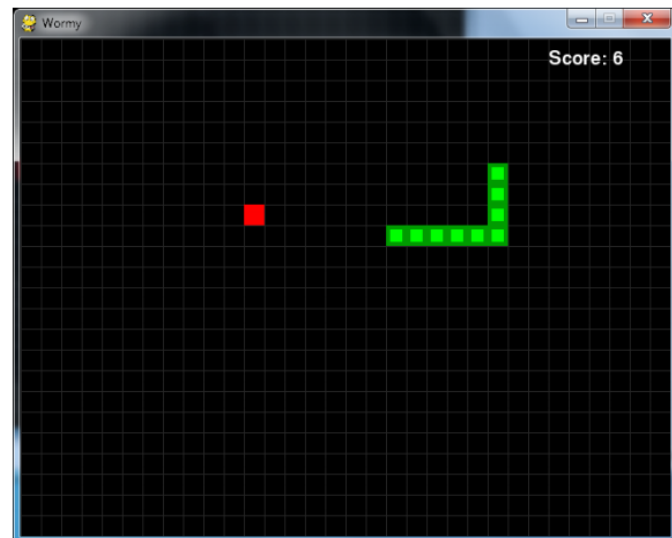


Wormy



Како се игра Змија

Змија је игра слична игри Нибблес. Играч започиње игру тако што контролише кратку змију која се константно креће по екрану. Играч не може да успори или заустави змију, већ само може да је усмерава у смеровима горе, доле, лево или десно. Црвена тачкица се појављује на насумичним местима на екрану и играч треба да усмери змију тако да поједе ту црвену тачкицу тако што ће прећи преко ње. Сваки пут када поједе јабуку, змија постане дужа и нова јабука се појави на новом насумичном месту на екрану. Игра се завршава када змија уједе саму себе или када удари о ивицу прозора по ком се игра.

Изворни код

У наставку је дат изворни код за игру Змија. Овај изворни код можете ископирати са адресе: <http://invpy.com/wormy.py>. Уколико добијете неку поруку о грешкама, погледајте на којој линији је пријављена грешка и проверите свој код. Проверите да ли је лепо урађено копирање. На адреси: <http://invpy.com/diff/wormy>, можете прекопирати свој код и упоредити га са кодом из ове књиге у потрази за различитостима.

```
1. # Wormy (a Nibbles clone)
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 = 15
10. WINDOWWIDTH = 640
11. WINDOWHEIGHT = 480
12. CELLSIZE = 20
13. assert WINDOWWIDTH % CELLSIZE == 0, "Window width must be a
    multiple of cell size."
14. assert WINDOWHEIGHT % CELLSIZE == 0, "Window height must be a
    multiple of cell size."
15. CELLWIDTH = int(WINDOWWIDTH / CELLSIZE)
16. CELLHEIGHT = int(WINDOWHEIGHT / CELLSIZE)
17.
18. #           R      G      B
19. WHITE      = (255, 255, 255)
20. BLACK      = ( 0,   0,   0)
21. RED        = (255,  0,   0)
22. GREEN      = ( 0, 255,  0)
23. DARKGREEN  = ( 0, 155,  0)
24. DARKGRAY   = ( 40,  40,  40)
25. BGCOLOR = BLACK
26.
27. UP = 'up'
28. DOWN = 'down'
29. LEFT = 'left'
30. RIGHT = 'right'
31.
32. HEAD = 0 # syntactic sugar: index of the worm's head
33.
34. def main():
35.     global FPSCLOCK, DISPLAYSURF, BASICFONT
36.
37.     pygame.init()
38.     FPSCLOCK = pygame.time.Clock()
39.     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH,
        WINDOWHEIGHT))
40.     BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
41.     pygame.display.set_caption('Wormy')
42.
43.     showStartScreen()
44.     while True:
45.         runGame()
46.         showGameOverScreen()
47.
48.
49. def runGame():
50.     # Set a random start point.
51.     startx = random.randint(5, CELLWIDTH - 6)
52.     starty = random.randint(5, CELLHEIGHT - 6)
53.     wormCoords = [{'x': startx,     'y': starty},
54.                   {'x': startx - 1, 'y': starty},
55.                   {'x': startx - 2, 'y': starty}]
56.     direction = RIGHT
57.
58.     # Start the apple in a random place.
59.     apple = getRandomLocation()
60.
61.     while True: # main game loop
62.         for event in pygame.event.get(): # event handling loop
63.             if event.type == QUIT:
64.                 terminate()
65.             elif event.type == KEYDOWN:

