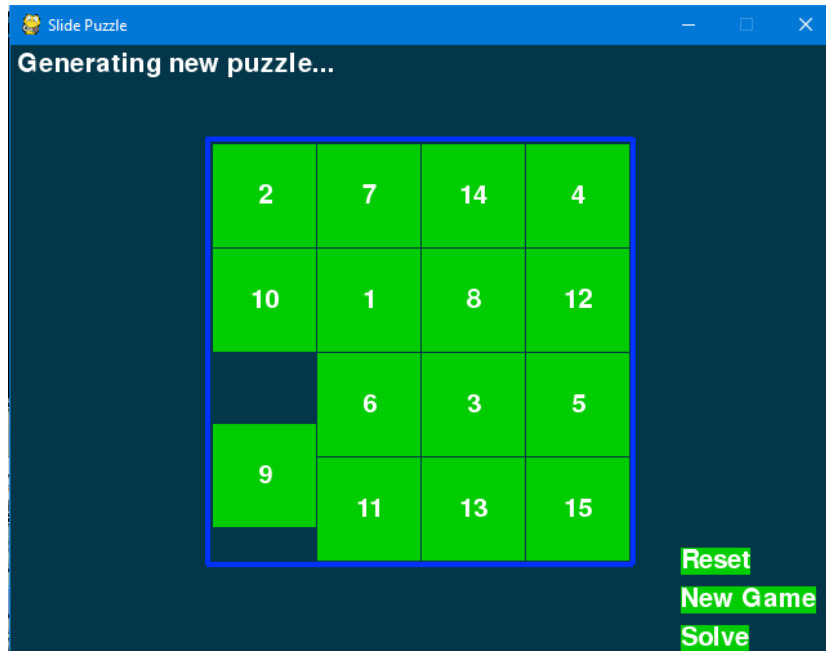


Клизећа слагалица



Како се игра клизећа слагалица

Табла за игру је мрежа 4x4 са петнаест поља означених бројевима од 1 до 15 слева на десно и једним празним пољем. Бројеви на пољима започињу да се ређају случајним избором, а играч помера поља унаоколо све док поља не буду поново у свом оригиналном распореду.

Изворни код клизеће слагалице

Овај изворни код се може преузети са <http://inventwithpython.com/slidepuzzle.py> . Ако добијеш неку поруку о грешци, погледај број линије који је споменут у поруци о грешци и провери да ли има грешака у куцању у твом коду. Такође, можеш и да ископираш и налепиш свој код у веб образац на <http://inventwithpython.com/diff/slidepuzzle> да провериш да ли постоје разлике између твог кода и кода у књизи.

```
1 # Slide Puzzle
2 # By Al Sweigart al@inventwithpython.com
3 # http://inventwithpython.com/pygame
4 # Released under a "Simplified BSD" license
5
6 import pygame, sys, random
7 from pygame.locals import *
8
9 # Create the constants (go ahead and experiment with different values)
10 BOARDWIDTH = 4 # number of columns in the board
11 BOARDHEIGHT = 4 # number of rows in the board
12 TILESIZE = 80
13 WINDOWWIDTH = 640
14 WINDOWHEIGHT = 480
15 FPS = 30
16 BLANK = None
```

```

17
18 #           R      G      B
19 BLACK =      ( 0,   0,   0)
20 WHITE =      (255, 255, 255)
21 BRIGHTBLUE = ( 0,   50, 255)
22 DARKTURQUOISE = ( 3,  54, 73)
23 GREEN =      ( 0, 204,   0)
24
25 BGCOLOR = DARKTURQUOISE
26 TILECOLOR = GREEN
27 TEXTCOLOR = WHITE
28 BORDERCOLOR = BRIGHTBLUE
29 BASICFONTSIZE = 20
30
31 BUTTONCOLOR = WHITE
32 BUTTONTXTCOLOR = BLACK
33 MESSAGECOLOR = WHITE
34
35 XMARGIN = int((WINDOWWIDTH - (TILESIZE * BOARDWIDTH + (BOARDWIDTH - 1))) / 2)
36 YMARGIN = int((WINDOWHEIGHT - (TILESIZE * BOARDHEIGHT + (BOARDHEIGHT - 1))) / 2)
37
38 UP = 'up'
39 DOWN = 'down'
40 LEFT = 'left'
41 RIGHT = 'right'
42
43 def main():
44     global FPSCLOCK, DISPLAYSURF, BASICFONT, RESET_SURF, RESET_RECT, NEW_SURF, \
45         NEW_RECT, SOLVE_SURF, SOLVE_RECT
46
47     pygame.init()
48     FPSCLOCK = pygame.time.Clock()
49     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
50     pygame.display.set_caption('Slide Puzzle')
51     BASICFONT = pygame.font.Font('freesansbold.ttf', BASICFONTSIZE)
52
53     # Store the option buttons and their rectangles in OPTIONS.
54     RESET_SURF, RESET_RECT = makeText('Reset', TEXTCOLOR, TILECOLOR, \
55         WINDOWWIDTH - 120, WINDOWHEIGHT - 90)
56     NEW_SURF, NEW_RECT = makeText('New Game', TEXTCOLOR, TILECOLOR, \
57         WINDOWWIDTH - 120, WINDOWHEIGHT - 60)
58     SOLVE_SURF, SOLVE_RECT = makeText('Solve', TEXTCOLOR, TILECOLOR, \
59         WINDOWWIDTH - 120, WINDOWHEIGHT - 30)
60
61     mainBoard, solutionSeq = generateNewPuzzle(80)
62     SOLVEDBOARD = getStartingBoard() # a solved board is the same as the
63         # board in a start state.
64     allMoves = [] # list of moves made from the solved configuration
65
66     while True: # main game loop
67         slideTo = None # the direction, if any, a tile should slide
68         msg = 'Click tile or press arrow keys to slide.' # contains the
69             # message to show in the upper left corner.
70         if mainBoard == SOLVEDBOARD:
71             msg = 'Solved!'
72
73         drawBoard(mainBoard, msg)
74
75         checkForQuit()
76         for event in pygame.event.get(): # event handling loop
77             if event.type == MOUSEBUTTONUP:
78                 spotx, spoty = getSpotClicked(mainBoard, event.pos[0], \
79                     event.pos[1])
80
81                 if (spotx, spoty) == (None, None):
82                     # check if the user clicked on an option button
83                     if RESET_RECT.collidepoint(event.pos):
84                         resetAnimation(mainBoard, allMoves) # clicked on

```

```

85                                     #Reset button
86         allMoves = []
87         elif NEW_RECT.collidepoint(event.pos):
88             mainBoard, solutionSeq = generateNewPuzzle(80)
89             # clicked on New Game button
90             allMoves = []
91         elif SOLVE_RECT.collidepoint(event.pos):
92             resetAnimation(mainBoard, solutionSeq + allMoves)
93             # clicked on Solve button
94             allMoves = []
95     else:
96         # check if the clicked tile was next to the blank spot
97
98         blankx, blanky = getBlankPosition(mainBoard)
99         if spotx == blankx + 1 and spoty == blanky:
100             slideTo = LEFT
101         elif spotx == blankx - 1 and spoty == blanky:
102             slideTo = RIGHT
103         elif spotx == blankx and spoty == blanky + 1:
104             slideTo = UP
105         elif spotx == blankx and spoty == blanky - 1:
106             slideTo = DOWN
107
108     elif event.type == KEYUP:
109         # check if the user pressed a key to slide a tile
110         if event.key in (K_LEFT, K_a) and isValidMove(mainBoard, LEFT):
111             slideTo = LEFT
112         elif event.key in (K_RIGHT, K_d) and isValidMove(mainBoard, RIGHT):
113             slideTo = RIGHT
114         elif event.key in (K_UP, K_w) and isValidMove(mainBoard, UP):
115             slideTo = UP
116         elif event.key in (K_DOWN, K_s) and isValidMove(mainBoard, DOWN):
117             slideTo = DOWN
118
119     if slideTo:
120         slideAnimation(mainBoard, slideTo, \
121             'Click tile or press arrow keys to slide.', 8)
122         # show slide on screen
123         makeMove(mainBoard, slideTo)
124         allMoves.append(slideTo) # record the slide
125         pygame.display.update()
126         FPSCLOCK.tick(FPS)
127
128
129 def terminate():
130     pygame.quit()
131     sys.exit()
132
133
134 def checkForQuit():
135     for event in pygame.event.get(QUIT): # get all the QUIT events
136         terminate() # terminate if any QUIT events are present
137     for event in pygame.event.get(KEYUP): # get all the KEYUP events
138         if event.key == K_ESCAPE:
139             terminate() # terminate if the KEYUP event was for the Esc key
140         pygame.event.post(event) # put the other KEYUP event objects back
141
142
143 def getStartingBoard():
144     # Return a board data structure with tiles in the solved state.
145     # For example, if BOARDWIDTH and BOARDHEIGHT are both 3, this function
146     # returns [[1, 4, 7], [2, 5, 8], [3, 6, BLANK]]
147     counter = 1
148     board = []
149     for x in range(BOARDWIDTH):
150         column = []
151         for y in range(BOARDHEIGHT):
152             column.append(counter)

