

INSTITUT SUPÉRIEUR
D'ÉLECTRONIQUE DE PARIS

TIPE

Le Rubik's Cube

					3			
5			7	1		2		
	4			9		8	6	
		3						8
	5	2	6			9	3	
9		7		3			5	
			3		5			
6				2		1		
	9		1				4	

Laurent Tainturier & Alphonse Terrier

supervisé par
M. Patrick COUVEZ

2016-2017

Table des matières

Introduction	1
1 Présentation du sudoku	3
2 Électronique	4
3 Mécanique	5
4 Informatique	6
4.1 Résolution du sudoku	6
4.2 Reconnaissance du sudoku	6
4.3 Écriture et contrôle des moteurs	6
A Fichier principal	7
B Script de résolution des sudokus	11
C Script de gestion de la caméra	14

Introduction

Chapitre 1

Présentation du sudoku

Chapitre 2

Électronique

Chapitre 3

Mécanique

Chapitre 4

Informatique

4.1 Résolution du sudoku

4.2 Reconnaissance du sudoku

4.3 Écriture et contrôle des moteurs

Annexe A

Fichier principal

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import os
5  import time
6  import threading
7
8  import lcd
9  import save
10 import numpy as np
11 import button as bt
12 import camera as cm
13 import resolution as rs
14
15
16 class Sudoku:
17     def __init__(self):
18         self.number = 0
19         self.errors = []
20         self.taille = (3, 3)
21         self.power = True
22         self.power_mode = True
23         self.select_mode = None
24         self.modes = ["Automatic", "Practice"]
25         self.mode = self.modes[self.number]
26         self.nb_cases = self.taille[0] * self.taille[1]
27         self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
28         self.liste_position = []
29         self.methode_resolution = "Backtracking"
30
31         self.stop = Stop(self)
32         self.camera = cm.Camera(self)
33         self.resolution = rs.Resolution(self)
34         self.button = bt.Buttons([11, 13, 15])
35
36     def start(self):
37         while self.power:
38             lcd.write("Sudoku Plotter Welcome!")
39             bt_pressed = self.wait(5)
40             if bt_pressed[2]:
41                 self.quit()
42             elif bt_pressed[1]:
43                 self.button.pressed = [0, 0, 0]
44                 lcd.write("Choose the mode")
45                 lcd.write(self.mode + "?", 2)
46                 time.sleep(4)
47                 while self.power_mode:
48                     if self.button.pressed[0]:
49                         self.button.pressed[0] = 0
50                         self.number = 1 - self.number
51                         self.mode = self.modes[self.number]
52                         lcd.write(self.mode + "?", 2)
53                         time.sleep(2)
```



```

55         if self.button.pressed[1]:
56             self.button.pressed[1] = 0
57             self.select_mode = self.mode
58
59         if self.button.pressed[2]:
60             self.button.pressed[2] = 0
61             self.quit()
62
63         if self.select_mode == "Automatic":
64             lcd.write("Sudoku Plotter Automatic Mode")
65             time.sleep(2)
66             self.stop.run()
67             self.automaticMode()
68
69         if self.select_mode == "Practice":
70             lcd.write("Sudoku Plotter Practice Mode")
71             time.sleep(2)
72             self.practiceMode()
73
74         if self.select_mode:
75             self.select_mode = None
76             self.start()
77
78         time.sleep(1)
79     time.sleep(1)
80
81     lcd.write("Sudoku Plotter Goodbye!")
82     time.sleep(2)
83     self.power = False
84     self.stop.power = False
85     self.button.stop()
86     os.system("sudo shutdown -h now")
87
88     def automaticMode(self):
89         lcd.write("Photo Capture... Please Wait!")
90         self.camera.takePhoto()
91         if "camera_error" not in self.errors:
92             lcd.write("Succeeded!", 2)
93             time.sleep(2)
94         else:
95             lcd.write("Camera isn't connected!")
96             time.sleep(3)
97             lcd.write("Camera error Continue?")
98             time.sleep(4)
99             if self.button.pressed[2]: self.start()
100         lcd.write("Recognition...")
101         lcd.write("Please Wait!", 2)
102         os.system("sudo python extraction.py")
103         lcd.write("Succeeded!", 2)
104         time.sleep(2)
105         lcd.write("Solving... Please Wait!")
106         time.sleep(0.5)
107         self.sudoku = save.readSudoku()
108         self.sudoku, self.liste_position = self.resolution.start(self.sudoku, self.methode_resolution)
109         if "sudoku_insoluble" not in self.errors:
110             lcd.write("Succeeded!", 2)
111             print("finished /n", self.sudoku)
112             time.sleep(2)
113             lcd.write("Sudoku Plotter completed!", 2)
114
115         if self.errors and self.errors != ["camera_error"]:
116             lcd.write("Failed!", 2)
117             self.wait()
118             self.errors = []
119             self.start()
120
121     def practiceMode(self):
122         pass
123
124     def wait(self, sleep=2):
125         previous_time = time.time()
126         while time.time() - previous_time < sleep:
127             if self.button.pressed is not None:

```