```

```

66.         if (event.key == K_LEFT or event.key == K_a) and
            direction != RIGHT:
67.             direction = LEFT
68.         elif (event.key == K_RIGHT or event.key == K_d) and
            direction != LEFT:
69.             direction = RIGHT
70.         elif (event.key == K_UP or event.key == K_w) and
            direction != DOWN:
71.             direction = UP
72.         elif (event.key == K_DOWN or event.key == K_s) and
            direction != UP:
73.             direction = DOWN
74.         elif event.key == K_ESCAPE:
75.             terminate()
76.
77.         # check if the worm has hit itself or the edge
78.         if wormCoords[HEAD]['x'] == -1 or wormCoords[HEAD]['x'] ==
CELLWIDTH or wormCoords[HEAD]['y'] == -1 or wormCoords[HEAD]['y'] ==
CELLHEIGHT:
79.             return # game over
80.         for wormBody in wormCoords[1:]:
81.             if wormBody['x'] == wormCoords[HEAD]['x'] and
wormBody['y'] == wormCoords[HEAD]['y']:
82.                 return # game over
83.
84.         # check if worm has eaten an apply
85.         if wormCoords[HEAD]['x'] == apple['x'] and
wormCoords[HEAD]['y'] == apple['y']:
86.             # don't remove worm's tail segment
87.             apple = getRandomLocation() # set a new apple somewhere
88.         else:
89.             del wormCoords[-1] # remove worm's tail segment
90.
91.         # move the worm by adding a segment in the direction it is
moving
92.         if direction == UP:
93.             newHead = {'x': wormCoords[HEAD]['x'], 'y':
wormCoords[HEAD]['y'] - 1}
94.         elif direction == DOWN:
95.             newHead = {'x': wormCoords[HEAD]['x'], 'y':
wormCoords[HEAD]['y'] + 1}
96.         elif direction == LEFT:
97.             newHead = {'x': wormCoords[HEAD]['x'] - 1, 'y':
wormCoords[HEAD]['y']}
98.         elif direction == RIGHT:
99.             newHead = {'x': wormCoords[HEAD]['x'] + 1, 'y':
wormCoords[HEAD]['y']}
100.         wormCoords.insert(0, newHead)
101.         DISPLAYSURF.fill(BGCOLOR)
102.         drawGrid()
103.         drawWorm(wormCoords)
104.         drawApple(apple)
105.         drawScore(len(wormCoords) - 3)
106.         pygame.display.update()
107.         FPSCLOCK.tick(FPS)
108.
109.     def drawPressKeyMsg():
110.         pressKeySurf = BASICFONT.render('Press a key to play.', True,
DARKGRAY)
111.         pressKeyRect = pressKeySurf.get_rect()
112.         pressKeyRect.topleft = (WINDOWWIDTH - 200, WINDOWHEIGHT - 30)