```

```

153         counter += BOARDWIDTH
154         board.append(column)
155         counter -= BOARDWIDTH * (BOARDHEIGHT - 1) + BOARDWIDTH - 1
156
157     board[BOARDWIDTH-1][BOARDHEIGHT-1] = BLANK
158     return board
159
160
161 def getBlankPosition(board):
162     # Return the x and y of board coordinates of the blank space.
163     for x in range(BOARDWIDTH):
164         for y in range(BOARDHEIGHT):
165             if board[x][y] == BLANK:
166                 return (x, y)
167
168
169 def makeMove(board, move):
170     # This function does not check if the move is valid.
171     blankx, blanky = getBlankPosition(board)
172
173     if move == UP:
174         board[blankx][blanky], board[blankx][blanky+1] = board[blankx][blanky+1], \
175             board[blankx][blanky]
176     elif move == DOWN:
177         board[blankx][blanky], board[blankx][blanky-1] = board[blankx][blanky-1], \
178             board[blankx][blanky]
179     elif move == LEFT:
180         board[blankx][blanky], board[blankx+1][blanky] = board[blankx+1][blanky], \
181             board[blankx][blanky]
182     elif move == RIGHT:
183         board[blankx][blanky], board[blankx-1][blanky] = board[blankx-1][blanky], \
184             board[blankx][blanky]
185
186
187 def isValidMove(board, move):
188     blankx, blanky = getBlankPosition(board)
189     return (move == UP and blanky != len(board[0]) - 1) or \
190         (move == DOWN and blanky != 0) or \
191         (move == LEFT and blankx != len(board) - 1) or \
192         (move == RIGHT and blankx != 0)
193
194
195 def getRandomMove(board, lastMove=None):
196     # start with a full list of all four moves
197     validMoves = [UP, DOWN, LEFT, RIGHT]
198
199     # remove moves from the list as they are disqualified
200     if lastMove == UP or not isValidMove(board, DOWN):
201         validMoves.remove(DOWN)
202     if lastMove == DOWN or not isValidMove(board, UP):
203         validMoves.remove(UP)
204     if lastMove == LEFT or not isValidMove(board, RIGHT):
205         validMoves.remove(RIGHT)
206     if lastMove == RIGHT or not isValidMove(board, LEFT):
207         validMoves.remove(LEFT)
208
209     # return a random move from the list of remaining moves
210     return random.choice(validMoves)
211
212
213 def getLeftTopOfTile(tileX, tileY):
214     left = XMARGIN + (tileX * TILESIZE) + (tileX - 1)
215     top = YMARGIN + (tileY * TILESIZE) + (tileY - 1)
216     return (left, top)
217
218
219 def getSpotClicked(board, x, y):
220     # from the x & y pixel coordinates, get the x & y board coordinates

```

```

221     for tileX in range(len(board)):
222         for tileY in range(len(board[0])):
223             left, top = getLeftTopOfTile(tileX, tileY)
224             tileRect = pygame.Rect(left, top, TILESIZE, TILESIZE)
225             if tileRect.collidepoint(x, y):
226                 return (tileX, tileY)
227     return (None, None)
228
229
230 def drawTile(tilex, tiley, number, adjx=0, adjy=0):
231     # draw a tile at board coordinates tilex and tiley, optionally a few
232     # pixels over (determined by adjx and adjy)
233     left, top = getLeftTopOfTile(tilex, tiley)
234     pygame.draw.rect(DISPLAYSURF, TILECOLOR, (left + adjx, top + adjy, \
235                                                  TILESIZE, TILESIZE))
236     textSurf = BASICFONT.render(str(number), True, TEXTCOLOR)
237     textRect = textSurf.get_rect()
238     textRect.center = left + int(TILESIZE / 2) + adjx, top + int(TILESIZE / 2) + adjy
239     DISPLAYSURF.blit(textSurf, textRect)
240
241
242 def makeText(text, color, bgcolor, top, left):
243     # create the Surface and Rect objects for some text.
244     textSurf = BASICFONT.render(text, True, color, bgcolor)
245     textRect = textSurf.get_rect()
246     textRect.topleft = (top, left)
247     return (textSurf, textRect)
248
249
250 def drawBoard(board, message):
251     DISPLAYSURF.fill(BGCOLOR)
252     if message:
253         textSurf, textRect = makeText(message, MESSAGECOLOR, BGCOLOR, 5, 5)
254         DISPLAYSURF.blit(textSurf, textRect)
255
256     for tilex in range(len(board)):
257         for tiley in range(len(board[0])):
258             if board[tilex][tiley]:
259                 drawTile(tilex, tiley, board[tilex][tiley])
260
261     left, top = getLeftTopOfTile(0, 0)
262     width = BOARDWIDTH * TILESIZE
263     height = BOARDHEIGHT * TILESIZE
264     pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, \
265                      (left - 5, top - 5, width + 11, height + 11), 4)
266
267     DISPLAYSURF.blit(RESET_SURF, RESET_RECT)
268     DISPLAYSURF.blit(NEW_SURF, NEW_RECT)
269     DISPLAYSURF.blit(SOLVE_SURF, SOLVE_RECT)
270
271
272 def slideAnimation(board, direction, message, animationSpeed):
273     # Note: This function does not check if the move is valid.
274
275     blankx, blanky = getBlankPosition(board)
276     if direction == UP:
277         movex = blankx
278         movey = blanky + 1
279     elif direction == DOWN:
280         movex = blankx
281         movey = blanky - 1
282     elif direction == LEFT:
283         movex = blankx + 1
284         movey = blanky
285     elif direction == RIGHT:
286         movex = blankx - 1
287         movey = blanky
288

```

```

289     # prepare the base surface
290     drawBoard(board, message)
291     baseSurf = DISPLAYSURF.copy()
292     # draw a blank space over the moving tile on the baseSurf Surface.
293     moveLeft, moveTop = getLeftTopOfTile(movex, movey)
294     pygame.draw.rect(baseSurf, BG_COLOR, (moveLeft, moveTop, TILESIZE, TILESIZE))
295
296     for i in range(0, TILESIZE, animationSpeed):
297         # animate the tile sliding over
298         checkForQuit()
299         DISPLAYSURF.blit(baseSurf, (0, 0))
300         if direction == UP:
301             drawTile(movex, movey, board[movex][movey], 0, -i)
302         if direction == DOWN:
303             drawTile(movex, movey, board[movex][movey], 0, i)
304         if direction == LEFT:
305             drawTile(movex, movey, board[movex][movey], -i, 0)
306         if direction == RIGHT:
307             drawTile(movex, movey, board[movex][movey], i, 0)
308
309     pygame.display.update()
310     FPSCLOCK.tick(FPS)
311
312
313 def generateNewPuzzle(numSlides):
314     # From a starting configuration, make numSlides number of moves (and
315     # animate these moves).
316     sequence = []
317     board = getStartingBoard()
318     drawBoard(board, '')
319     pygame.display.update()
320     pygame.time.wait(500) # pause 500 milliseconds for effect
321     lastMove = None
322     for i in range(numSlides):
323         move = getRandomMove(board, lastMove)
324         slideAnimation(board, move, 'Generating new puzzle...', \
325                        animationSpeed=int(TILESIZE / 3))
326         makeMove(board, move)
327         sequence.append(move)
328         lastMove = move
329     return (board, sequence)
330
331
332 def resetAnimation(board, allMoves):
333     # make all of the moves in allMoves in reverse.
334     revAllMoves = allMoves[:] # gets a copy of the list
335     revAllMoves.reverse()
336
337     for move in revAllMoves:
338         if move == UP:
339             oppositeMove = DOWN
340         elif move == DOWN:
341             oppositeMove = UP
342         elif move == RIGHT:
343             oppositeMove = LEFT
344         elif move == LEFT:
345             oppositeMove = RIGHT
346         slideAnimation(board, oppositeMove, '', animationSpeed=int(TILESIZE/2))
347         makeMove(board, oppositeMove)
348
349
350 if __name__ == '__main__':
351     main()

