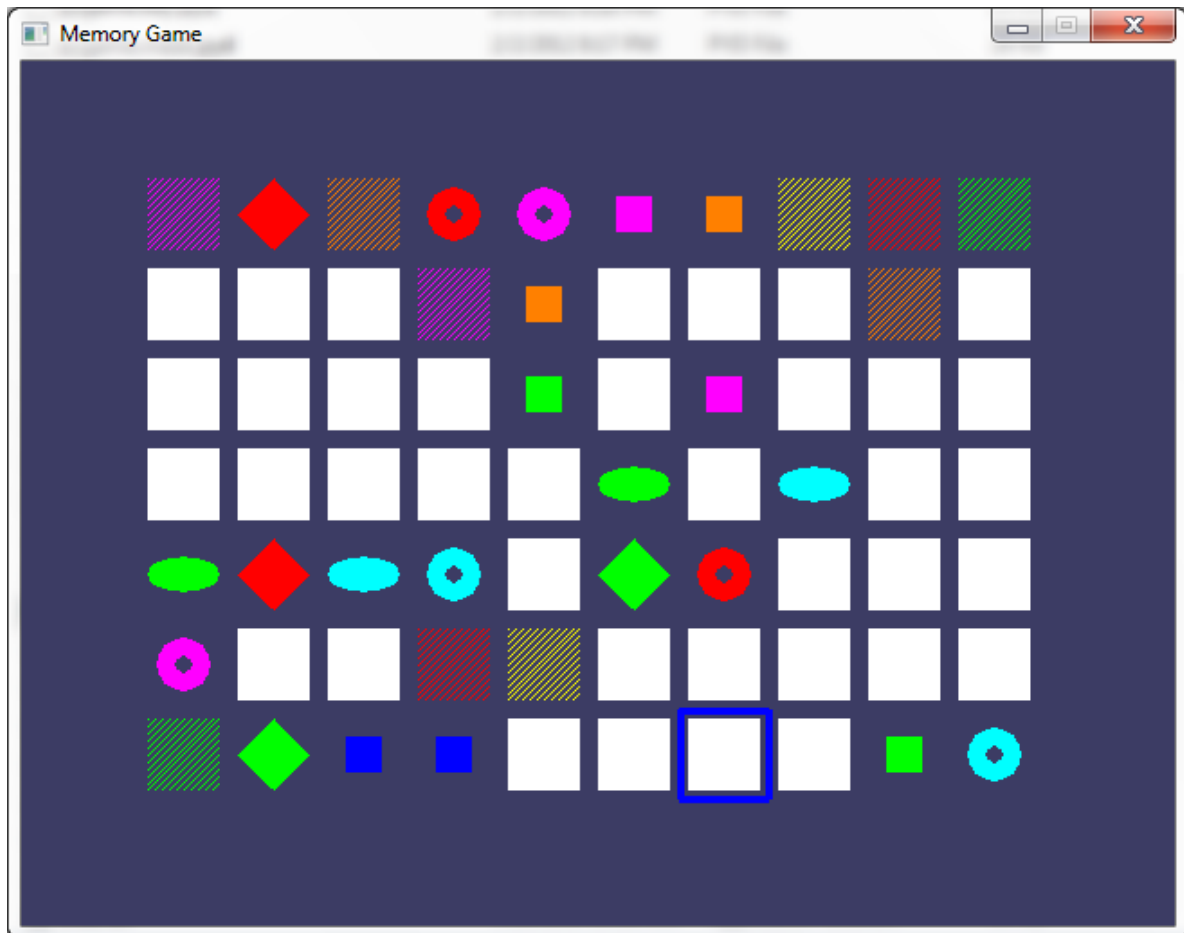


## ИГРА МЕМОРИЈЕ

### Како играти игру меморије

У игри меморије неколико сличица је прекривено белим квадратима. Постоји по две од сваке сличице. Играч може да кликне на два квадратића како би проверио које се сличице налазе иза њих. Уколико се сличице подударају, тада квадратићи остају откривени. Играч побеђује када су сви квадратићи на табли откривени. Да би помогли играчу, сви квадратићи су на почетку откривени на кратко.



### Угњежене „for“ петље

Једна од ствари које ће се често виђати у коду Игре меморије јесте коришћење for“ петље унутар друге „for“ петље. Овакви делови кода се зову **угњежене „for“ петље**. Оне се користе када се проверава свака могућа комбинација две листе. За проверу, искуцајте следећи код у „shell“-у:

```
>>> for x in [0, 1, 2, 3, 4]:
...     for y in ['a', 'b', 'c']:
...         print(x, y)
...
0 a
0 b
0 c
1 a
1 b
1 c
2 a
2 b
2 c
3 a
3 b
3 c
4 a
4 b
4 c
>>>
```

На више места у коду за Игру меморије јавиће нам се потреба да прођемо кроз сваку могућу X и Y координату на табли. Притом ћемо користити угњежене петље како бисмо добили сваку могућу комбинацију координата. Приметимо да ће унутрашња „for“ петља (односно „for“ петља која се налази унутар друге „for“ петље) проћи кроз све своје могуће кораке пре него што пређе на наредни корак спољашње „for“ петље. Ако заменимо редослед ових „for“ петљи, биће одштампане исте вредности, али по другачијем редоследу. Искуцајте наредни код у „shell“-у и упоредите редослед одштампаних вредности са редоследом одштампаних вредности из претходног примера:

```
>>> for y in ['a', 'b', 'c']:
...     for x in [0, 1, 2, 3, 4]:
...         print(x, y)
...
0 a
1 a
2 a
3 a
4 a
0 b
1 b
2 b
3 b
4 b
0 c
1 c
2 c
3 c
4 c
>>>
```

## Изворни код Игре меморије

Овај код можете преузети на следећој адреси: <http://invpy.com/memorypuzzle.py>.

Прекопирајте цео код у „IDLE“ окружење, сачувајте га под називом *memorypuzzle.py* и покрените га. Ако добијете било какву поруку о грешци, погледајте која линија кода је поменута у тој поруци и у тој линији проверите да ли постоји нека грешка у куцању. Такође, код можете да прекопираете и у „web“ форми на адреси: <http://invpy.com/diff/memorypuzzle> како бисте уочили разлике између свог кода и кода који се налази у приручнику.

Вероватно ћете током прекуцавања кода усвојити неке идеје о томе како програм функционише, а када завршите са тим, можете и сами да играте игру.

```
1. # Memory Puzzle
2. # By Al Sweigart al@inventwithpython.com
3. # http://inventwithpython.com/pygame
4. # Released under a "Simplified BSD" license
5.
6. import random, pygame, sys
7. from pygame.locals import *
8.
9. FPS = 30 # frames per second, the general speed of the
program
10. WINDOWWIDTH = 640 # size of window's width in pixels
11. WINDOWHEIGHT = 480 # size of windows' height in pixels
12. REVEALSPEED = 8 # speed boxes' sliding reveals and covers
13. BOXSIZE = 40 # size of box height & width in pixels
14. GAPSIZE = 10 # size of gap between boxes in pixels
15. BOARDWIDTH = 10 # number of columns of icons
16. BOARDHEIGHT = 7 # number of rows of icons
17. assert (BOARDWIDTH * BOARDHEIGHT) % 2 == 0, 'Board needs to
have an even number of boxes for pairs of matches.'
18. XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE +
GAPSIZE))) / 2)
19. YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT * (BOXSIZE +
GAPSIZE))) / 2)
20.
21. #           R       G       B
22. GRAY       = (100, 100, 100)
23. NAVYBLUE   = ( 60,  60, 100)
24. WHITE      = (255, 255, 255)
25. RED        = (255,  0,  0)
26. GREEN      = ( 0, 255,  0)
27. BLUE       = ( 0,  0, 255)
28. YELLOW     = (255, 255,  0)
29. ORANGE     = (255, 128,  0)
30. PURPLE     = (255,  0, 255)
31. CYAN       = ( 0, 255, 255)
```

```

32.
33. BGCOLOR = NAVYBLUE
34. LIGHTBGCOLOR = GRAY
35. BOXCOLOR = WHITE
36. HIGHLIGHTCOLOR = BLUE
37.
38. DONUT = 'donut'
39. SQUARE = 'square'
40. DIAMOND = 'diamond'
41. LINES = 'lines'
42. OVAL = 'oval'
43.
44. ALLCOLORS = (RED, GREEN, BLUE, YELLOW, ORANGE, PURPLE, CYAN)
45. ALLSHAPES = (DONUT, SQUARE, DIAMOND, LINES, OVAL)
46. assert len(ALLCOLORS) * len(ALLSHAPES) * 2 >= BOARDWIDTH *
BOARDHEIGHT, "Board is too big for the number of shapes/colors
defined."
47.
48. def main():
49.     global FPSCLOCK, DISPLAYSURF
50.     pygame.init()
51.     FPSCLOCK = pygame.time.Clock()
52.     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH,
WINDOWHEIGHT))
53.
54.     mousex = 0 # used to store x coordinate of mouse event
55.     mousey = 0 # used to store y coordinate of mouse event
56.     pygame.display.set_caption('Memory Game')
57.
58.     mainBoard = getRandomizedBoard()
59.     revealedBoxes = generateRevealedBoxesData(False)
60.
61.     firstSelection = None # stores the (x, y) of the first
box clicked.
62.
63.     DISPLAYSURF.fill(BGCOLOR)
64.     startGameAnimation(mainBoard)
65.
66.     while True: # main game loop
67.         mouseClicked = False
68.
69.         DISPLAYSURF.fill(BGCOLOR) # drawing the window
70.         drawBoard(mainBoard, revealedBoxes)
71.
72.         for event in pygame.event.get(): # event handling
loop