```

128         bt = self.button.pressed
129         self.button.pressed = [0, 0, 0]
130         return bt
131         time.sleep(0.1)
132         return [0, 0, 0]
133
134     def setError(self, error):
135         if error not in self.errors:
136             self.errors.append(error)
137
138     def quit(self):
139         lcd.write("Sudoku Plotter Goodbye?")
140         time.sleep(2)
141         bt_pressed = self.wait(4)
142         if bt_pressed[1] or bt_pressed[2]:
143             self.power = False
144             self.power_mode = False
145             self.start()
146
147
148     class Stop(threading.Thread):
149         def __init__(self, boss):
150             threading.Thread.__init__(self)
151             self.power = True
152             self.boss = boss
153
154         def start(self):
155             while self.power:
156                 if self.boss.button[2]:
157                     self.boss.power = False
158                     self.boss.power_mode = False
159                     self.boss.start()
160                     self.power = False
161
162     """
163     def start(self):
164         lcd.write("Sudoku Plotter Welcome!")
165         time.sleep(2)
166         while self.power:
167             lcd.write("Sudoku Plotter Welcome!")
168             print("Welcome")
169             if self.button.pressed == 0:
170                 self.button.pressed = None
171                 self.number = 1 - self.number
172                 self.mode = self.modes[self.number]
173
174             if self.button.pressed == 1:
175                 self.button.pressed = None
176                 self.select_mode = self.mode
177
178             if self.button.pressed == 2:
179                 self.button.pressed = None
180                 lcd.write("Goodbye?", 2)
181                 while not self.button.pressed:
182                     time.sleep(0.1)
183                 if self.button.pressed == 1:
184                     self.power = False
185                     self.button.pressed = None
186
187             while self.select_mode == "automatic":
188                 lcd.write("Automatic?", 2)
189                 if self.button.pressed == 2:
190                     self.select_mode = None
191                     self.button.pressed = None
192
193             while self.select_mode == "practice":
194                 lcd.write("Practice?", 2)
195                 if self.button.pressed == 2:
196                     self.select_mode = None
197                     self.button.pressed = None
198
199             time.sleep(2)
200         print("Bye!")

```

```

201  """
202
203  lcd = lcd.LCD()
204  sudoku = Sudoku()
205  sudoku.button.start()
206  sudoku.start()
207
208  """
209  import resolution as rs
210  import camera as cm
211  import step_motor as stp
212  import os
213
214  lcd = lcd.LCD()
215  lcd.write("Sudoku Plotter Welcome!")
216  time.sleep(2)
217  lcd.write("Automatic?", 2)
218  time.sleep(2)
219  lcd.write("Practice?", 2)
220  time.sleep(2)
221  lcd.write("Step by step?", 2)
222  time.sleep(2)
223  lcd.write("Photo capture in progress...")
224  time.sleep(2)
225  lcd.write("Use the joystick to move or write")
226  time.sleep(2)
227  lcd.write("Joystick isn't connected!")
228  time.sleep(2)
229  lcd.write("Recognition...")
230  lcd.write("in progress...", 2)
231  time.sleep(2)
232  lcd.write("Failed!", 2)
233  time.sleep(2)
234  lcd.write("Succeeded!", 2)
235  time.sleep(2)
236  lcd.write("Solving...")
237  time.sleep(2)
238  lcd.write("Sudoku Plotter Goodbye!")
239  """

```

Annexe B

Script de résolution des sudokus