```

Већи део кода је сличан онима које смо гледали у претходним играма, нарочито константе које се постављају на почетку самог кода.

```
1 # Slide Puzzle
2 # By Al Sweigart al@inventwithpython.com
3 # http://inventwithpython.com/pygame
4 # Released under a "Simplified BSD" license
5
6 import pygame, sys, random
7 from pygame.locals import *
8
9 # Create the constants (go ahead and experiment with different values)
10 BOARDWIDTH = 4 # number of columns in the board
11 BOARDHEIGHT = 4 # number of rows in the board
12 TILESIZE = 80
13 WINDOWWIDTH = 640
14 WINDOWHEIGHT = 480
15 FPS = 30
16 BLANK = None
17
18 #           R      G      B
19 BLACK =    ( 0,   0,   0)
20 WHITE =    (255, 255, 255)
21 BRIGHTBLUE = ( 0,  50, 255)
22 DARKTURQUOISE = ( 3,  54, 73)
23 GREEN =    ( 0, 204,  0)
24
25 BGCOLOR = DARKTURQUOISE
26 TILECOLOR = GREEN
27 TEXTCOLOR = WHITE
28 BORDERCOLOR = BRIGHTBLUE
29 BASICFONTSIZE = 20
30
31 BUTTONCOLOR = WHITE
32 BUTTONTXTCOLOR = BLACK
33 MESSAGECOLOR = WHITE
34
35 XMARGIN = int((WINDOWWIDTH - (TILESIZE * BOARDWIDTH + (BOARDWIDTH - 1)))) / 2)
36 YMARGIN = int((WINDOWHEIGHT - (TILESIZE * BOARDHEIGHT + (BOARDHEIGHT - 1)))) / 2)
37
38 UP = 'up'
39 DOWN = 'down'
40 LEFT = 'left'
41 RIGHT = 'right'
```

Овај код на врху програма само управља свим основним уносима модула и креирањем константи. Исти је као почетак игре Меморијска слагалица .

Подешавање тастера

```
43 def main():
44     global FPSCLOCK, DISPLAYSURF, BASICFONT, RESET_SURF, RESET_RECT, NEW_SURF, \
45         NEW_RECT, SOLVE_SURF, SOLVE_RECT
46
47     pygame.init()
48     FPSCLOCK = pygame.time.Clock()
49     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
50     pygame.display.set_caption('Slide Puzzle')
51     BASICFONT = pygame.font.Font('freesansbold.ttf', BASICFONTSIZE)
52     # Store the option buttons and their rectangles in OPTIONS.
53     RESET_SURF, RESET_RECT = makeText('Reset', TEXTCOLOR, TILECOLOR, \
54         WINDOWWIDTH - 120, WINDOWHEIGHT - 90)
```

```

56 NEW_SURF, NEW_RECT = makeText('New Game', TEXTCOLOR, TILECOLOR, \
57                               WINDOWWIDTH - 120, WINDOWHEIGHT - 60)
58 SOLVE_SURF, SOLVE_RECT = makeText('Solve', TEXTCOLOR, TILECOLOR, \
59                                   WINDOWWIDTH - 120, WINDOWHEIGHT - 30)
60
61 mainBoard, solutionSeq = generateNewPuzzle(80)
62 SOLVEDBOARD = getStartingBoard() # a solved board is the same as the
63                                  # board in a start state.

```

Баш као и у последњем поглављу, функције које се позивају преко функције `main()` ће бити објашњене у овом поглављу касније. За сада, само треба да знаш шта раде и које вредности оне враћају.

Први део `main()` функције ће управљати креирањем прозора, објекта `Clock`, и објекта `Font`. Функција креирања текста `makeText()` је дефинисана касније у програму, али за сада ти је потребно само да знаш да она враћа објекат пугаме. Рецт објекат се може употребити за креирање тастера на које се може кликнути. Игра Клизећа слагалица ће имати три тастера.

- Reset (Поново) који поништава све кораке које је играч направио,
- New (Нова игра) је тастер који креира нову игру,
- Solve (Решити) који ће решити слагалицу уместо играча.

Потребне су нам две структуре података за таблу у овом програму. Једна структура ће представљати тренутно стање игре. Друга структура ће имати поља у решеном стању, што значи да су сва поља поређана по реду. Када је структуру тренутног стања игре потпуно иста као решена, тада знамо да је играч победио. Решену структуру никада не мењамо, она је ту за поређене са структуром тренутног стања игре.

Табла на почетку игре има решену структуру. Функција `GenerateNewPuzzle()` има задатак да креира нову структуру тако што ће извршити 80 насумичних покрета померања картица. Уколико желиш да табла буде још више помешана у 61-вој и 88-мој линији кода тај број можеш повећати. То ће довести таблу у насумично помешано стање, које ће играч морати да реши што ће се чувати у варијабли названој `mainBoard`. `GenerateNewBoard()` враћа листу свих случајних покрета који су начињени на њој који ће бити сачувани у варијабли названој `solutionSeq`.

Буди паметан помоћу глупог кода

- Листа покрета до решене структуре

```

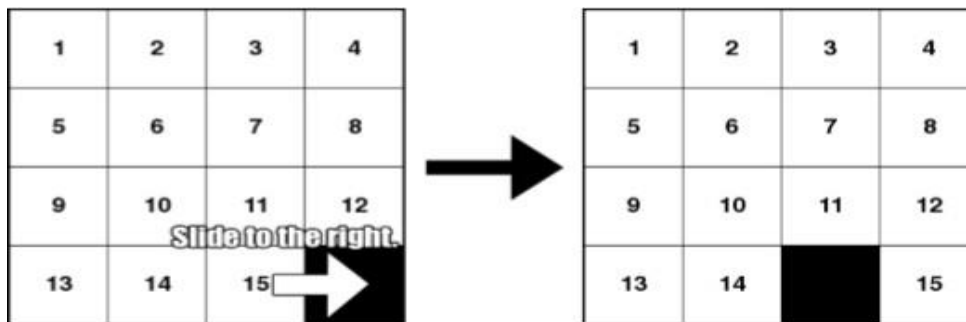
64 allMoves = [] # list of moves made from the solved configuration

```

Решавање Клизеће слагалице може бити веома незгодно. Могли би смо да програмирамо компјутер да то уради, али би то захтевало од нас да схватимо алгоритам који може да реши Клизећу слагалицу. То би било врло тешко и укључивало би много размишљања и напора да се то стави у овај програм.

Срећом, постоји лакши начин. Можемо само да подесимо да рачунар запамти све случајне покрете које је начинио када је креирао структуру тренутног стања игре, а онда се табла може решити једноставним извођењем супротних покрета. С обзиром да је табла првобитно била у решеном стању, поништавање свих покрета би је вратило у решено стање.

На пример, у наставку смо извршили покрет удесно на табли на левој страни странице, који оставља таблу у стање које је приказано на десној страни странице.



Након покрета удесно, ако начинимо супротан покрет (покрет улево) табла ће се вратити у првобитно стање. Дакле, да се вратимо у првобитно стање након неколико покрета, само треба да чинимо супротне покрете обрнутим редоследом. Ако смо учинили покрет удесно, а затим још један покрет удесно, а затим покрет на доле, морали би смо да начинимо покрет на горе, улево и још један улево како би смо поништили та прва три корака. Ово је много лакше него писање функције која може да реши ове слагалице само гледајући тренутно стање на њима.

