 4, 5)
98          """
99          cases = [(1, "up"), (2, "up"), (3, "up"), (4, "up"), (5, "up")]
100         results = (1, 2, 3, 4, 5)
101         self.elevator.add_list(cases)
102         self.assertEqual(self.elevator.floor_actions(), results)
103
104     def test_down_all(self):
105         """
106         Cas senzill de tot baixades. Cal fixar-se que l'ascensor amb aquest
107         conjunt d'estímuls anirà primer de tot al pis d'adalt i llavors
108         començarà a baixar.
109         (0, 1b, 2b, 3b, 4b, 5b) -> (5, 4, 3, 2, 1)
110         """
111         cases = [(1, "down"), (2, "down"), (3, "down"), (4, "down"), (5,"down")]
112         results = (5, 4, 3, 2, 1)
113         self.elevator.add_list(cases)
114         self.assertEqual(self.elevator.floor_actions(), results)
115
116
117     def test_up_from_middle(self):
118         """
119         Cas particular on l'ascensor comença del centre i volem fer una pujada.
120         (3, 2a, 4a) -> (4, 2)
121         """
122         cases = [(2, "up"), (4, "up")]
123         results = (4, 2)
124         self.elevator.floor = 3
125         self.elevator.direction = "up"
126         self.elevator.add_list(cases)
127         self.assertEqual(self.elevator.floor_actions(), results)
128
129
130     def test_down_from_middle(self):
131         """
132         Cas particular on l'ascensor comença del centre i volem fer una
133         baixada. Lo interessant és que abans de començar a baixar pujarà un
134         pis.
135         (3, 2b, 4b) -> (4, 2)
136         """
137         cases = [(2, "down"), (4, "down")]
138         results = (4, 2)
139         self.elevator.floor = 3
140         self.elevator.direction = "up"
141         self.elevator.add_list(cases)
142         self.assertEqual(self.elevator.floor_actions(), results)
```