```

```

73.         if event.type == QUIT or (event.type == KEYUP and
event.key == K_ESCAPE):
74.             pygame.quit()
75.             sys.exit()
76.         elif event.type == MOUSEMOTION:
77.             mousex, mousey = event.pos
78.         elif event.type == MOUSEBUTTONDOWN:
79.             mousex, mousey = event.pos
80.             mouseClicked = True
81.
82.         boxx, boxy = getBoxAtPixel(mousex, mousey)
83.         if boxx != None and boxy != None:
84.             # The mouse is currently over a box.
85.             if not revealedBoxes[boxx][boxy]:
86.                 drawHighlightBox(boxx, boxy)
87.             if not revealedBoxes[boxx][boxy] and
mouseClicked:
88.                 revealBoxesAnimation(mainBoard, [(boxx,
boxy)])
89.                 revealedBoxes[boxx][boxy] = True # set the
box as "revealed"
90.                 if firstSelection == None: # the current box
was the first box clicked
91.                     firstSelection = (boxx, boxy)
92.                 else: # the current box was the second box
clicked
93.                     # Check if there is a match between the
two icons.
94.                     icon1shape, icon1color =
getShapeAndColor(mainBoard, firstSelection[0], firstSelection[1])
95.                     icon2shape, icon2color =
getShapeAndColor(mainBoard, boxx, boxy)
96.
97.                     if icon1shape != icon2shape or icon1color
!= icon2color:
98.                         # Icons don't match. Re-cover up both
selections.
99.                         pygame.time.wait(1000) # 1000
milliseconds = 1 sec
100.                        coverBoxesAnimation(mainBoard,
[(firstSelection[0], firstSelection[1]), (boxx, boxy)])
101.                        revealedBoxes[firstSelection[0]]
[firstSelection[1]] = False
102.                        revealedBoxes[boxx][boxy] = False
103.                        elif hasWon(revealedBoxes): # check if
all pairs found
104.                            gameWonAnimation(mainBoard)
105.                            pygame.time.wait(2000)
106.
107.                            # Reset the board

```

```

108.             mainBoard = getRandomizedBoard()
109.             revealedBoxes =
generateRevealedBoxesData(False)
110.
111.             # Show the fully unrevealed board for
a second.
112.             drawBoard(mainBoard, revealedBoxes)
113.             pygame.display.update()
114.             pygame.time.wait(1000)
115.
116.             # Replay the start game animation.
117.
startGameAnimation(mainBoard)
118.             firstSelection = None # reset
firstSelection variable
119.
120.             # Redraw the screen and wait a clock tick.
121.             pygame.display.update()
122.             FPSLOCK.tick(FPS)
123.
124.
125. def generateRevealedBoxesData(val):
126.     revealedBoxes = []
127.     for i in range(BOARDWIDTH):
128.         revealedBoxes.append([val] * BOARDHEIGHT)
129.     return revealedBoxes
130.
131.
132. def getRandomizedBoard():
133.     # Get a list of every possible shape in every possible
color.
134.     icons = []
135.     for color in ALLCOLORS:
136.         for shape in ALLSHAPES:
137.             icons.append( (shape, color) )
138.
139.     random.shuffle(icons) # randomize the order of the icons
list
140.     numIconsUsed = int(BOARDWIDTH * BOARDHEIGHT / 2) #
calculate how many icons are needed
141.     icons = icons[:numIconsUsed] * 2 # make two of each
142.     random.shuffle(icons)
143.
144.     # Create the board data structure, with randomly placed
icons.

```

```

145.     board = []
146.     for x in range(BOARDWIDTH):
147.         column = []
148.         for y in range(BOARDHEIGHT):
149.             column.append(icons[0])
150.             del icons[0] # remove the icons as we assign them
151.         board.append(column)
152.     return board
153.
154.
155. def splitIntoGroupsOf(groupSize, theList):
156.     # splits a list into a list of lists, where the inner
157.     # most groupSize number of items.
158.     result = []
159.     for i in range(0, len(theList), groupSize):
160.         result.append(theList[i:i + groupSize])
161.     return result
162.
163.
164. def leftTopCoordsOfBox(boxx, boxy):
165.     # Convert board coordinates to pixel coordinates
166.     left = boxx * (BOXSIZE + GAPSIZ) + XMARGIN
167.     top = boxy * (BOXSIZE + GAPSIZ) + YMARGIN
168.     return (left, top)
169.
170.
171. def getBoxAtPixel(x, y):
172.     for boxx in range(BOARDWIDTH):
173.         for boxy in range(BOARDHEIGHT):
174.             left, top = leftTopCoordsOfBox(boxx, boxy)
175.             boxRect = pygame.Rect(left, top, BOXSIZE,
176. BOXSIZE)
177.             if boxRect.collidepoint(x, y):
178.                 return (boxx, boxy)
179.     return (None, None)
180.
181. def drawIcon(shape, color, boxx, boxy):
182.     quarter = int(BOXSIZE * 0.25) # syntactic sugar
183.     half = int(BOXSIZE * 0.5) # syntactic sugar
184.
185.     left, top = leftTopCoordsOfBox(boxx, boxy) # get pixel
186.     # Draw the shapes
187.     if shape == DONUT:
188.         pygame.draw.circle(DISPLAYSURF, color, (left + half,
189. top + half), half - 5)
190.         pygame.draw.circle(DISPLAYSURF, BGCOLOR, (left +
191. half, top + half), quarter - 5)

```



```

190.     elif shape == SQUARE:
191.         pygame.draw.rect(DISPLAYSURF, color, (left + quarter,
top + quarter, BOXSIZE - half, BOXSIZE - half))
192.     elif shape == DIAMOND:
193.         pygame.draw.polygon(DISPLAYSURF, color, ((left +
half, top), (left + BOXSIZE - 1, top + half), (left + half, top +
BOXSIZE - 1), (left, top + half)))
194.     elif shape == LINES:
195.         for i in range(0, BOXSIZE, 4):
196.             pygame.draw.line(DISPLAYSURF, color, (left, top +
i), (left + i, top))
197.             pygame.draw.line(DISPLAYSURF, color, (left + i,
top + BOXSIZE - 1), (left + BOXSIZE - 1, top + i))
198.     elif shape == OVAL:
199.         pygame.draw.ellipse(DISPLAYSURF, color, (left, top +
quarter, BOXSIZE, half))
200.
201.
202. def getShapeAndColor(board, boxx, boxy):
203.     # shape value for x, y spot is stored in board[x][y][0]
204.     # color value for x, y spot is stored in board[x][y][1]
205.     return board[boxx][boxy][0], board[boxx][boxy][1]
206.
207.
208. def drawBoxCovers(board, boxes, coverage):
209.     # Draws boxes being covered/revealed. "boxes" is a list
210.     # of two-item lists, which have the x & y spot of the
box.
211.     for box in boxes:
212.         left, top = leftTopCoordsOfBox(box[0], box[1])
213.         pygame.draw.rect(DISPLAYSURF, BGCOLOR, (left, top,
BOXSIZE, BOXSIZE))
214.         shape, color = getShapeAndColor(board, box[0],
box[1])
215.         drawIcon(shape, color, box[0], box[1])
216.         if coverage > 0: # only draw the cover if there is an
coverage
217.             pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left,
top, coverage, BOXSIZE))
218.         pygame.display.update()
219.         FPSLOCK.tick(FPS)
220.
221.
222. def revealBoxesAnimation(board, boxesToReveal):
223.     # Do the "box reveal" animation.
224.     for coverage in range(BOXSIZE, (-REVEALSPEED) - 1, -
REVEALSPEED):
225.         drawBoxCovers(board, boxesToReveal, coverage)
226.