Главна петља игре

```

66     while True: # main game loop
67         slideTo = None # the direction, if any, a tile should slide
68         msg = 'Click tile or press arrow keys to slide.' # contains the
69                 #message to show in the upper left corner.
70         if mainBoard == SOLVEDBOARD:
71             msg = 'Solved!'
72
73         drawBoard(mainBoard, msg)

```

У главној петљи игре, варијабла `slideTo` ће пратити у ком правцу играч жели да помери поље (започиње на почетку петље као `None` (ништа, празан) и поставља се касније), а `msg` варијабла прати шта треба да испише на врху прозора. Програм врши брзу проверу на линији 70 да види да ли структура тренутног стања табле има исту вредност као и решена структура табле која се налази у `SOLVEDBOARD`. Ако је тако, онда се `msg` варијабла мења у „Solved!“ тј. решено.

Ово се неће појавити на екрану док функција `drawBoard()` не буде позвана да то нацрта на `DISPLAYSURF` `Surface` објекту (што се врши на линији 73) и док `pygame.display.update()` не буде позвана да нацрта `display` `Surface` објекат на конкретном екрану компјутера (што се врши на линији 309 на крају петље игре).

Притисак на тастер

```

75     checkForQuit()
76     for event in pygame.event.get(): # event handling loop
77         if event.type == MOUSEBUTTONUP:
78             spotx, spoty = getSpotClicked(mainBoard, event.pos[0], \
79                                         event.pos[1])
80
81         if (spotx, spoty) == (None, None):
82             # check if the user clicked on an option button
83             if RESET_RECT.collidepoint(event.pos):
84                 resetAnimation(mainBoard, allMoves) # clicked on

```

```

85                                     #Reset button
86         allMoves = []
87     elif NEW_RECT.collidepoint(event.pos):
88         mainBoard, solutionSeq = generateNewPuzzle(80)
89         # clicked on New Game button
90         allMoves = []
91     elif SOLVE_RECT.collidepoint(event.pos):
92         resetAnimation(mainBoard, solutionSeq + allMoves)
93         # clicked on Solve button
94         allMoves = []

```

Пре одласка у петљу догађаја, програм позива функцију `checkForQuit()` на линији 75 да види да ли су креирани неки QUIT догађаји (и прекида програм ако их је било). Зашто имамо посебну функцију за управљање QUIT догађајима ће бити објашњено касније. For петља на линији 76 извршава код за управљање догађајем за било који други догађај који је креиран од када је последњи `pygame.event.get()` позван (или од када је програм започео, ако `pygame.event.get()` никада пре није био позван).

Ако је врста догађаја био `MOUSEBUTTONUP` догађај (то јест, ако је отпуштен тастер миша негде преко прозора), онда пуштамо координате миша `getSpotClicked()` функцији, која ће вратити координате тог места на табли где се отпуштање догодило. `Event.pos[0]` је X координата, а `event.pos[1]` је Y координата.

Ако се отпуштање тастер миша није догодило на једном од простора на табли (али се очигледно ипак догодило негде на прозору, јер је `MOUSEBUTTONUP` догађај креиран), онда ће функција `getSpotClicked()` вратити `None`. Ако је то случај, желимо да извршимо додатну проверу да видимо да ли је играч можда кликнуо на тастер `Reset` (поново), `New` (нова игра) или `Solve` (реши) који се не налазе на табле.

Кординате тастер миша се чувају у `pygame.Rect` објекат где се чувају варијабле `RESET_RECT`, `NEW_RECT` и `SOLVE_RECT`. Можемо проверити координате миша методом `collidepoint()` која враћа `True` ако се координате миша налазе унутар `Rect` објекта иначе враћа `False`.

Померање поља помоћу миша

```

95         else:
96             # check if the clicked tile was next to the blank spot
97
98             blankx, blanky = getBlankPosition(mainBoard)
99             if spotx == blankx + 1 and spoty == blanky:
100                 slideTo = LEFT
101             elif spotx == blankx - 1 and spoty == blanky:
102                 slideTo = RIGHT
103             elif spotx == blankx and spoty == blanky + 1:
104                 slideTo = UP
105             elif spotx == blankx and spoty == blanky - 1:
106                 slideTo = DOWN

```

Ако функција `getSpotClicked()` не врати `(None, None)` онда враћа уређени низ два цела броја који представљају X и Y координате тачке на табли где кликнуто. Онда `if` и `elif` функције у линијама 99 и 101 проверавају да ли је тачка на коју је кликнуто поље које се налази поред празног поља (у супротном поље неће имати где да се помери).

Наша `getBlankPosition()` функција ће преузети структуру података табле и вратити X и Y координате празног поља на табли, које чувамо у варијабли `blankx` и `blanky`. Ако је место на које је корисник кликнуо било поред празног поља, постављамо вредност варијабле `slideTo` на ону где поље треба да се помери.

Померање поља помоћу тастатуре

```
108         elif event.type == KEYUP:
109             # check if the user pressed a key to slide a tile
110             if event.key in (K_LEFT, K_a) and isValidMove(mainBoard, LEFT):
111                 slideTo = LEFT
112             elif event.key in (K_RIGHT, K_d) and isValidMove(mainBoard, RIGHT):
113                 slideTo = RIGHT
114             elif event.key in (K_UP, K_w) and isValidMove(mainBoard, UP):
115                 slideTo = UP
116             elif event.key in (K_DOWN, K_s) and isValidMove(mainBoard, DOWN):
117                 slideTo = DOWN
```

Такође можемо дозволити да корисник помера поља тако што ће притиснути дугме на тастатури. If и elif функције омогућавају кориснику подешавање slideTo варијабле било притиском на тастере са стрелицама или WASD тастере (биће објашњено касније). Свака if и elif функција има и позив за isValidMove() како би била сигурна да поље може да се помери у том правцу. (Нисмо морали да имамо овај позив када су у питању кликови мишем, јер провере суседног празног поља раде исту ствар).

„Једнако једном од“ – Трик са оператором

Израз event.key in (K_LEFT, K_a) је само Пајтон трик да би код био једноставнији. То је начин да се каже “ процени као True ако је event.key једнак једном од K_LEFT or K_a”. Следећа два израза врше процену на описан начин:

```
event.key in (K_LEFT, K_a)
```

```
event.key == K_LEFT or event.key == K_a
```

Можеш заиста да сачуваш одређени простор користећи овај трик када мораш да провераваш да ли је вредност једнака једној од више вредности. Један такав пример је дат на слици испод:

```
spam == 'dog' or spam == 'cat' or spam == 'mouse' or spam == 'horse' or spam == 42 or spam == 'dingo'
```

```
spam in ('dog', 'cat', 'mouse', 'horse', 42, 'dingo')
```

WASD и тастери са стрелицама

W, A, S и D тастери се обично користе у компјутерским играма да ураде исту ствар као и тастери са стрелицама, осим што играч може да користи своју леву руку уместо десне (пошто се тастери WASD налазе на левој страни тастатуре). W је за горе, A за лево, S је за доле, а D је за десно:



Конкретно ивођење померање поља

```
119     if slideTo:
120         slideAnimation(mainBoard, slideTo, \
121             'Click tile or press arrow keys to slide.', 8)
122                                     # show slide on screen
123         makeMove(mainBoard, slideTo)
124         allMoves.append(slideTo) # record the slide
125     pygame.display.update()
126     FPSLOCK.tick(FPS)
```

Сада када су сви догађаји регулисани, требало би ажурирамо одговарајуће варијабле које ће нам приказати ново стање игре на екрану. Ако је `slideTo` задато (било путем кода за управљање догађаја мишем или путем тастатуре) онда можемо позвати `slideAnimation()` да би смо извршили анимацију померања. Параметри су структура података табле, правац померања, порука за приказ током померања поља и брзина померања.

Након што се померање изврши треба да ажурирамо актуелну структуру података табле (што се врши помоћу `makeMove()` функције), а затим да додамо покрет `allMoves` листи свих досадашњих покрета. То се ради због тога што ако играч кликне на дугме `Reset` (поново) онда знамо како да понишtimo све покрете овог играча.