```
1  #!/usr/bin/env python3
2
3  import numpy as np
4  from time import time
5
6
7  class Resolution:
8      """
9      Classe permettant de résoudre un sudoku grâce à différentes méthodes, à savoir :
10         - inclusion
11         - exclusion
12         - backtracking
13         - ...
14      Si le sudoku n'est pas résoluble, lève une erreur
15      """
16
17  def __init__(self, boss):
18      self.boss = boss
19      self.taille = self.boss.taille
20      self.nb_cases = self.boss.nb_cases
21      self.methode_resolution = None
22      self.vitesse = None
23      self.sudoku = np.zeros((self.nb_cases, self.nb_cases), int)
24      self.liste_sudoku = []
25      self.liste_position = []
26      self.power = True
27
28  def start(self, sudoku, methode, vitesse=None):
29      self.sudoku = sudoku
30      self.methode_resolution = methode
31      self.vitesse = vitesse
32      print(self.methode_resolution)
33      zero_time = time()
34      if not self.checkBeforeStart():
35          self.boss.setError("sudoku_insoluble")
36          return self.sudoku, []
37      if self.methode_resolution == "Globale": self.optimale()
38      elif self.methode_resolution == "Inclusion": self.inclusion()
39      elif self.methode_resolution == "Exclusion": self.exclusion()
40      elif self.methode_resolution == "Backtracking":
41          self.createListe()
42          self.backTracking()
43      else: print("La méthode n'est pas reconnue")
44      print(time() - zero_time, '\n')
45      return np.copy(self.sudoku), self.liste_position
46
47  def checkBeforeStart(self):
48      for i in range(self.nb_cases):
49          liste_ligne = []
50          liste_colonne = []
51          liste_carre = []
52          x = 3 * (i // 3)
53          y = 3 * (i % 3)
54          for j in range(self.nb_cases):
```

```

55         if self.sudoku[x + j // 3, y + j % 3] in liste_carre \
56             or self.sudoku[j, i] in liste_colonne \
57             or self.sudoku[i, j] in liste_ligne:
58             return False
59         if self.sudoku[i, j] != 0:
60             liste_ligne.append(self.sudoku[i, j])
61         if self.sudoku[j, i] != 0:
62             liste_colonne.append(self.sudoku[j, i])
63         if self.sudoku[x + j // 3, y + j % 3] != 0:
64             liste_carre.append(self.sudoku[x + j // 3, y + j % 3])
65     return True
66
67 def createListe(self):
68     self.liste_position = []
69     liste_tailles_cases_vides = []
70     liste_tailles = [[] for i in range(self.nb_cases - 1)]
71     for x in range(self.nb_cases):
72         for y in range(self.nb_cases):
73             if not self.sudoku[x][y]:
74                 self.liste_position.append((x, y))
75                 liste_tailles_cases_vides.append(len(self.checkListe(x, y)))
76                 liste_tailles[liste_tailles_cases_vides[-1] - 2].append((x, y))
77     liste_position = []
78     for i in range(self.nb_cases - 1):
79         liste_position += liste_tailles[i]
80     return liste_position
81
82 def checkListe(self, x, y):
83     """
84     Renvoie la liste des valeurs possibles pour la case de coordonnées x et y
85     :param x: int: ligne
86     :param y: int: colonne
87     :return: liste: list
88     """
89     liste = []
90     if self.sudoku[x][y] == 0:
91         liste = [i + 1 for i in range(self.nb_cases)]
92         block_x = x - x % self.taille[0]
93         block_y = y - y % self.taille[1]
94         for i in range(self.nb_cases):
95             if self.sudoku[x, i] in liste:
96                 liste.remove(self.sudoku[x, i])
97             if self.sudoku[i, y] in liste:
98                 liste.remove(self.sudoku[i, y])
99             if self.sudoku[block_x + i % self.taille[1], block_y + i // self.taille[0]] in liste:
100                 liste.remove(self.sudoku[block_x + i % self.taille[1], block_y + i // self.taille[0]])
101     return liste
102
103 def optimale(self):
104     self.liste_position = self.createListe()
105     size = []
106     for x, y in self.liste_position:
107         size.append(len(self.checkListe(x, y)))
108     self.backTracking()
109
110 def inclusion(self):
111     pass
112
113 def exclusion(self):
114     pass
115
116 def backTracking(self):
117     """
118     Résoud un sudoku selon la méthode de backtracking
119     :return: None or -1
120     """
121     self.liste_sudoku = []
122     i = 0
123     while self.power:
124         x, y = self.liste_position[i]
125         liste = self.checkListe(x, y)
126         if liste:
127             self.sudoku[x][y] = liste.pop(0)

```