```



```

227.
228. def coverBoxesAnimation(board, boxesToCover):
229.     # Do the "box cover" animation.
230.     for coverage in range(0, BOXSIZE + REVEALSPEED,
REVEALSPEED):
231.         drawBoxCovers(board, boxesToCover, coverage)
232.
233.
234. def drawBoard(board, revealed):
235.     # Draws all of the boxes in their covered or revealed
state.
236.     for boxx in range(BOARDWIDTH):
237.         for boxy in range(BOARDHEIGHT):
238.             left, top = leftTopCoordsOfBox(boxx, boxy)
239.             if not revealed[boxx][boxy]:
240.                 # Draw a covered box.
241.                 pygame.draw.rect(DISPLAYSURF, BOXCOLOR,
(left, top, BOXSIZE, BOXSIZE))
242.             else:
243.                 # Draw the (revealed) icon.
244.                 shape, color = getShapeAndColor(board, boxx,
boxy)
245.                 drawIcon(shape, color, boxx, boxy)
246.
247.
248. def drawHighlightBox(boxx, boxy):
249.     left, top = leftTopCoordsOfBox(boxx, boxy)
250.     pygame.draw.rect(DISPLAYSURF, HIGHLIGHTCOLOR, (left - 5,
top - 5, BOXSIZE + 10, BOXSIZE + 10), 4)
251.
252.
253. def startGameAnimation(board):
254.     # Randomly reveal the boxes 8 at a time.
255.     coveredBoxes = generateRevealedBoxesData(False)
256.     boxes = []
257.     for x in range(BOARDWIDTH):
258.         for y in range(BOARDHEIGHT):
259.             boxes.append( (x, y) )
260.     random.shuffle(boxes)
261.     boxGroups = splitIntoGroupsOf(8, boxes)
262.
263.     drawBoard(board, coveredBoxes)
264.     for boxGroup in boxGroups:
265.         revealBoxesAnimation(board, boxGroup)
266.         coverBoxesAnimation(board, boxGroup)
267.
268.

```

```

269. def gameWonAnimation(board):
270.     # flash the background color when the player has won
271.     coveredBoxes = generateRevealedBoxesData(True)
272.     color1 = LIGHTBGCOLOR
273.     color2 = BGCOLOR
274.
275.     for i in range(13):
276.         color1, color2 = color2, color1 # swap colors
277.         DISPLAYSURF.fill(color1)
278.         drawBoard(board, coveredBoxes)
279.         pygame.display.update()
280.         pygame.time.wait(300)
281.
282.
283. def hasWon(revealedBoxes):
284.     # Returns True if all the boxes have been revealed,
    otherwise False
285.     for i in revealedBoxes:
286.         if False in i:
287.             return False # return False if any boxes are
    covered.
288.     return True
289.
290.
291. if __name__ == '__main__':
292.     main()

```

### Заслуге и коришћене библиотеке

```

1. # Memory Puzzle
2. # By Al Sweigart al@inventwithpython.com
3. # http://inventwithpython.com/pygame
4. # Released under a "Simplified BSD" license
5.
6. import random, pygame, sys
7. from pygame.locals import *

```

На почетку програма се налазе коментари о томе која је игра у питању, ко ју је направио, као и где корицник може да нађе додатне информације. Ту се такође налази и информација да је код доступан за умножавање под „Simplified BSD“ лиценцом.

Овај програм користи многе функције које се налазе у екстерним библиотекама које увози у шестој линији. Линија 7 такође представља „import“ команду, али у формату:

„from (назив библиотеке) import \*“,

што заправо значи да не морамо да укуцамо назив библиотеке иза те команде. У команди `pygame.locals` не постоје функције, али је ту смештено неколико константи које ми желимо да користимо, као што су `MOUSEMOTION`, `KEYUP` или `QUIT`. Користећи овај формат команде „import“, ми треба да укуцамо само `MOUSEMOTION`, уместо `pygame.locals.MOUSEMOTION`.

## Магични бројеви су лоши

```
9. FPS = 30 # frames per second, the general speed of the program
10. WINDOWWIDTH = 640 # size of window's width in pixels
11. WINDOWHEIGHT = 480 # size of windows' height in pixels
12. REVEALSPEED = 8 # speed boxes' sliding reveals and covers
13. BOXSIZE = 40 # size of box height & width in pixels
14. GAPSIZE = 10 # size of gap between boxes in pixels
```

Програм ове игре садржи доста константи, а оне могу бити јако корисне. На пример, уместо да користимо променљиву BOXSIZE у нашем коду, могли бисмо да укуцамо само број 40 директно у код. Међутим, постоје два разлога зашто користимо константе.

Прво, ако бисмо касније желели да променимо величину сваког квадратића, морали бисмо да прођемо кроз цео програм и да преправимо сваку линију где смо укуцали број 40. Коришћењем константе BOXSIZE, ми треба да изменимо само линију 13 и остатак програма ће аутоматски бити ажуриран. Овај начин је много бољи, поготово зато што вредност 40 можемо да искористимо и за још нешто поред величине белих квадратића, па исправљањем сваке линије у којој бисмо уочили број 40 могли бисмо да изазовемо грешку у нашем програму.

Друго, овај приступ чини код прегледнијим. Овратите пажњу на следећи сегмент и погледајте линију 18. Она израчунава константу XMARGIN која одређује колико пиксела се налази са бочних страна читаве табле. Израз делује компликовано, али јасно објашњава како се рачуна поменута константа.

Линиј 18 изгледа овако:

```
XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE + GAPSIZE))) / 2)
```

Када линија 18 не би садржала константе, она би изгледала овако:

```
XMARGIN = int((640 - (10 * (40 + 10))) / 2)
```

На овај начин постаје немогуће да се разуме шта заправо означава ова константа и како је добијена. Ови необјашњени бројеви у изворном коду се често називају **магичним бројевима**. Када год ухватите себе да уносите магичне бројеве, требало би да размотрите опцију да их замените константама. Претходне линије ће рачунару бити идентичне, али програмеру који чита изворни код и покушава да разуме како он функционише, друга верзија линије 18 неће имати нимало смисла. Константе доприносе прегледности изворног кода.

Са друге стране, могуће је и да претерате са заменом бројева константама. Посматрајмо следећи код:

```
ZERO = 0
ONE = 1
TWO = 99999999
TWOANDTHREEQUARTERS = 2.75
```

Немојте писати овакав код. У оваквим случајевима, замена бројева константама је бесмислена.

## Провера смислености помоћу команде assert

```
15. BOARDWIDTH = 10 # number of columns of icons
16. BOARDHEIGHT = 7 # number of rows of icons
17. assert (BOARDWIDTH * BOARDHEIGHT) % 2 == 0, 'Board needs to
have an even number of boxes for pairs of matches.'
18. XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE +
GAPSIZE))) / 2)
19. YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT * (BOXSIZE +
GAPSIZE))) / 2)
```

Команда `assert` коју видимо у линији 15 осигурава да се у ширини и висини табле коју смо означили налази паран број квадратића (што је битно јер ћемо у овој игри имати парове сличица). Постоје три дела `assert` команде: кључна реч – `assert`, затим израз, који за вредност `False` резултује прекидом програма, и трећи део (иза зареза иза израза) је ниска, односно стринг, који се појављује у случају прекида програма због „асертовања“.