Стање мировања и прекид Pygame програма

```
129 def terminate():
130     pygame.quit()
131     sys.exit()
```

Ово је функција коју можемо да позовемо да позове и `pygame.quit()` и `sys.exit()` функције. Ово је мало синтаксичког шећера, тако да уместо да памтиш да учиниш оба ова позива, постоји само једна функција коју можемо позвати уместо тога.

Провера за одређени догађај и постављање догађаја у Pygame ред догађаја

```
134 def checkForQuit():
135     for event in pygame.event.get(QUIT): # get all the QUIT events
136         terminate() # terminate if any QUIT events are present
137     for event in pygame.event.get(KEYUP): # get all the KEYUP events
138         if event.key == K_ESCAPE:
139             terminate() # terminate if the KEYUP event was for the Esc key
140         pygame.event.post(event) # put the other KEYUP event objects back
```

Функција `checkForQuit()` ће проверити да ли има `QUIT` догађаја (или да ли је корисник притиснуо тастер `Esc`), а затим позвати функцију `terminate()` за прекидање. Али ово је мало компликовано и захтева неко објашњење.

Pygame интерно има своју структуру података листе коју креира и додаје јој објекте догађаја како се они стварају. Ова структура података се зове ред догађаја. Када се `pygame.event.get()` функција позове без параметара, цела листа се враћа. Међутим можеш да пустиш константу као што је `QUIT` до `pygame.event.get()`, тако да ће она само вратити `QUIT` догађаје (ако их има) који се налазе у унутрашњем реду догађаја. Остали догађаји ће остати у реду догађаја када се следећи пут позове `pygame.event.get()`.

Требало би напоменути да Pygame ред догађаја чува само до 127 објекта догађаја. Ако твој програм не позива довољно често pygame.event.get(), и ред се напуни, онда нови догађаји који се дешавају неће бити додати у ред догађаја.

Линија 135 извлачи листу QUIT догађаја из Pygame реда догађаја и враћа их. Ако у реду нема QUIT догађаја, програм се прекида.

Линија 137 извлачи све KEYUP догађаје из реда догађаја и проверава да ли су неки од њих за дугме ESC. Ако неки од догађаја то јесте, онда се програм прекида. Међутим, могу постојати KEYUP догађаји за друге тастере осим за ESC. У том случају, морамо да вратимо KEYUP догађај назад у Pygame ред догађаја. То можемо урадити помоћу функције pygame.event.post(), која додаје објекат догађаја који јој је прослеђен на крај Pygame реда догађаја.

Креирање структуре података табле

```
143 def getStartingBoard():
144     # Return a board data structure with tiles in the solved state.
145     # For example, if BOARDWIDTH and BOARDHEIGHT are both 3, this function
146     # returns [[1, 4, 7], [2, 5, 8], [3, 6, BLANK]]
147     counter = 1
148     board = []
149     for x in range(BOARDWIDTH):
150         column = []
151         for y in range(BOARDHEIGHT):
152             column.append(counter)
153             counter += BOARDWIDTH
154         board.append(column)
155         counter -= BOARDWIDTH * (BOARDHEIGHT - 1) + BOARDWIDTH - 1
156
157     board[BOARDWIDTH-1][BOARDHEIGHT-1] = BLANK
158     return board
```

Подаци структуре getStartingBoard() ће креирати и вратити структуру података која представља решену таблу, где су сва поља нумерисана по реду, а празно поље се налази у доњем десном углу. То се ради са угњежђеним петљама, баш као што је направљена структура података табле у игри MemoryPuzzle.

Међутим, треба обратити пажњу да прва колона неће бити [1,2,3] него [1,4,7] ако је табле 3x3. То је због тога што се бројеви на пољима повећавају за један идући дуж реда, не низ колону. Идући кроз колону, бројеви се повећавају у зависности од висине табле (што је сачувано у BOARDWIDTH константи). Ми ћемо користити бројач варијабле за праћење броја који би требало да буде на следећем пољу. Када се заврши нумерација поља у колони, онда би требало да поставимо бројач на број на почетку следеће колоне.

Не праћење празне позиције

```
161 def getBlankPosition(board):
162     # Return the x and y of board coordinates of the blank space.
163     for x in range(BOARDWIDTH):
164         for y in range(BOARDHEIGHT):
165             if board[x][y] == BLANK:
166                 return (x, y)
```

Када год наш код треба да пронађе XY координате празног простора, уместо праћења где се налази празно поље после сваког покрета, можемо само да креирамо функцију која пролази кроз читаву таблу и проналази координате празног простора. Вредност None се користи у структури података табле да представља празан простор. Код у функцији getBlankPosition() једноставно користи угњежђене петље да пронађе који простор на табли је празан простор.

Извођење померања кроз ажурирање структуре података табле

```
169 def makeMove(board, move):
170     # This function does not check if the move is valid.
171     blankx, blanky = getBlankPosition(board)
172
173     if move == UP:
174         board[blankx][blanky], board[blankx][blanky+1] = board[blankx][blanky+1], \
175                                                         board[blankx][blanky]
176     elif move == DOWN:
177         board[blankx][blanky], board[blankx][blanky-1] = board[blankx][blanky-1], \
178                                                         board[blankx][blanky]
179     elif move == LEFT:
180         board[blankx][blanky], board[blankx+1][blanky] = board[blankx+1][blanky], \
181                                                         board[blankx][blanky]
182     elif move == RIGHT:
183         board[blankx][blanky], board[blankx-1][blanky] = board[blankx-1][blanky], \
184                                                         board[blankx][blanky]
```

Структура података у параметру табле је 2D листа где се налазе сва поља. Кад год играч направи покрет, програм мора да ажурира ову структуру података. Оно што се дешава је да се вредност за поља замени вредношћу за празан простор.

Функција makeMove() не мора да врати никакве вредности, јер параметар табле има листу референци прослеђену као аргумент. То значи да ће било која промена коју уносимо на табли у овој функцији бити учињена на листи вредности која је прослеђена до makeMove(). Можеш да погледаш концепт референци на <http://invy.com/references>.

Када НЕ користити тврдње

```
187 def isValidMove(board, move):
188     blankx, blanky = getBlankPosition(board)
189     return (move == UP and blanky != len(board[0]) - 1) or \
190           (move == DOWN and blanky != 0) or \
191           (move == LEFT and blankx != len(board) - 1) or \
192           (move == RIGHT and blankx != 0)
```

Функцији isValidMove() је прослеђена структура података табле и покрет који играч жели да направи. Повратна вредност је True ако је овај покрет могућ и False ако није. На пример, не можеш да помериш поље улево стотину пута за редом, јер ће на крају ивице бити празан простор и неће бити више поља за померање улево.

Да ли је покрет важећи или не зависи од тога где је празан простор. Ова функција упућује позив функцији getBlankPosition() да пронађе X и Y координате празног места. Линије 189 до 192 су повратне вредности тврдње са једним изразом. Косе црте на крају прве три линије говоре Пајтон тумачу да то није крај линије кода (иако се налази на крају линије). То ће нам омогућити да раздвојимо једну линију кода на више линија тако да код изгледа лепо.

Делови израза у загради су придружени од стране оператора, зато што један од њих мора да буде True да би цео израз био True. Свака од ових делова проверава шта је намеравани покрет, а онда гледа да ли координате празног простора омогућавају тај покрет.

Добијање не-тако-случајног покрета

```
195 def getRandomMove(board, lastMove=None):
196     # start with a full list of all four moves
197     validMoves = [UP, DOWN, LEFT, RIGHT]
198
199     # remove moves from the list as they are disqualified
200     if lastMove == UP or not isValidMove(board, DOWN):
201         validMoves.remove(DOWN)
202     if lastMove == DOWN or not isValidMove(board, UP):
203         validMoves.remove(UP)
204     if lastMove == LEFT or not isValidMove(board, RIGHT):
205         validMoves.remove(RIGHT)
206     if lastMove == RIGHT or not isValidMove(board, LEFT):
207         validMoves.remove(LEFT)
208
209     # return a random move from the list of remaining moves
210     return random.choice(validMoves)
```

На почетку игре, почињемо са структуром података табле у решеном, уређеном стању и креирамо слагалице насумице померајући поља унаоколо. Да би смо одлучили у ком од четири правца би требало да се померимо, позивамо нашу функцију getRandomMove(). Иначе, могли би смо користити само функцију random.choice() и проследити јој четворку (UP, DOWN, LEFT, RIGHT) како би Пајтон једноставно насумично изабрао вредност правца за нас. Али, игра Клизећа слагалица има мала ограничења за нас, спречава нас да бирамо чисто случајан број.