```

```

113.     DISPLAYSURF.blit(pressKeySurf, pressKeyRect)
114.
115.
116. def checkForKeyPress():
117.     if len(pygame.event.get(QUIT)) > 0:
118.         terminate()
119.
120.     keyUpEvents = pygame.event.get(KEYUP)
121.     if len(keyUpEvents) == 0:
122.         return None
123.     if keyUpEvents[0].key == K_ESCAPE:
124.         terminate()
125.     return keyUpEvents[0].key
126.
127.
128. def showStartScreen():
129.     titleFont = pygame.font.Font('freesansbold.ttf', 100)
130.     titleSurf1 = titleFont.render('Wormy!', True, WHITE, DARKGREEN)
131.     titleSurf2 = titleFont.render('Wormy!', True, GREEN)
132.
133.     degrees1 = 0
134.     degrees2 = 0
135.     while True:
136.         DISPLAYSURF.fill(BGCOLOR)
137.         rotatedSurf1 = pygame.transform.rotate(titleSurf1,
degrees1)
138.         rotatedRect1 = rotatedSurf1.get_rect()
139.         rotatedRect1.center = (WINDOWWIDTH / 2, WINDOWHEIGHT / 2)
140.         DISPLAYSURF.blit(rotatedSurf1, rotatedRect1)
141.
142.         rotatedSurf2 = pygame.transform.rotate(titleSurf2,
degrees2)
143.         rotatedRect2 = rotatedSurf2.get_rect()
144.         rotatedRect2.center = (WINDOWWIDTH / 2, WINDOWHEIGHT / 2)
145.         DISPLAYSURF.blit(rotatedSurf2, rotatedRect2)
146.
147.         drawPressKeyMsg()
148.
149.         if checkForKeyPress():
150.             pygame.event.get() # clear event queue
151.             return
152.         pygame.display.update()
153.         FPSCLOCK.tick(FPS)
154.         degrees1 += 3 # rotate by 3 degrees each frame
155.         degrees2 += 7 # rotate by 7 degrees each frame
156.
157.
158. def terminate():
159.     pygame.quit()
160.     sys.exit()
161.
162.
163. def getRandomLocation():
164.     return {'x': random.randint(0, CELLWIDTH - 1), 'y':
random.randint(0, CELLHEIGHT - 1)}
165.
166.
167. def showGameOverScreen():
168.     gameOverFont = pygame.font.Font('freesansbold.ttf', 150)
169.     gameSurf = gameOverFont.render('Game', True, WHITE)
170.     overSurf = gameOverFont.render('Over', True, WHITE)

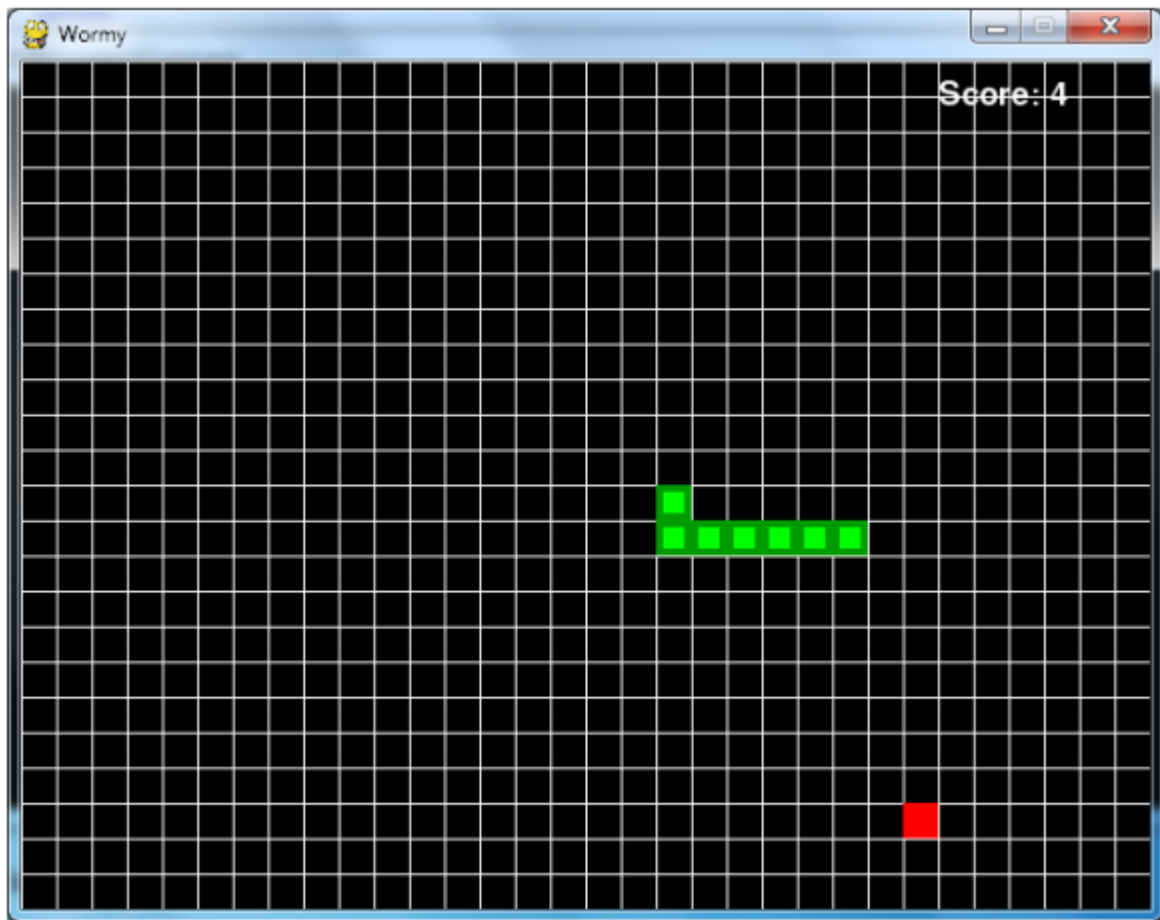
```

```

171.     gameRect = gameSurf.get_rect()