Команда `assert` заједно са изразом у суштини говори: „Програмер тврди да овај израз мора бити тачан, иначе се програм прекида.“. Ово је добар начин провере смислености програма, тј. да осигурамо да ако извршење програма прође „асертовање“, можемо да сматрамо да код ради како је очекивано.

## Испитивање парности броја

Ако производ ширине и висине табле поделимо са два и притом је остатак тог дељења једнак нули (тај остатак можемо да добијемо помоћу оператора `%`), тада је тај број паран. Парни бројеви при дељењу са два увек дају остатак нула. Непарни бројеви при дељењу са два увек дају остатак један. Ово је згодан трик за упамтити ако вам затреба код који ће вам рећи да ли је неки број паран или непаран.

```
>>> isEven = someNumber % 2 == 0
>>> isOdd = someNumber % 2 != 0
```

У примеру изнад, ако је број `someNumber` паран, тада ће вредност променљиве `isEven` бити `True`. У случају када би број `someNumber` био непаран, тада би вредност променљиве `isOdd` имала вредност `True`.

## Рано и често пуцање програма

Пуцање програма је лоша ствар. То се дешава када ваш програм има неких грешака у коду и не може да настави са радом. Ипак, постоје случајеви где рано пуцање програма спречава горе грешке у његовом каснијем раду. Ако вредности које смо изабрали за ширину и висину табле, односно вредности константи `BOARDWIDTH` и `BOARDHEIGHT` које смо одредили у линијама 15 и 16, резултују непарним бројем квадратића (на пример, ако би ширина била 3, а висин 5), тада би увек остала једна случица која не би имала свој пар. Ово би изазвало квар у каснијем раду програма и било би потребно доста после око његовог исправљања да би се на крају дошло до закључка да се грешка налази на самом почетку програма. Покушајте да део кода који се односи на „асертовање“ ставите под коментар (како се не би извршила та команда), а затим подесите

ширину и висину табле, тј. константе BOARDWIDTH и BOARDHEIGHT, да буду непарни бројеви. Када покренете програм, одмах ће вам се приказати грешка у линији 149 вашег програма *memorypuzzle.py* која се налази у оквиру функције *getRandomizedBoard()*.

Traceback (most recent call last):

```
Traceback (most recent call last):
  File "C:\book2svn\src\memorypuzzle.py", line 292, in <module>
    main()
  File "C:\book2svn\src\memorypuzzle.py", line 58, in main
    mainBoard = getRandomizedBoard()
  File "C:\book2svn\src\memorypuzzle.py", line 149, in getRandomizedBoard
    columns.append.icons[0])
IndexError: list index out of range
```

Изгубили бисмо доста времена посматрајући *getRandomizedBoard()* функцију покушавајући да схватимо шта није у реду са њом, да бисмо на крају схватили да у њој нема грешака. Прави извор квара налази се још у линијама 15 и 16 где су дефинисане константе BOARDWIDTH и BOARDHEIGHT.

„Асертовање“ осигурава да се овако нешто не деси. Ако наш програм већ мора да „пукне“, ми жемо да се то деси чим открије да нешто није у реду, јер, у супротном, квар може постати видљив тек много касније у програму. Зато је боље да програм пукне раније.

Ми заправо желимо да додамо команду „assert“ када год имамо неки услов у програму који уек мора бити тачан, тј. да има вредност True. Зато је битно и да програм пуца често (ако има грешака, наравно). Не морамо ни да претерујемо и да уводимо команду „assert“ свуда, али често пуцање програма уз „асертовање“ умногоме доприноси откривању правог извора квара.

### Дотеривање изворног кода

```
21. #           R    G    B
22. GRAY       = (100, 100, 100)
23. NAVYBLUE   = ( 60,  60, 100)
24. WHITE      = (255, 255, 255)
25. RED        = (255,  0,  0)
26. GREEN      = (  0, 255,  0)
27. BLUE       = (  0,  0, 255)
28. YELLOW     = (255, 255,  0)
29. ORANGE     = (255, 128,  0)
30. PURPLE     = (255,  0, 255)
31. CYAN       = (  0, 255, 255)
32.
33. BGCOLOR    = NAVYBLUE
34. LIGHTBGCOLOR = GRAY
35. BOXCOLOR   = WHITE
36. HIGHLIGHTCOLOR = BLUE
```

Подсетимо се да су боје у Пајтејму представљене помоћу уређених тројки целих бројева од 0 до 255. Ова три броја представљају количину црвене, зелене и плаве боје, редом, због чега се ове тројке и називају RGB вредностима (Red, Green, Blue). Приметимо да је распоред уређених

тројки, које видимо у линијама 22-31, такав да се R, G и B вредности свих уређених тројки налазе у по једној линији. У Пајтону увлачење текста (празан простор на почетку линије) мора бити прецизан, али размаци у остатку линије нису толико битни. Увођењем размака у уређеним тројкама постаје лако уочљиво поређење RGB вредности сваке боје.

На овај начин код чинимо прегледнијим и лепшим, али не морате да трошите много времена на то. Сам код не мора да буде леп да би радио.

### Вођење рачуна о довољном броју сличица

```
44. ALLCOLORS = (RED, GREEN, BLUE, YELLOW, ORANGE, PURPLE, CYAN)
45. ALLSHAPES = (DONUT, SQUARE, DIAMOND, LINES, OVAL)
46. assert len(ALLCOLORS) * len(ALLSHAPES) * 2 >= BOARDWIDTH *
BOARDHEIGHT, "Board is too big for the number of shapes/colors
defined."
```

Да би програм наше игрице могао да креира сличице за сваку могућу комбинацију боје и облика, морамо да формирамо торку која ће садржати све ове вредности. Постоји још једно „асертовање“ у линији 46 које ће осигурати да постоји довољно комбинација боја и облика како би се попунила цела табла задате величине. Ако не би постојало довољно комбинација, програм би пукао на линији 46 и ми бисмо знали да – или морамо да додамо још боја и облика, или да смањимо ширину и висину наше табле. Са 7 боја и 5 облика можемо да направимо 35 ( $7 \cdot 5$ ) различитих сличица, а пошто нам требају парови ових сличица, то значи да можемо да креирамо таблу са до 70 квадратића.

### Уређене торке или листе, непроменљиве или променљиве

Можемо приметити да су ALLCOLORS и ALLSHAPES променљиве уствари уређене торке уместо листе. Торке и листе су исте по свему, сем две разлике: торке користе обичне заграде уместо угласте, и чланови тори не могу да се мењају (непроменљиви), док се у листама чланови могу мењати (променљиви).

Примери:

```
>>> listVal = [1, 1, 2, 3, 5, 8]
>>> tupleVal = (1, 1, 2, 3, 5, 8)
>>> listVal[4] = 'hello!'
>>> listVal
[1, 1, 2, 3, 'hello!', 8]
>>> tupleVal[4] = 'hello!'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tupleVal
(1, 1, 2, 3, 5, 8)
>>> tupleVal[4]
5
```

Када покушавамо да променимо индекс 2 у торци, Пајтон диже грешку са поруком да торка не подржава „item assignment“.



### Један члан у уређеној торци захтева запету која га прати

Такође, још један детаљ везан за уређене торке: ако вам икада затреба да напишете код о торци која има само једну вредност у њој, тада је неопхоно ставити пратећу запету као на примеру:

```
oneValueTuple = (42, )
```

Уколико заборавите запету, Пајтон неће моћи да разликује уређену торку и заграде које се користе да замене редослед операција. Пример:

```
variableA = (5 * 6)
variableB = (5 * 6, )
```

### Претварање између листи и уређених торки

Могуће је претварање листи у уређене торке и обратно, као што би претворили стринг у цео број и обратно. Само је потребно да проследимо вредност уређене торке `list()` функцији и она ће нам вратити листу од вредности уређене торке. Исто тако, могуће је проследити вредности листе функцији `tuple()` и она ће нам вратити уређену торку вредности листе.