Ако смо у Клизећој слагалици померили поље улево, а затим померили удесно, завршили би смо на потпуно истој табли коју смо имали на почетку. Бесмислено је направити покрет, пратећи га супротним покретом. Исто тако, ако је празан простор у доњем десном углу онда је немогуће померити поље на горе или улево.

Код у функцији getRandomMove() ће узети ове факторе у обзир. Да би се спречило да функција одабере последњи покрет који је направљен, позивана функција може да проследи вредности правца за lastMove параметар. Линија 197 почиње листом све четири вредности правца које су сачуване у варијабли validMoves. Вредност lastMove (ако није постављена на None) је уклоњена из validMoves. У зависности од тога да ли је празан простор на крају табле, линије 200 до 207 ће уклонити друге вредности правца са lastMove листе.

Од вредности које су преостале у lastMove, бира се једна насумично позивом random.choice() и враћа се.

Претварање координата поља у координате пиксела

```
213 def getLeftTopOfTile(tileX, tileY):
214     left = XMARGIN + (tileX * TILESIZE) + (tileX - 1)
215     top = YMARGIN + (tileY * TILESIZE) + (tileY - 1)
216     return (left, top)
```

Функција getLeftTopOfTile() претвара координате табле у координате пиксела. За XY координате табле које су јој прослеђене, функција израчунава и враћа XY координате пиксела у горњем левом углу простора табле.

Претварање из пиксел координата у координате табле

```
219 def getSpotClicked(board, x, y):
220     # from the x & y pixel coordinates, get the x & y board coordinates
221     for tileX in range(len(board)):
222         for tileY in range(len(board[0])):
223             left, top = getLeftTopOfTile(tileX, tileY)
224             tileRect = pygame.Rect(left, top, TILESIZE, TILESIZE)
225             if tileRect.collidepoint(x, y):
226                 return (tileX, tileY)
227     return (None, None)
```

Функција `getSpotClicked()` ради супротно од `getLeftTopOfTile()` и претвара из пиксел координата у координате табле. Угњежђене петље на линијама 221 и 222 пролазе кроз све могуће XY координате табле, а ако се координате пиксела које су јој прослеђене налазе у простору на табли, она враћа те координате табле. Пошто сва поља имају ширину и висину које су постављене у `TILESIZE` константи, можемо да креирамо `Rect` објекат који представља простор на табли за добијање координата пиксела на горњем левој углу простора табле, а затим да користимо `collidepoint()` `Rect` начин да видимо да ли се координате пиксела налазе унутар подручја тог `Rect` објекта.

Ако координате пиксела које су јој прослеђене нису у било ком простору табле, онда се вредност `(None, None)` враћа.

Цртање поља

```
230 def drawTile(tilex, tiley, number, adjx=0, adjy=0):
231     # draw a tile at board coordinates tilex and tiley, optionally a few
232     # pixels over (determined by adjx and adjy)
233     left, top = getLeftTopOfTile(tilex, tiley)
234     pygame.draw.rect(DISPLAYSURF, TILECOLOR, (left + adjx, top + adjy, \
235                                                  TILESIZE, TILESIZE))
236     textSurf = BASICFONT.render(str(number), True, TEXTCOLOR)
237     textRect = textSurf.get_rect()
238     textRect.center = (left + int(TILESIZE / 2) + adjx, top + int(TILESIZE / 2) + adjy)
239     DISPLAYSURF.blit(textSurf, textRect)
240
```

`DrawTile()` функција ће нацртати једно нумерисано поље на табли. Параметри `tilex` и `tiley` су координате поља на табли. Бројчани параметар је низ броја поља (као „3“ или „12“). Кључне речи `adjx` и `adjy` параметра служе за вршење мањих подешавања позиција поља. На пример, прослеђивање 5 за `adjx` би учинило да се поље појави 5 пиксела удесно од `tilex` и `tiley` простора на табли. Прослеђивање -10 за `adjx` би учинило да се поље појави 10 пиксела улево од простора.

Ове вредности подешавања ће бити од користи када треба да нацртамо поље усред померања. Ако вредности нису прослеђене за ове аргументе када се позове `drawTile()`, онда се оне подразумевано постављају на нула. То значи да ће поље бити тачно на простору табле који су дали `tilex` и `tiley`.

Pygame функције цртања користе само координате пиксела, тако да прва линија 233 претвара координате табле из `tilex` и `tiley` у пиксел координате, које ћемо чувати у варијаблама лево и горе (пошто `getLeftTopOfTile()` враћа координате горњег левог угла). Цртамо квадрат поља у позадини са позивом `pygame.draw.rect()`, а додајемо `adjx` и `adjy` вредности лево и горе у случају да код мора да прилагоди положај поља.

Линије 236 и 239 онда креирају Surface објекат који има текстуални број на себи. Функција drawTile() не зове pygame.display.update() функцију, пошто ће позивалац drawTile() вероватно желети да нацрта још поља за остатак табле пре него што се она појави на екрану.

Појављивање текста на екрану

```
242 def makeText(text, color, bgcolor, top, left):
243     # create the Surface and Rect objects for some text.
244     textSurf = BASICFONT.render(text, True, color, bgcolor)
245     textRect = textSurf.get_rect()
246     textRect.topleft = (top, left)
247     return (textSurf, textRect)
```

Функција makeText() управља креирањем Surface и Rect објеката за позиционирање текста на екрану. Уместо да начинимо све ове позиве сваки пут када желимо да направимо текст на екрану, можемо само да позовемо makeText(). То нам смањује неопходну количину куцања за наш програм.

Цртање табле

```
250 def drawBoard(board, message):
251     DISPLAYSURF.fill(BGCOLOR)
252     if message:
253         textSurf, textRect = makeText(message, MESSAGECOLOR, BGCOLOR, 5, 5)
254         DISPLAYSURF.blit(textSurf, textRect)
255
256     for tilex in range(len(board)):
257         for tiley in range(len(board[0])):
258             if board[tilex][tiley]:
259                 drawTile(tilex, tiley, board[tilex][tiley])
```

Ова функција управља цртањем целокупне табле и свих њених поља на дисплеју DISPLAYSURF Surface објекта. Метод fill() у линији 251 у потпуности црта преко било чега што је било нацртано на дисплеју Surface објекта раније, тако да започињемо од нуле.

Линије 252 до 254 управљају писањем поруке на врху прозора. Ово користимо за „Generating new puzzle...” то јест генерисање нове слагалице и било ког другог текста који желимо да прикажемо на врху прозора. Имајте на уму да услови if тврдњи подразумевају да је празан низ False (нетачна) вредност, тако да ако је порука постављена на ‘ ‘ (једноструки наводници и између празан простор) онда услов False и линије 253 и 254 се прескачу.

Даље, угњежђене петље се користе за цртање сваког поља на дисплеју Surface објекта позивањем функције drawTile().

Цртање граница табле

```
261 left, top = getLeftTopOfTile(0, 0)
262 width = BOARDWIDTH * TILESIZE
263 height = BOARDHEIGHT * TILESIZE
264 pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, \
265                 (left - 5, top - 5, width + 11, height + 11), 4)
```

Линије 261 до 265 цртају границу око поља. Горњи леви угао границе ће бити 5 пиксела лево и 5 пиксела изнад горњег левог угла на координатама табле (0,0). Ширина и висина границе се рачунају у односу на број поља одговарајуће висине и ширине (које су сачуване у BOARDWIDTH и BOARDHEIGHT константи) помножен са величином поља (која су сачувана у TILESIZE константи).