```
143
144
145     def test_mixed_up(self):
146         """
147         Primer cas complexe de pujades i baixades.
148         (0 1a 2d 3b 4a 5d) -> (1, 2, 4, 5, 3)
149         """
150         cases = [(1, "up"), (2, "open"), (3, "down"), (4, "up"), (5, "open")]
151         results = (1, 2, 4, 5, 3)
152         self.elevator.add_list(cases)
153         self.assertEqual(self.elevator.floor_actions(), results)
154
155
156     def test_mixed_up_from_middle(self):
157         """
158         Primer cas complexe de pujades i baixades però amb l'ascensor centrat.
159         (3 4a 5b 2b 1a) -> (4, 5, 2, 1)
160         """
161         cases = [(1, "up"), (2, "down"), (4, "up"), (5, "down")]
162         results = (4, 5, 2, 1)
163         self.elevator.floor = 3
164         self.elevator.add_list(cases)
165         self.assertEqual(self.elevator.floor_actions(), results)
166
167
168     def test_door_conflict(self):
169         """
170         Cas complexe on a un pis tenim vàries cridades i cal resoldre el servei
171         sense obrir la porta dos pics seguits.
172         (2, 3d, 3a, 3b, 4d, 1d) -> (3, 4, 3, 1)
173         """
174         cases = [(3, "open"), (3, "up"), (3, "down"), (4, "open"), (1, "open")]
175         results = (3, 4, 3, 1)
176         self.elevator.floor = 2
177         self.elevator.add_list(cases)
178         self.assertEqual(self.elevator.floor_actions(), results)
179
180
181 if __name__ == '__main__':
182     unittest.main()
```

Interfície gràfica - igu.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  __version__ = "0.0.1"
5
```

```
6  import elevator
7
8  from PyQt4 import *
9  from PyQt4.QtCore import *
10 from PyQt4.QtGui import *
11
12 class MainWindow(QWidget):
13     """Classe del widget principal. No cal fer una finestra principal perquè no
14     tenim menus ni semblants"""
15
16     def __init__(self):
17         super(MainWindow, self).__init__()
18         self.elev = elevator.Elevator(6, 0)
19         self.connect(self.elev, SIGNAL("update()"), self.update)
20
21         self.elevator = ElevatorWidget(self.elev)
22         self.demo = DemoWidget(self.elev)
23
24         exit_button = QPushButton("Tanca")
25         exit_button.setIcon(QIcon("img/exit.svg"))
26         self.connect(exit_button, SIGNAL("clicked()"), SLOT("close()"))
27
28         control_pane = QVBoxLayout()
29         control_pane.addWidget(self.demo)
30         control_pane.addStretch()
31         control_pane.addWidget(exit_button)
32
33         main_pane = QHBoxLayout()
34         main_pane.addWidget(self.elevator)
35         main_pane.addStretch()
36         main_pane.addLayout(control_pane)
37         self.setLayout(main_pane)
38
39         self.timer = QTimer()
40         self.connect(self.timer, SIGNAL("timeout()"), self.elev.action)
41         self.timer.start(1000)
42
43
44     def update(self):
45         self.elevator.update()
46
47
48 class FloorWidget(QGroupBox):
49     """Classe del widget de gestió de l'ascensor. Té la lògica interna i desa
50     les dades de l'ascensor"""
51
52     def __init__(self, floor, elev):
53         super(FloorWidget, self).__init__()
54         self.floor = floor
```

```
55          self.elevator = elev
56          if self.floor == 0:
57              self.setTitle(QString("PB"))
58          else:
59              self.setTitle(QString(str(self.floor)))
60
61          self.open = OpenButton(self.floor, elev)
62          self.up = UpButton(self.floor, elev)
63          self.down = DownButton(self.floor, elev)
64
65          floor_pane = QHBoxLayout()
66          floor_pane.addWidget(self.open)
67          floor_pane.addWidget(self.up)
68          floor_pane.addWidget(self.down)
69          self.setLayout(floor_pane)
70
71      def update(self):
72          self.open.update()
73          self.up.update()
74          self.down.update()
75
76      def remove_down(self):
77          """Desactiva la possibilitat d'anar per amunt. Necessari pel pis
78          d'adalt del tot"""
79          self.down.setEnabled(False)
80          self.down.never_on = True
81          self.down.setIcon(QIcon("img/empty.svg"))
82
83      def remove_up(self):
84          """Desactiva la possibilitat d'anar per avall. Necessari pel pis
85          d'abaix"""
86          self.up.setEnabled(False)
87          self.up.never_on = True
88          self.up.setIcon(QIcon("img/empty.svg"))
89
90  class ElevatorActionButton(QToolButton):
91      """Botó de l'acció per introduir estímuls a l'ascensor. Està fet de tal
92      manera que sigui molt senzill per generar subclasses d'obrir, pujar i
93      baixar.
94
95      El «never_on» serveix per posar els pisos inferior i superior desconnectats
96      per no poder pujar i baixar més."""
97
98      def __init__(self, floor, elev, action):
99          super(ElevatorActionButton, self).__init__()
100         self.setAutoRaise(True)
101         self.floor = floor
102         self.action = action
103         self.elevator = elev
```

```
104             self.connect(self, SIGNAL("clicked()"), self.do)
105             filename = "img/%s.svg" % action
106             self.setIcon(QIcon(filename))
107             self.never_on = False
108 
109         def do(self):