Пробајте следеће команде у „shell“-у:

```
>>> spam = (1, 2, 3, 4)
>>> spam = list(spam)
>>> spam
[1, 2, 3, 4]
>>> spam = tuple(spam)
>>> spam
(1, 2, 3, 4)
>>>
```

### Структуре података и 2D листе

```
58.     mainBoard = getRandomizedBoard()
59.     revealedBoxes = generateRevealedBoxesData(False)
```

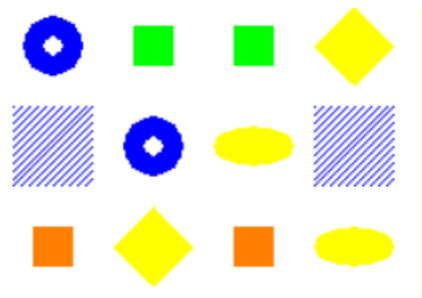
Функција `getRandomizedBoard()` враћа структуру података која представља стање на табли. `generateRevealedBoxesData()` функција враћа структуру података помоћу које видимо који од квадратића су сакривени. Повратне вредности ове две функције су дводимензионалне листе, или листе листи.

Ако имамо листу сачувану у променљивој под називом „spam“, можемо да приступимо члановима те листе са угластим заградама, као на пример `spam[2]` како би приступили трећем члану листе. Ако је вредност у `spam[2]` такође листа, можемо користити још један пар угластих заграда како би приступили вредности у тој листи. У том случају `spam[2][4]` би враћало пету вредност у листи која је трећа вредност у листи `spam`. Користећи ову нотацију у игри, сличици на позицији (4, 5) би могли да приступимо изразом `mainBoard[4][5]`. Пошто су иконице



чуване као уређени парови са обликом и бојом, целокупна структура података је листа листи уређених парова.

Ево малог примера. Рецимо да табла изгледа овако:



Одговарајућа структура података би била:

```
mainBoard = [[(DONUT, BLUE), (LINES, BLUE), (SQUARE, ORANGE)], [(SQUARE, GREEN), (DONUT, BLUE), (DIAMOND, YELLOW)], [(SQUARE, GREEN), (OVAL, YELLOW), (SQUARE, ORANGE)], [(DIAMOND, YELLOW), (LINES, BLUE), (OVAL, YELLOW)]]
```

У међувремену, структура података откривених поља је такође 2D листа, осим што се уместо уређених парова чува само Boolean вредност: True ако је квадратић у (x, y) координати откривен и False уколико је сакривен. Ако проследимо False функцији `generateRevealedBoxesData()` постављамо све Boolean вредности на False.

**Ове две структуре података користимо да би могли да пратимо стања табле за игру.**

#### Animacija „Започни игру“

```
61.     firstSelection = None # stores the (x, y) of the first box
        clicked.
62.
63.     DISPLAYSURF.fill(BGCOLOR)
64.     startGameAnimation(mainBoard)
```

У линији 61 постављамо променљиву са називом `firstSelection` и додељујемо јој вредност `None` (вредност која представља недостатак стања). Када играч кликне на сличицу на табли, програм мора да испрати да ли је кликнуто на прву или другу сличицу пара. Ако је `firstSelection None`, кликнуто је на прву иконицу и смештамо (X, Y) координату у променљиву `firstSelection` ако уређени пар два цела броја. Када се други пут кликне, вредност ће бити овај уређени пар, помоћу кога ће програм испратити да је у питању други клик на сличицу. У линији 63 попуњавамо целокупну подлогу са бојом позадине. Овим потезом ћемо такође обојити све што је постојало на подлози, чиме добијамо чисто „платно“ на коме можемо исцртавати графичке елементе.

Уколико сте играли Игру меморије, приметићете да су на почетку игре сви квадратићи на брзину сакривени и откривени насумично да би играчу омогућили да краичком ока види које су сличице под којим квадратићем. Ово се све дешава у функцији `startGameAnimation()`, која ће бити објашњена касније.

Битно је дозволити играчу да види накратко где се које сличице налазе зато што у супротном не би имали идеју где се било која од сличица налази. Кликтање на слепо није забавно толико као када се омогући мала помоћ.

### Петља игре

```
66.     while True: # main game loop
67.         mouseClicked = False
68.
69.         DISPLAYSURF.fill(BG_COLOR) # drawing the window
70.         drawBoard(mainBoard, revealedBoxes)
```

Петља игре је уствари бесконачна петља која почиње на линији 66 и која пролази кроз „for“ петљи све док игра траје. Запамтите да петља игре управља догађајима, мења стање игре, и исцртава стање игре на екрану.

Стање игре за Игру меморије програм чува кроз следеће променљиве:

- mainBoard
- revealedBoxes
- firstSelection
- mouseClicked
- mousex
- mousey

Svakim prolaskom kroz petlju igre u programu, mouseClicked промењива чува Boolean вредност која је True уколико је играч кликнуо при проласку кроз петљу игре.

На линији 69, подлога је обојена бојом позадине да би се обрисало све што је претходно нацртано на њој. Програм тада позива drawBoard() да би се исцртало тренутно стање табле.

Запамтите да наша функција за цртање само исцртава у меморију, а за приказ на екрану је потребно да позовемо pygame.display.update(), која се извршава на линији 121.

### Петља за управљање догађајима

```
72.     for event in pygame.event.get(): # event handling loop
73.         if event.type == QUIT or (event.type == KEYUP and
event.key == K_ESCAPE):
74.             pygame.quit()
75.             sys.exit()
76.         elif event.type == MOUSEMOTION:
77.             mousex, mousey = event.pos
78.         elif event.type == MOUSEBUTTONDOWN:
79.             mousex, mousey = event.pos
80.             mouseClicked = True
```

„For“ петља на линији 72 извршава код за сваки догађај који се догодио од претходног проласка кроз петљу игре. Ова петља се назива петља за управљање догађајима и која пролази кроз листу pygame.Event објеката који су повратна вредност функције pygame.event.get().

Ако је тип догађаја QUIT или KEYUP Esc тастера, тада би програм требао да прекине са извршавањем. У супротном, у случају MOUSEMOTION догађаја или MOUSEBUTTONUP догађаја, позиција показивача би требала да се сачува у mousex и mousey promenljivima. Ако је догађај MOUSEBUTTONUP, mouseClicked се такође поставља на True.

Када смо прихватили све догађаје, вредности сачуване у mousex, mousey и mouseClicked ће нам рећи који је улаз играч проследио. Сада је потребно да изменимо стање игре и да исцртамо резултате на екрану.

### Провера изнад којег квадратића је показивач

```
82.         boxx, boxy = getBoxAtPixel(mousex, mousey)
83.         if boxx != None and boxy != None:
84.             # The mouse is currently over a box.
85.             if not revealedBoxes[boxx][boxy]:
86.                 drawHighlightBox(boxx, boxy)
```

Функција getBoxAtPixel() ће враћати уређени пар целих бројева. Цели бројеви представљају XY координате квадратића на којем се тренутно налази показивач миша. Како ради ова функција је објашњено касније. Све што је потребно да знамо за сада је да уколико су mousex и mousey координате изнад квадратића, уређени пар XY координата су повратна вредност функције и чувају се у boxx и boxy. Уколико показивач миша није изнад неког од квадратића, у том случају је повратна вредност функције уређени пар недефинисаних вредности и та вредност функције и чувају се у None ће бити смештено у boxx и boxy.

Ми смо заинтересовани само ако boxx и boxy имају конкретне вредности смештене у њима. Следећих пар линија кода се извршавају у блоку if уколико је то случај. Када год се показивач миша налази изнад квадратића означимо га плавом бојом око квадратића. Бојење се врши функцијом drawHighlightBox().

```
87.         if not revealedBoxes[boxx][boxy] and mouseClicked:
88.             revealBoxesAnimation(mainBoard, [(boxx, boxy)])
89.             revealedBoxes[boxx][boxy] = True # set the box as
"revealed"
```

На линији 87 не проверавано само да ли је миш изнад квадратића, већ и да ли је кликнут. У том случају желимо да пустимо анимацију откривања тог квадратића тако што ћемо позвати функцију revealBoxesAnimation(). Требало би да обратимо пажњу да позивањем ове функције само исцртавамо анимацију откривања, а тек на линији 89 постављамо вредност revealedBoxes[boxx][boxy] на True.