Правоугаоник који цртамо у линији 265 ће имати дебљину од 4 пиксела, па ћемо померити границу 5 пиксела улево и изнад места где горња и лева варијабла показују, тако да дебљина линије не преклапа поља. Такође ћемо додати 11 ширини и дужини (5 од ових 11 пиксела служе за надокнаду померања правоугаоника улево и на горе).

Цртање тастера

```
267 DISPLAYSURF.blit(RESET_SURF, RESET_RECT)
268 DISPLAYSURF.blit(NEW_SURF, NEW_RECT)
269 DISPLAYSURF.blit(SOLVE_SURF, SOLVE_RECT)
```

Коначно, повлачимо тастере покрета на екрану. Текст и положај ових тастера се никада не мења, због чега су они сачувани у константним варијаблама на почетку main() функције.

Анимација померања поља

```
272 def slideAnimation(board, direction, message, animationSpeed):
273     # Note: This function does not check if the move is valid.
274
275     blankx, blanky = getBlankPosition(board)
276     if direction == UP:
277         movex = blankx
278         movey = blanky + 1
279     elif direction == DOWN:
280         movex = blankx
281         movey = blanky - 1
282     elif direction == LEFT:
283         movex = blankx + 1
284         movey = blanky
285     elif direction == RIGHT:
286         movex = blankx - 1
287         movey = blanky
```

Прва ствар коју наш код анимације померања поља мора да израчуна је где је празан простор и где је поље које се помера. Коментар на линији 273 нас подсећа да би код који позива slideAnimation() требало да обезбеди да је покрет који прослеђује за параметар правца валидан покрет.

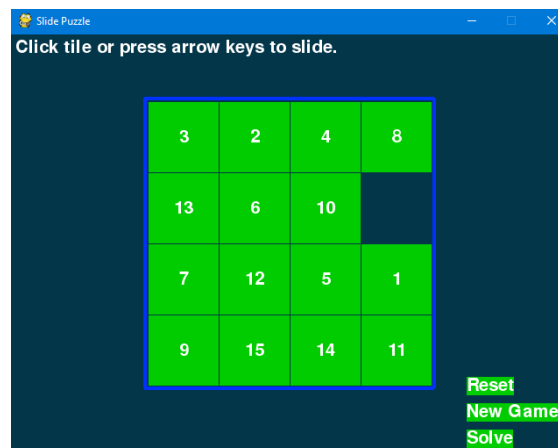
Координате празног простора добијамо позивом функције getBlankPosition(). Из тих координата и правца померања, можемо закључити XY координате поља табле које ће се померити. Ове координате ће бити сачуване у movex и movey варијаблама.

Метода copy() Surface

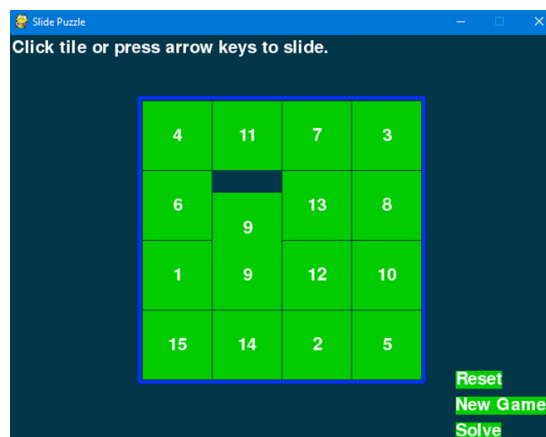
```
289 # prepare the base surface
290 drawBoard(board, message)
291 baseSurf = DISPLAYSURF.copy()
292 # draw a blank space over the moving tile on the baseSurf Surface.
293 moveLeft, moveTop = getLeftTopOfTile(movex, movey)
294 pygame.draw.rect(baseSurf, BG_COLOR, (moveLeft, moveTop, TILE_SIZE, TILE_SIZE))
```

Сору() метода Surface објекта враћа нови Surface објект који има исту слику нацртану на себи. Али, то су два одвојена Surface објекта. Након позива сору() методе, ако цртамо на једном Surface објекту помоћу blit() или Pygame функција за цртање, то неће променити слику на другом Surface објекту. Ову копију чувамо у BaseSurf варијабли на линији 291.

Даље, бојимо још један празан простор изнад поља које ће се померити. То је зато што када цртамо сваки кадар анимације, цртамо поље које се помера преко различитих делова BaseSurf Surface објекта. Ако не уклонимо поље које се помера на BaseSurf Surface, онда ће оно и даље бити ту док цртамо поље које се помера. У том случају, BaseSurf Surface би требало да изгледа овако:



А затим би требало да изгледа овако ако нацртамо поље 9 како се помера на горе:



```

296     for i in range(0, TILESIZ, animationSpeed):
297         # animate the tile sliding over
298         checkForQuit()
299         DISPLAYSURF.blit(baseSurf, (0, 0))
300         if direction == UP:
301             drawTile(movex, movey, board[movex][movey], 0, -i)
302         if direction == DOWN:
303             drawTile(movex, movey, board[movex][movey], 0, i)
304         if direction == LEFT:
305             drawTile(movex, movey, board[movex][movey], -i, 0)
306         if direction == RIGHT:
307             drawTile(movex, movey, board[movex][movey], i, 0)
308
309     pygame.display.update()
310     FPSCLOCK.tick(FPS)

```

Да би смо нацртали кадрове анимације померања, морамо да нацртамо BaseSurf Surface на дисплеју Surface, а затим на сваком кадру анимације да нацртамо поље које се помера све ближе коначној позицији, где је првобитно био празан простор. Простор између два суседна поља је исте величине као једно поље, коју смо сачували у TILESIZE. Код користи петљу да иде од 0 до TILESIZE.

То би иначе значило да ћемо нацртати поље 0 пиксела изнад, а затим на следећем кадру поље 1 пиксел изнад, затим 2 пиксела, па 3, и тако даље. Сваки од ових кадрова би заузео 1/30 секунде. Ако је TILESIZE подешен на 80 (што је то у програму у овој књизи у линији 12) онда би померање поља трајало преко две и по секунде, што је заправо споро.

Дакле, уместо тога, имамо for петљу изведену од 0 до TILESIZE по неколико пиксела у сваком кадру. Број пиксела који прескаче се чува у animationSpeed, што се прослеђује када се позове slideAnimation(). На пример, ако је animationSpeed подешен на 8 и константа TILESIZE је постављена на 80, онда ће for петља и range (0,TILESIZE, animationSpeed) поставити варијаблу на вредности 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, (не укључујући 80 јер range() функција иде на горе, али не укључује и другу границу). То значи да ће целокупна анимација померања бити учињена у 10 кадрова, што би значило да ће се извршити у 10/30 секунди (трећи део секунде), пошто игра ради на 30фпс кадрова у секунди.

Линије 300 до 307 обезбеђују да нацртамо поље које се помера у исправном правцу (на основу онога што је вредност варијабле правца). Након што се заврши анимација, враћа се функција. Обрати пажњу да док се анимација дешава, било којим догађајима креираним од стране корисника се не управља. Овим догађајима ће се управљати када следећи пут извршење достигне линију 70 у main() функцији или код у checkForQuit() функцији.

Креирање нове слагалице

```
313 def generateNewPuzzle(numSlides):
314     # From a starting configuration, make numSlides number of moves (and
315     # animate these moves).
316     sequence = []
317     board = getStartingBoard()
318     drawBoard(board, '')
319     pygame.display.update()
320     pygame.time.wait(500) # pause 500 milliseconds for effect
```

GenerateNewPuzzle() функција ће бити позвана на почетку сваке нове игре. То ће створити нову структуру података табле позивањем getStartingBoard(), а затим насумичним мешањем. Првих неколико линија generateNewPuzzle() узимају таблу, а затим је цртају на екрану (замрзавањем на пола секунде како би играч видео нову таблу за тренутак).

```
321     lastMove = None
322     for i in range(numSlides):
323         move = getRandomMove(board, lastMove)
324         slideAnimation(board, move, 'Generating new puzzle...', \
325                        animationSpeed=int(TILESIZE / 3))
326         makeMove(board, move)
327         sequence.append(move)
328         lastMove = move
329     return (board, sequence)
```

Параметар numSlides ће показати функцији колико од тих случајних покрета да се направи. Код за обављање случајног покрета је getRandomMove() позив у линији 323 да би се добио сам покрет, а онда се позива slideAnimation() да изврши анимацију на екрану.