```

128         self.liste_sudoku.append(liste)
129         i += 1
130     else:
131         while not liste:
132             i -= 1
133             x, y = self.liste_position[i]
134             try:
135                 liste = self.liste_sudoku[i]
136             except IndexError:
137                 self.sudoku = self.boss.sudoku
138                 self.boss.setError("sudoku_insoluble")
139                 return -1
140             if liste:
141                 self.sudoku[x][y] = liste.pop(0)
142                 self.liste_sudoku[i] = liste
143                 i += 1
144                 break
145             else:
146                 self.liste_sudoku.pop(i)
147                 self.sudoku[x][y] = 0
148         if i < len(self.liste_position):
149             self.power = False
150         self.power = True
151
152
153
154 if __name__ == '__main__':
155     class Boss:
156         def __init__(self):
157             self.taille = (3, 3)
158             self.nb_cases = 9
159             self.methode = 'Globale'
160             self.vitesse = 'Pas à Pas'
161             self.sudoku = np.array([[0, 0, 0, 0, 0, 0, 0, 1, 2],
162                                     [0, 0, 0, 0, 0, 0, 0, 0, 3],
163                                     [0, 0, 2, 3, 0, 0, 4, 0, 0],
164                                     [0, 0, 1, 8, 0, 0, 0, 0, 5],
165                                     [0, 6, 0, 0, 7, 0, 8, 0, 0],
166                                     [0, 0, 0, 0, 0, 9, 0, 0, 0],
167                                     [0, 0, 8, 5, 0, 0, 0, 0, 0],
168                                     [9, 0, 0, 0, 4, 0, 5, 0, 0],
169                                     [4, 7, 0, 0, 0, 0, 6, 0, 0]])
170             self.Resolution = Resolution(self)
171             self.Resolution.start(self.sudoku, self.methode, self.vitesse)
172             print(self.sudoku, '\n')
173
174         def setError(self, error):
175             if error == "sudoku_insoluble":
176                 print("Le sudoku n'est pas résoluble")
177
178
179     Boss()

```

Annexe C

Script de gestion de la caméra

```
1  #!/usr/bin/env python3
2
3
4  class Camera:
5      """
6      Permet la gestion de la camera de la raspberry pi
7      Si celle-ci n'est pas disponible ou le module 'picamera'
8      n'a pas été installé correctement, lève une exception.
9      """
10
11     def __init__(self, boss):
12         self.boss = boss
13         self.camera = None
14         self.tryError()
15
16     def tryError(self):
17         try:
18             import picamera
19             self.camera = picamera.PiCamera()
20         except:
21             self.boss.setError("camera_error")
22
23     def takePhoto(self):
24         try:
25             self.camera.capture("Images/photos.jpg")
26             print("The photo has been taken")
27         except:
28             self.boss.setError("camera_error")
29
30
31 if __name__ == '__main__':
32     class Boss:
33         def setError(self, error):
34             if error == "module_camera":
35                 print("Le module 'picamera' n'a pas été installé correctement !")
36             if error == "disponibilite_camera":
37                 print("La caméra n'est pas disponible !")
38
39     Camera = Camera(Boss())
40     Camera.takePhoto()
```