### Управљање кликом на први квадратић

```
90.          if firstSelection == None: # the current box was the
first box clicked
91.              firstSelection = (boxx, boxy)
92.          else: # the current box was the second box clicked
93.              # Check if there is a match between the two
icons.
94.              icon1shape, icon1color =
getShapeAndColor(mainBoard, firstSelection[0], firstSelection[1])
95.              icon2shape, icon2color =
getShapeAndColor(mainBoard, boxx, boxy)
```

Пре него што се изврши улазак у пељу игре, вредност променљиве `firstSelection` је недефинисана. Наш програм ће то превести тако да није кликнуто ни на један квадратић, тако да уколико је услов на линији 90 `True`, то значи да је то први од два квадратића. Ми желимо да покренемо анимацију за дати квадратић и да држимо то поље отворено. Такође постављамо `firstSelection` променљиву на уређени пар координата за квадратић који је кликнут.

Ако је у питању други квадратић на који је играч кликнуо, желимо да пустимо анимацију и за тај квадрат и да након тога проверимо да ли су две сличице исте. `getShapeAndColor()` функција ће нам доставити вредности облика и боље сличице.

### Управљање упаривањем различитих сличица

```
97.          if icon1shape != icon2shape or icon1color !=
icon2color:
98.              # Icons don't match. Re-cover up both
selections.
99.              pygame.time.wait(1000) # 1000 milliseconds =
1 sec
100.             coverBoxesAnimation(mainBoard,
[(firstSelection[0], firstSelection[1]), (boxx, boxy)])
101.             revealedBoxes[firstSelection[0]]
[firstSelection [1]] = False
102.             revealedBoxes[boxx][boxy] = False
```

На линији 97 проверавамо да ли су облик и боја две сличице различити. Уколико јесу, тада хоћемо да паузирамо игру на 1000 милисекунди тако што позивамо `pygame.time.wait(1000)`. Овим дајемо играчу времена да запамти које две иконице се не слажи и где се налазе. Након тога покрећемо анимацију за скривање обе сличице и мењамо стање игре тако да показује да су ова два поља сакривена.

## Управљање победом играча

```
103.             elif hasWon(revealedBoxes): # check if all pairs
found
104.             gameWonAnimation(mainBoard)
105.             pygame.time.wait(2000)
106.
107.             # Reset the board
108.             mainBoard = getRandomizedBoard()
109.             revealedBoxes =
generateRevealedBoxesData(False)
110.
111.             # Show the fully unrevealed board for a
second.
112.             drawBoard(mainBoard, revealedBoxes)
113.             pygame.display.update()
114.             pygame.time.wait(1000)
115.
116.             # Replay the start game animation.
117.             startGameAnimation(mainBoard)
118.             firstSelection = None # reset firstSelection
variable
```

Уколико услов на линији 97 није задовоље, то значи да су две сличице исте. Програм ништа више не мора да ради са датим пољима, потребно је да остану откривена. Међутим, програм треба да провери да ли је то био последњи неоткривени пар на табли. То се проверава у функцији `hasWon()` која има повратну вредност `True` уколико је то случај.

Уколико је играч победио, желимо да пустимо анимацију за победу позивом `gameWonAnimation()`, затим да на кратко паузирамо извршавање програма и на послетку да ресетујемо структуре како би могли да започнемо нову игру.

На линији 117 пуштамо анимацију за почетак игре изнова. Након тога програм ће се извршавати у петљи као и обично, а играч може да настави са игром све док не изађе из програма.

Без обзира да ли се две сличице подударају или не, након што је играч кликнуо на други квадратић поставићемо вредност `firstSelection` променљиве на недефинисано тако да ће у сваком случају следећи квадратић на који играч кликне бити први квадрат који је кликнут.

## Цртање стања игре на екрану

```
120.             # Redraw the screen and wait a clock tick.
121.             pygame.display.update()
122.             FPSLOCK.tick(FPS)
```

У овом тренутно, стање игре се мења у зависности од играчевог уноса. Овде долазимо до краја извршавања петље игре, па можемо да позовемо `pygame.display.update()` како би исцртали `DISPLAYSURF` подлогу на екрану.

На линији 9 смо поставили FPS константу на целобројну вредност 30, што значи да ћемо исцртавати сваку 1/30 секунде. У том случају се `pygame.display.update()` и цео код петље игре мора се извршити за мање од 33.3 милисекунде. Сваки новији рачуран то може лагано да оствари. Да би спречили да се програм извршава пребрзо, позивамо `tick()` методу `pygame.Clock` објекта да би зауставили програм на 33.3 милисекунде.

### Pravljenje „Revealed Boxes“ структуре података

```
125. def generateRevealedBoxesData(val):
126.     revealedBoxes = []
127.     for i in range(BOARDWIDTH):
128.         revealedBoxes.append([val] * BOARDHEIGHT)
129.     return revealedBoxes
```

Функција `generateRevealedBoxesData()` треба да направи листу `Boolean` вредности. Вредност ће бити она која је прослеђена функцији као `val` параметар. Требало би да започнемо структуру као празну листу у `revealedBoxes` променљивој.

Да би направили структуру података која има формат `revealedBoxes[x][y]`, потребно је да обезбедимо да унутрашња листа представља вертикалне колоне табле, а не хоризонталне редове. У супротном, структура података би имала формат `revealedBoxes[y][x]`.

У петљи ћемо правити колоне, а затим их додавати на `revealedBoxes`. Колоне се праве коришћењем репликације листе, тако да ће листа колона имати толико `val` вредности колико јој `BOARDHEIGHT` дозволи.

### Прављење структуре података табле: Корак 1 – Дај све могуће сличице

```
132. def getRandomizedBoard():
133.     # Get a list of every possible shape in every possible color.
134.     icons = []
135.     for color in ALLCOLORS:
136.         for shape in ALLSHAPES:
137.             icons.append( (shape, color) )
```

Структура података табле је само листа листи уређених парова, где сваки пар има две вредности: једна за боју сличице и друга за облик сличице. Али прављење ове структуре је мало компликованије. Потребно је да обезбедимо да имамо тачно толико сличица колико има квадратића на табли и такође да обезбедимо да постоје две и само две сличице истог типа.

Први корак је да направимо листу за сваку могућу комбинацију боје и облика. Сетимо се да имамо листу свих боја и облика у `ALLCOLORS` и `ALLSHAPES`, тако да угњежђеном петљом на линијама 135 и 136 пролазимо кроз сваки могућу облик сваке могуће боје. Све ово је додато у листу променљиве иконс на линији 137.

## Корак 2 – Мешање и скраћивање листе свих сличица

```
139.     random.shuffle(icons) # randomize the order of the icons list
140.     numIconsUsed = int(BOARDWIDTH * BOARDHEIGHT / 2) # calculate how
many icons are needed
141.     icons = icons[:numIconsUsed] * 2 # make two of each
142.     random.shuffle(icons)
```

Запамтимо, можда постоји више комбинација него поља на табли. Потребно је да израчунамо број поља на табли тако што ћемо помножити BOARDWIDTH и BOARDHEIGHT. Затим ћемо поделити тај број са 2 пошто ћемо имати парове сличица. Овај број ћемо чувати у numIconsUsed.

На линији 141 користимо резање листе како би узели први numIconsUsed број сличица из листе. Ова листа је измешана на линији 139, тако да нећемо користити исте сличице у свакој игри. Затим је листа дуплирана коришћењем оператора \* како би постојале по две од сваке сличице. Ова нова дуплирана листа ће пребрисати стару листу у icons променљивој. Пошто је прва половина ове листе идентична са другом, позваћемо shuffle() методу поново како и измешали редослед сличица.

## Корак 3 – Посављање сличица на таблу

```
144.     # Create the board data structure, with randomly placed icons.
145.     board = []
146.     for x in range(BOARDWIDTH):
147.         column = []
148.         for y in range(BOARDHEIGHT):
149.             column.append(icons[0])
150.             del icons[0] # remove the icons as we assign them
151.         board.append(column)
152.     return board
```

Сада је потребно да направимо листу листу структура за таблу. Ово можемо урадити угњежденим петљама као код generateRevealedBoxes() функције. За сваку колону на табли, направимо листу насумично изабраних сличица. Како будемо додавали сличице колонама на линији 149, такоћемо их брисати са почетка icons листе на линији 150.