110             self.elevator.add_list([(self.floor, self.action)])
111             self.setEnabled(False)
112 
113         def update(self):
114             """Actualitza l'estat del botó segons si l'acció està pendent a processar"""
115             super(ElevatorActionButton, self).update()
116             if self.never_on:
117                 return
118             present = self.elevator.present((self.floor, self.action))
119             self.setEnabled(not present)
120 
121 
122     class OpenButton(ElevatorActionButton):
123 
124         def __init__(self, floor, elev):
125             super(OpenButton, self).__init__(floor, elev, "open")
126 
127     class UpButton(ElevatorActionButton):
128 
129         def __init__(self, floor, elev):
130             super(UpButton, self).__init__(floor, elev, "up")
131 
132     class DownButton(ElevatorActionButton):
133 
134         def __init__(self, floor, elev):
135             super(DownButton, self).__init__(floor, elev, "down")
136 
137 
138     class ElevatorWidget(QWidget):
139         """Classe del widget de gestió de l'ascensor. Té la lògica interna i desa
140         les dades de l'ascensor"""
141 
142         def __init__(self, elev):
143             super(ElevatorWidget, self).__init__()
144             self.elevator = elev
145 
146             self.elevator_anim = ElevatorAnimation(self.elevator)
147 
148             elevator_buttons = QVBoxLayout()
149             self.buttons = [FloorWidget(x, elev) for x in range(self.elevator.n_floors)]
150             self.buttons[0].remove_down()
151             self.buttons[-1].remove_up()
152             self.buttons.reverse()
```

```
153         [elevator_buttons.addWidget(x) for x in self.buttons]
154
155         elevator_pane = QHBoxLayout()
156         elevator_pane.addWidget(self.elevator_anim)
157         elevator_pane.addLayout(elevator_buttons)
158         self.setLayout(elevator_pane)
159         self.update()
160
161
162     def update(self):
163         self.elevator_anim.update()
164         for i in self.buttons:
165             i.update()
166
167
168
169
170 class DemoWidget(QGroupBox):
171     """Classe del widget de gestió de l'ascensor. Té la lògica interna i desa
172     les dades de l'ascensor"""
173
174     def __init__(self, elev):
175         super(DemoWidget, self).__init__()
176         cadena_titol = self.trUtf8("Demonstracions d'execució")
177         self.setTitle(cadena_titol)
178         cases = []
179         cases.append(("Cridades internes",
180                   [(0, "open"), (1,"open"), (2,"open"),
181                    (3,"open"), (4,"open"), (5,"open")]))
182         cases.append(("Pujada simple",
183                   [(2, "up"), (3, "open")]))
184         cases.append(("Baixada simple",
185                   [(4, "down"), (3, "open")]))
186         cases.append(("Tot pujades",
187                   [(0, "up"), (1, "up"), (2, "up"), (3, "up"), (4, "up")]))
188         cases.append(("Tot baixades",
189                   [(1, "down"), (2, "down"), (3, "down"),
190                    (4, "down"), (5,"down")]))
191         cases.append(("Adalt des del mig",
192                   [(2, "up"), (4, "up")]))
193         cases.append(("Abaix des del mig",
194                   [(2, "down"), (4, "down")]))
195         cases.append(("Mescla des del mig",
196                   [(1, "up"), (2, "open"), (3, "down"),
197                    (4, "up"), (5, "open")]))
198         cases.append(("Cas complexe des del mig",
199                   [(1, "up"), (2, "down"), (4, "up"), (5, "down")]))
200         cases.append(("Conflicte de portes",
201                   [(3, "open"), (3, "up"), (3, "down"),
```

```
202                     (4, "open"), (1, "open")]))
203         buttons = [DemoButton(i, elev) for i in cases]
204         demo_pane = QVBoxLayout()
205         for i in buttons:
206             demo_pane.addWidget(i)
207         self.setLayout(demo_pane)
208
209
210 class DemoButton(QPushButton):
211     """Classe del widget de gestió de l'ascensor. Té la lògica interna i desa
212     les dades de l'ascensor"""
213
214     def __init__(self, case, elev):
215         title, actions = case[0], case[1]
216         super(DemoButton, self).__init__()
217         self.setText(QString(title))
218         self.elevator = elev
219         self.actions = actions
220         self.connect(self, SIGNAL("clicked()"), self.add_actions)
221
222     def add_actions(self):
223         self.elevator.add_list(self.actions)
224
225
226
227
228 class ElevatorAnimation(QGraphicsView):
229     """Animació de l'ascensor. El «delta» són els milisegons d'acció i
230     transició d'obrir portes."""
231     def __init__(self, elev):
232         super(ElevatorAnimation, self).__init__()
233         self.elevator = elev
234         self.delta = 1*1000
235         # tamany d'una cel.lla, com si fos una casella
236         self.cell = 64
237
238         self.max_width = self.cell*3
239         self.max_height = self.cell*self.elevator.n_floors
240         self.scene = QGraphicsScene(0, 0, self.max_width, self.max_height)
241         self.setScene(self.scene)
242
243         self.populate()
244
245         self.animator = QTimer()
246         self.animator.setInterval(self.delta)