Пошто вршење анимације померања заправо не ажурира структура података табле, ми ажурирамо таблу позивом `makeMove()` у линији 326.

Морамо да пратимо сваки од случајних покрета који је направљен тако да играч може касније да кликне на дугме `Solve` и да програм поништи све ове случајне покрете. (Одељак „Бити паметан помоћу глупог кода“ говори о томе зашто и како ово радимо). Дакле, тај покрет је додат на листу покрета у низу у линији 327.

Затим чувамо случајан покрет у варијабли званој `lastMove` која ће бити прослеђена до `getRandomMove()` при следећем извођењу. Ово спречава да следећи случајан покрет поништи случајан покрет који смо управо направили.

Све то треба да се деси `numSlides` број пута, па смо ставили линије 323 до 328 у `for` петљу. Када се заврши мешање на табли, онда враћамо структуру података табле, као и листу случајних покрета направљених на њој.

Анимација поновног подешавања табле

```
332 def resetAnimation(board, allMoves):
333     # make all of the moves in allMoves in reverse.
334     revAllMoves = allMoves[:] # gets a copy of the list
335     revAllMoves.reverse()
336
337     for move in revAllMoves:
338         if move == UP:
339             oppositeMove = DOWN
340         elif move == DOWN:
341             oppositeMove = UP
342         elif move == RIGHT:
343             oppositeMove = LEFT
344         elif move == LEFT:
345             oppositeMove = RIGHT
346         slideAnimation(board, oppositeMove, '', animationSpeed=int(TILESIZE/2))
347         makeMove(board, oppositeMove)
```

Када играч кликне на `Reset` или `Solve`, програм игре Клизећа слагалица треба да поништи све покрете који су направљени на табли. Списак вредности праваца за покрете ће бити прослеђени као аргумент за параметар `allMoves`.

Линија 334 користи одсецање листе за креирање дупликата листе `allMoves`. Имај на уму да ако не наведеш број пре `:` онда Пајтон претпоставља да тај део треба да се настави до самог краја листе. Тако `allMoves[:]` креира део читаве `allMoves` листе. То ствара копију актуелне листе која ће се чувати у `revAllMoves`, а не само копија листе референце. (Види <http://invpy.com/references> за детаље).

Да би се поништили сви покрети у `allMoves`, морамо да обавимо супротне покрете од покрета у `allMoves`, и обрнутим редоследом. Постоји метода листе која се зове `reverse()` која ће преокренути редослед ставки у листи. То позивамо на листи `revAllMoves` и линији 335.

`for` петља на линији 337 се изводи преко листе вредности праваца. Запамти, желимо супротан покрет, тако да `if` и `elif` тврдње линије 338 до 345 подешавају исправну вредност праваца у варијабли `oppositeMove`. Затим зовемо `slideAnimation()` за вршење анимације, и `makeMove()` за ажурирање структуре података табле.

```
350 if __name__ == '__main__':
351     main()
```

Баш као у игри Меморијска слагалица, након што су све `def` тврдње позване да креирају све функције, позивамо `main()` функцију да започне главни део програма.

То је све што се тиче програма Клизећа слагалица. Али, хајде да разговарамо о неким општим програмским концептима који су се појавили у овој игри.

Компромиси – Време наспрот Меморије

Наравно, постоји неколико различитих начина да се напише игра Клизећа слагалица тако да изгледа и функционише на потпуно исти начин, иако је код другачији. Постоји много различитих начина на који програм који ради задатак може да се напише. Најчешће разлике су прављење компромиса између времена извршења и коришћења меморије.

Обично, што брже програм може да ради, то је боље. Ово се посебно односи на програме који би требало да изврше много рачунања, без обзира на то да ли су научни временски симулатори или игре са великом количином детаљне 3D графике коју би требало нацртати. Такође је добро користити најмању могућу количину меморије. Што има више варијабли и што су веће листе које наш програм користи, то он више меморије заузима. (Можеш да сазнаш како да измериш колико меморије заузима твој програм, као и време извршења на <http://invpy.com/profiling> .)

Сада, програми у овој књизи нису довољно велики и компликовани где мораш да бринеш о очувању меморије или оптимизацији времена извршења. Али, то може бити нешто о чему можеш да размишљаш када постанеш вештији програмер.

На пример, размотримо функцију `getBlankPosition()`. Овој функцији је потребно време да ради, јер пролази кроз све могуће координате табле да пронађе где је празан простор. Уместо тога, можемо само да имамо `blankspacex` и `blankspacey` варијабле које би имале ове XY координате, тако да не морамо да претражујемо читаву таблу сваки пут када желимо да знамо где је. (Такође нам је потребан и код који ажурира `blankspacex` и `blankspacey` варијабле кад год се направи покрет. Овај код може да иде у `makeMove()`.) Коришћење ових варијабли може да заузме више меморије, али би уштедело на времену извршења, па би твој програм радио брже.

Други пример је да чувамо структуру података табле у решеном стању у варијабли `SOLVEBOARD`, тако да можемо да упоредимо тренутну таблу са `SOLVEBOARD` да видимо да ли је играч решио слагалицу. Сваки пут када би смо хтели да урадимо ову проверу, можемо само да позовемо функцију `GetStartingBoard()` и упоредимо враћену вредност са тренутном таблом. Онда нам `SOLVEBOARD` варијабла не би била потребна. То би нам сачувало мало меморије, али онда би нашем програму требало више времена да ради, јер онда поново креира структуру података табле у решеном стању сваки пут када вршимо ову проверу.

Постоји једна ствар коју ипак треба запамтити. Писање кода који је читљив је веома важна вештина. Код који је читљив је код који је лако разумљив, нарочито од стране програмера који нису писали код. Ако неки други програмер може да погледа изворни код твог програма и схвати шта он ради без много проблема, онда је тај програм веома читљив. Читљивост је важна, јер када желиш да поправиш кварове или додаш нове функције програму (и кварови и идеје за нове функције се увек појављују), код читљивог програма се ови задаци много лакше обављају.

Никог није брига за само неколико бајтова

Такође, постоји једна ствар која може изгледати мало смешном да се каже у овој књизи, јер изгледа очигледно, али многи људи се питају у вези са тим. Требало би да знаш да коришћење кратких назива варијабли као `x` или `num` уместо дужих, назива варијабли који више описују, као што су `blankx` или `numSlides` не штеди нимало меморије када твој програм заправо ради. Коришћење ових дужих назива варијабли је боље, јер ће учинити твој програм читљивијим.

Можда се такође сетиш неких паметних трикова који штеде неколико бајтова меморије ту и тамо. Један трик је да када ти варијабла више није потребна, можеш поново да искористиш називе те варијабле у различите сврхе, уместо коришћења две различито именоване варијабле.

Покушај да одолош искушењу да то учиниш. Обично ови трикови смањују читљивост кода и отежавају отклањање кварова твог програма. Модерни рачунари имају милијарде бајтова меморије, а уштедети неколико бајтова ту и тамо заиста није вредно тога да се код направи збуњујућим за људе програмере.

Никог није брига за неколико милиона наносекунди

Слично томе, постоје ситуације када можеш да преуредиш свој код на неки начин, тако да га направиш нешто бржим за неколико наносекунди. Ови трикови такође обично чине код теже читљивим. Када узмеш у обзир да је неколико милијарди наносекунди прошло за време које ти је потребно да прочиташ ову реченицу, уштедети неколико наносекунди времена извршења у свом програму неће бити примећено од стране играча.

Преглед

Ово поглавље није увело никакве нове Ругаме програмске концепте које игра Меморијска слагалица није користила, осим коришћења `copy()` методе `Surface` објекта. Само познавање неколико различитих концепата ће ти омогућити да креираш потпуно различите игре.

За вежбање, можеш да преузмеш луде верзије програма Клизећа слагалица са <http://invpy.com/buggy/slidepuzzle>.