Да би објаснили ово боље, укуцајте следећи код у „shell“-у. Приметите како `del` команда мења `myList` листу.

```
>>> myList = ['cat', 'dog', 'mouse', 'lizard']
>>> del myList[0]
>>> myList
['dog', 'mouse', 'lizard']
>>> del myList[0]
>>> myList
['mouse', 'lizard']
>>> del myList[0]
>>> myList
['lizard']
>>> del myList[0]
>>> myList
[]
>>>
```

### Дељење листе у листу листи

```
155. def splitIntoGroupsOf(groupSize, theList):
156.     # splits a list into a list of lists, where the inner lists have
    at
157.     # most groupSize number of items.
158.     result = []
159.     for i in range(0, len(theList), groupSize):
160.         result.append(theList[i:i + groupSize])
161.     return result
```

Функција `splitIntoGroupsOf()` дели листу на листу листи, где унутрашње листе имају `groupSize` број елемената у њима.

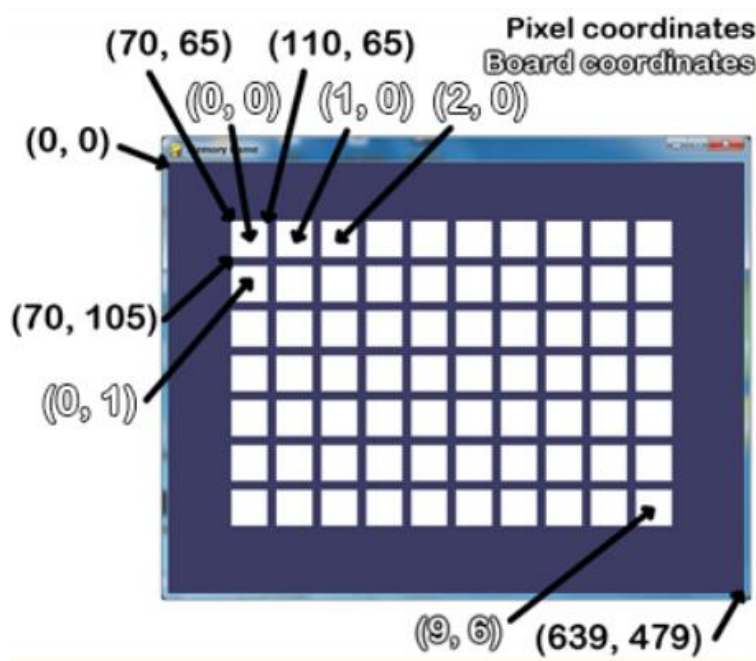
Дељење листе на линији 160 са `theList[i:i + groupSize]` прави листе које се додају резулт листи. Кроз сваки пролаз где је и 0, 8 и 16, овај израз за дељење листе би био `theList[0:8]`, па `theList[8:16]` у другом проласку, и `theList[16:24]` у трећем проласку.

Приметимо да иако би највећи индекс у нашем примеру био 19, `theList[16:24]` не би избацивала грешку иако је 24 веће од 19. Само ће направити део листе са остатком сличица у листи. Дељење листи не уништава или мења првобитну листу сачувану у `theList`. Само копира њен део и прави вредности нове листе. Ова нова листа је додата листи `result` на линији 160. Тако да када вратимо `result` на крају функције, враћамо листу листи.

### Дефинисање координатног система

```
164. def leftTopCoordsOfBox(boxx, boxy):  
165.     # Convert board coordinates to pixel coordinates  
166.     left = boxx * (BOXSIZE + GAPSIZE) + XMARGIN  
167.     top = boxy * (BOXSIZE + GAPSIZE) + YMARGIN  
168.     return (left, top)
```

Ево слике игре и два различита координатна система које користимо. Прозор је широк 640 пиксела и висок 480 пиксела тако да је (639, 479) је координата доњег десног ћошка, док је горњи леви (0, 0).



`leftTopCoordsOfBox()` функција ће примити координате квадратића и вратити координате у пикселима. Пошто квадратић заузима више пиксела на екрану, увек ћемо враћати координате пиксела горњег левог угла квадратића. Ова вредност ће се враћати као уређена двојка. Дата функција ће често бити коришћења за исцртавање ових квадратића.

## Претварање из пиксел координата у квадратић координате

```
171. def getBoxAtPixel(x, y):
172.     for boxx in range(BOARDWIDTH):
173.         for boxy in range(BOARDHEIGHT):
174.             left, top = leftTopCoordsOfBox(boxx, boxy)
175.             boxRect = pygame.Rect(left, top, BOXSIZE, BOXSIZE)
176.             if boxRect.collidepoint(x, y):
177.                 return (boxx, boxy)
178.     return (None, None)
```

Биће нам потреба функција која ће претварати пиксел координате у координате квадратића. Објекти типа правоуганик имају методу `collidepoint()` којој можемо проследити X Y координате и она ће нам вратити `True` ако су координате унутар правоуганика.

Да би пронашли на ком квадратићу се налазе координате показивача миша, мораћемо да прођемо кроз сваки квадратић и да позовемо `collidepoint()` са тим координатама као параметрима. Када метод врати `True` знамо да смо пронашли квадратић на који је кликнуто или на којем се налази показивач миша, па можемо наћи и координате квадратића. Уколико се то не деси, то значи да се показивач не налази ни на једном квадратићу и онда враћамо недефинисане вредности.

## Цртање сличица и синтактички шећер

```
181. def drawIcon(shape, color, boxx, boxy):
182.     quarter = int(BOXSIZE * 0.25) # syntactic sugar
183.     half = int(BOXSIZE * 0.5) # syntactic sugar
184.
185.     left, top = leftTopCoordsOfBox(boxx, boxy) # get pixel coords
186.     from board coords
```

Функција `drawIcon()` ће цртати сличицу на месту чије су координате дате у `boxx` и `boxy` параметрима. Сваки могући облик позива другачије Пајтејм функције, тако да морамо имати велики број `if` и `elif` наредби да би их користили.

Координатама X и Y горње и леве границе квадратића се могу добити позивањем функције `leftTopCoordsOfBox()`. Ширина и висина квадратића су дефинисани у `BOXSIZE` константи.

Овакве променљиве су пример синтактичког шећера. Синтактички шећер је када додајемо код који би могао написан другачије, али али омогућава да изворни код буде читљивији. Константе су један вид синтактичког шећера. Нема потребе да имамо променљиве за четвртину и половину, али када их имамо чинимо код читљивијим. Код које лак за читање је лак за дебаговање и мењање у будућности.

```

186.     # Draw the shapes
187.     if shape == DONUT:
188.         pygame.draw.circle(DISPLAYSURF, color, (left + half, top +
half), half - 5)
189.         pygame.draw.circle(DISPLAYSURF, BGCOLOR, (left + half, top +
half), quarter - 5)
190.     elif shape == SQUARE:
191.         pygame.draw.rect(DISPLAYSURF, color, (left + quarter, top +
quarter, BOXSIZE - half, BOXSIZE - half))
192.     elif shape == DIAMOND:
193.         pygame.draw.polygon(DISPLAYSURF, color, ((left + half, top),
(left + BOXSIZE - 1, top + half), (left + half, top + BOXSIZE - 1), (left,
top + half)))
194.     elif shape == LINES:
195.         for i in range(0, BOXSIZE, 4):
196.             pygame.draw.line(DISPLAYSURF, color, (left, top + i),
(left + i, top))
197.             pygame.draw.line(DISPLAYSURF, color, (left + i, top +
BOXSIZE - 1), (left + BOXSIZE - 1, top + i))
198.     elif shape == OVAL:
199.         pygame.draw.ellipse(DISPLAYSURF, color, (left, top + quarter,
BOXSIZE, half))

```

### Синтактички шећер са узивањем облика и боје сличица на табли

Функција `getShapeAndColor()` има само једну линију кода. Можете да се питате зашто онда користимо функцију уместо да само искуцамо ту линију кода сваки пут кад нам треба. То се такође користи ради читљивости кода.