247         self.animator.timeout.connect(self.animate)
248         self.update()
249
250
```

```
251     def update(self):
252         self.animate()
253 
254 
255     def populate(self):
256         """Posa els elements dins l'animació"""
257         #self.pos = self.max_height - self.cell, self.cell
258 
259         pixmap = QPixmap("img/elevator.svg").scaledToHeight(self.cell)
260         self.elev_back = QGraphicsPixmapItem(pixmap)
261         pixmap = QPixmap("img/elevator_left.svg").scaledToHeight(self.cell)
262         self.elev_left = QGraphicsPixmapItem(pixmap)
263         pixmap = QPixmap("img/elevator_right.svg").scaledToHeight(self.cell)
264         self.elev_right = QGraphicsPixmapItem(pixmap)
265         pixmap = QPixmap("img/elevator_closed.svg").scaledToHeight(self.cell)
266         self.elev_closed = QGraphicsPixmapItem(pixmap)
267         # l.setZValue(-100)
268 
269 
270         self.scene.addItem(self.elev_back)
271         self.scene.addItem(self.elev_left)
272         self.scene.addItem(self.elev_right)
273         self.scene.addItem(self.elev_closed)
274         self.elev_back.setVisible(False)
275         self.elev_left.setVisible(False)
276         self.elev_right.setVisible(False)
277         self.elev_closed.setVisible(False)
278 
279 
280     def animate(self):
281         action = self.elevator.last
282         if action == "up" or action == "down":
283             self.animate_up_down()
284         elif action == "open":
285             self.animate_open()
286         elif action == "close":
287             self.animate_close()
288         elif action == "wait":
289             self.animate_wait()
290         elif action == "init":
291             self.animate_init()
292         else:
293             raise NotImplementedError
294 
295     def animate_to(self, when, item, x, y):
296         animation = QGraphicsItemAnimation()
297         timeline = QTimeLine(self.delta)
298         timeline.setFrameRange(0, 60)
299         animation.setPosAt(when, QPointF(x, y))
```

```
300             animation.setItem(item)
301             animation.setTimeLine(timeline)
302             return animation
303 
304         def animate_open(self):
305             self.elev_back.setPos(*self.pos)
306             self.elev_left.setPos(*self.pos)
307             self.elev_right.setPos(*self.pos)
308 
309             self.elev_closed.setVisible(False)
310             self.elev_back.setVisible(True)
311             self.elev_left.setVisible(True)
312             self.elev_right.setVisible(True)
313             self.anim_left = self.animate_to(0.2, self.elev_left, \
314                                              self.cell*2, self.calc_where())
315             self.anim_right = self.animate_to(0.2, self.elev_right, \
316                                               0, self.calc_where())
317             self.anim_left.timeLine().start()
318             self.anim_right.timeLine().start()
319             self.animator.start()
320 
321         def animate_wait(self):
322             pass
323 
324         def animate_close(self):
325             self.elev_closed.setVisible(False)
326             self.elev_back.setVisible(True)
327             self.elev_left.setVisible(True)
328             self.elev_right.setVisible(True)
329             self.anim_left = self.animate_to(0.2, self.elev_left, \
330                                              self.cell, self.calc_where())
331             self.anim_right = self.animate_to(0.2, self.elev_right, \
332                                                self.cell, self.calc_where())
333             self.anim_left.timeLine().start()
334             self.anim_right.timeLine().start()
335             self.animator.start()
336 
337 
338         def animate_init(self):
339             self.elev_closed.setVisible(True)
340             self.elev_back.setVisible(False)
341             self.elev_left.setVisible(False)
342             self.elev_right.setVisible(False)
343             self.anim = self.animate_to(0.2, self.elev_closed, \
344                                              self.cell, self.calc_where())
345             self.anim.timeLine().start()
346             self.animator.start()
347 
348         def animate_up_down(self):
```

```
349             self.elev_closed.setVisible(True)
350             self.elev_back.setVisible(False)
351             self.elev_left.setVisible(False)
352             self.elev_right.setVisible(False)
353             self.anim = self.animate_to(0.2, self.elev_closed, \
354                                         self.cell, self.calc_where())
355             self.anim.timeLine().start()
356             self.animator.start()
357 
358 
359         @property
360         def pos(self):
361             return self.cell, self.calc_where()
362 
363         def calc_where(self):
364             offset = self.elevator.n_floors - (self.elevator.floor + 1)
365             return self.cell * offset
```

Programa principal - main.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

__version__ = "0.0.1"

from igu import MainWindow

import sys

from PyQt4.QtGui import QApplication

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec_()
    sys.exit()
```