Лако је закључити шта код као `shape, color = getShapeAndColor()` ради. Али ако би гледали кода као `shape, color = board[boxx][boxy][0], board[boxx][boxy][1]`, било би мало конфузно.

## Цртање квадратића

```
208. def drawBoxCovers(board, boxes, coverage):
209.     # Draws boxes being covered/revealed. "boxes" is a list
210.     # of two-item lists, which have the x & y spot of the box.
211.     for box in boxes:
212.         left, top = leftTopCoordsOfBox(box[0], box[1])
213.         pygame.draw.rect(DISPLAYSURF, BGCOLOR, (left, top, BOXSIZE,
BOXSIZE))
214.         shape, color = getShapeAndColor(board, box[0], box[1])
215.         drawIcon(shape, color, box[0], box[1])
216.         if coverage > 0: # only draw the cover if there is an
coverage
217.             pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top,
coverage, BOXSIZE))
218.     pygame.display.update()
219.     FPSLOCK.tick(FPS)
```

Функција `drawBoxCovers()` има три параметра: структуру табле, листу (X, Y) уређених парова за сваки квадрат који треба да се нацрта, и простор који треба да прекрије квадрат.

Пошто желимо да користимо исти код за цртање за сваки квадратић, користићемо петљу на линији 211 тако да извршава исти код за сваки квадратић. Унутар петље, код би требао да ради три ствар: црта боју позадине, црта иконицу, и квадрат који покрива иконицу. За цртање квадратића ћемо користити `pygame.draw.rect()`. Уколико се не задовољи услов на линији 216, нећемо позивати функцију за цртање квадратића.

Функција `drawBoxCover()` ће бити позивана из засебне петље. Због тога, мора да има свој позив `pygame.display.update()` и `FPSCLOCK.tick()` да би приказао анимацију.

## Управљање анимацијом откривања и сакривања

```
222. def revealBoxesAnimation(board, boxesToReveal):
223.     # Do the "box reveal" animation.
224.     for coverage in range(BOXSIZE, (-REVEALSPEED) - 1, -
REVEALSPEED):
225.         drawBoxCovers(board, boxesToReveal, coverage)
226.
227.
228. def coverBoxesAnimation(board, boxesToCover):
229.     # Do the "box cover" animation.
230.     for coverage in range(0, BOXSIZE + REVEALSPEED, REVEALSPEED):
231.         drawBoxCovers(board, boxesToCover, coverage)
```

Сетимо се да је анимација само приказивање различитих слика у кратким тренуцима, и тако се чини да се ствари крећу по екрану. `revealBoxesAnimation()` и `coverBoxesAnimation()` само требају да нацртају иконицу са различитим покривањем белог квадрата. Можемо написати једну функцију и назвати је `drawBoxCovers()` која може да обави то, а да наша анимација позива ту функцију за сваки фрејм анимације.

Да би урадили ово, поставићемо петљу са увећавањем или смањивањем цовераге променливе. На линији 12 смо поставили константу REVEALSPEED на 8, што значи да ће се квадратић смањивати/повећавати за 8 пиксела у сваком проласку кроз петљу.

### Цртање целе табле

```
234. def drawBoard(board, revealed):
235.     # Draws all of the boxes in their covered or revealed state.
236.     for boxx in range(BOARDWIDTH):
237.         for boxy in range(BOARDHEIGHT):
238.             left, top = leftTopCoordsOfBox(boxx, boxy)
239.             if not revealed[boxx][boxy]:
240.                 # Draw a covered box.
241.                 pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top,
BOXSIZE, BOXSIZE))
242.             else:
243.                 # Draw the (revealed) icon.
244.                 shape, color = getShapeAndColor(board, boxx, boxy)
245.                 drawIcon(shape, color, boxx, boxy)
```

drawBoard() функција позива drawIcon() за сваки квадратић на табли. Угњеждена петља на линијама 236 и 237 ће се вртети кроз сваку могућу X и Y координату за квадратиће, и или ће нацртати икону на локацији или нацртати бели квадратић уместо тога.

### Цртање обележавања

```
248. def drawHighlightBox(boxx, boxy):
249.     left, top = leftTopCoordsOfBox(boxx, boxy)
250.     pygame.draw.rect(DISPLAYSURF, HIGHLIGHTCOLOR, (left - 5, top - 5,
BOXSIZE + 10, BOXSIZE + 10), 4)
```

Да би помогли играчу да разазна да могу да кликну на квадратић да би га открили, направимо плави оквир око квадратића да би га обележили. Оквир цитамо позивом pygame.draw.rect() да би направили квадрат са ширином 4 пиксела.

### Анимација почетка игре

```
253. def startGameAnimation(board):
254.     # Randomly reveal the boxes 8 at a time.
255.     coveredBoxes = generateRevealedBoxesData(False)
256.     boxes = []
257.     for x in range(BOARDWIDTH):
258.         for y in range(BOARDHEIGHT):
259.             boxes.append( (x, y) )
260.     random.shuffle(boxes)
261.     boxGroups = splitIntoGroupsOf(8, boxes)
```

Анимација која се приказује на почетку игре даје играчу малу помоћ о томе где се налазе које сличице. Да би направили ову анимацију морамо да сакривамо и откивамо групе квадратића једну по једну.

Отриваћемо 8 по 8 квадратића редом. Да би променили квадратиће сваки пут кад игра почне, користићемо `random.shuffle()` функцију. Да би добили листу од 8 квадратића, позиваћемо `splitIntoGroupsOf()` и прослеђиваћемо јој 8 и листу квадратића. Листу листе која ће бити повратна вредност ћемо сместити у `boxGroups`.

### Откривање и сакривање група квадратића

```
263.     drawBoard(board, coveredBoxes)
264.     for boxGroup in boxGroups:
265.         revealBoxesAnimation(board, boxGroup)
266.         coverBoxesAnimation(board, boxGroup)
```

На почетку, нацртамо табу. Пошто је свака вредност у `coveredBoxes` постављена на `False`, позив `drawBoard()` ће цртати само квадратиће који покривају. Анимације са откривање и сакривање ће цртати преко места тих белих квадратића.

Петљом ћемо проћи кроз сваку од унутрашних листи листе `boxGroups`. Функцијама за анимације ће приказивати сличице скривене испод, односно сакривати их белим квадратићима који се шире.



### Анимација „Победили сте“

```
269. def gameWonAnimation(board):
270.     # flash the background color when the player has won
271.     coveredBoxes = generateRevealedBoxesData(True)
272.     color1 = LIGHTBGCOLOR
273.     color2 = BGCOLOR
274.
275.     for i in range(13):
276.         color1, color2 = color2, color1 # swap colors
277.         DISPLAYSURF.fill(color1)
278.         drawBoard(board, coveredBoxes)
279.         pygame.display.update()
280.         pygame.time.wait(300)
```

Када је играч открио сва поља и упарио их, желимо да му честитамо тако што ћемо мењати позадинску боју. У for петљи ћемо исцртавати боју у color1 променљивој за боју позадине и онда цртати таблу преко ње. Међутима при сваком проласку кроз петљу, вредности color1 и color2 ће се мењати међусобно на линији 276. Овако ће програм смењивати боју позадине и табле.

Ова функција мора да позове pygame.display.update() да би анимацију приказали на екрану.

### Саопштавање играчу да је победио

```
283. def hasWon(revealedBoxes):
284.     # Returns True if all the boxes have been revealed, otherwise
285.     False
286.     for i in revealedBoxes:
287.         if False in i:
288.             return False # return False if any boxes are covered.
289.     return True
```

Играч је победио у игри када је упарио све сличице. Можемо проћи кроз свако поље на табли и уколико у revealedBoxes не наиђемо на False вредност, то значи да је играч победио.

Пошто је revealedBoxes листа листе, петља на линији 285 ће поставити параметар унутрашње листе као вредности од i. Али можемо да користимо in оператор да пронађемо False вредност у целој унутрашњој листи. На овај начин нећемо морати да додајемо још једну петљу.

```
for x in revealedBoxes:
    for y in revealedBoxes[x]:
        if False == revealedBoxes[x][y]:
            return False
```