172.     overRect = overSurf.get_rect()
173.     gameRect.midtop = (WINDOWWIDTH / 2, 10)
174.     overRect.midtop = (WINDOWWIDTH / 2, gameRect.height + 10 + 25)
175.
176.     DISPLAYSURF.blit(gameSurf, gameRect)
177.     DISPLAYSURF.blit(overSurf, overRect)
178.     drawPressKeyMsg()
179.     pygame.display.update()
180.     pygame.time.wait(500)
181.     checkForKeyPress() # clear out any key presses in the event
queue
182.
183.     while True:
184.         if checkForKeyPress():
185.             pygame.event.get() # clear event queue
186.             return
187.
188. def drawScore(score):
189.     scoreSurf = BASICFONT.render('Score: %s' % (score), True,
WHITE)
190.     scoreRect = scoreSurf.get_rect()
191.     scoreRect.topleft = (WINDOWWIDTH - 120, 10)
192.     DISPLAYSURF.blit(scoreSurf, scoreRect)
193.
194.
195. def drawWorm(wormCoords):
196.     for coord in wormCoords:
197.         x = coord['x'] * CELLSIZE
198.         y = coord['y'] * CELLSIZE
199.         wormSegmentRect = pygame.Rect(x, y, CELLSIZE, CELLSIZE)
200.         pygame.draw.rect(DISPLAYSURF, DARKGREEN, wormSegmentRect)
201.         wormInnerSegmentRect = pygame.Rect(x + 4, y + 4, CELLSIZE -
8, CELLSIZE - 8)
202.         pygame.draw.rect(DISPLAYSURF, GREEN, wormInnerSegmentRect)
203.
204.
205. def drawApple(coord):
206.     x = coord['x'] * CELLSIZE
207.     y = coord['y'] * CELLSIZE
208.     appleRect = pygame.Rect(x, y, CELLSIZE, CELLSIZE)
209.     pygame.draw.rect(DISPLAYSURF, RED, appleRect)
210.
211.
212. def drawGrid():
213.     for x in range(0, WINDOWWIDTH, CELLSIZE): # draw vertical lines
214.         pygame.draw.line(DISPLAYSURF, DARKGRAY, (x, 0), (x,
WINDOWHEIGHT))
215.     for y in range(0, WINDOWHEIGHT, CELLSIZE): # draw horizontal
lines
216.         pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, y),
(WINDOWWIDTH, y))
217.
218.
219. if __name__ == '__main__':
220.     main()

```

Координатна мрежа (The Grid)



Ако мало боље погледате приликом играња игре, приметићете да црвена тачка, као и тело змије, потпуно упадају, односно уклапају се са мрежом квадратића која је исцртана у позадини. Сваки од ових квадратића назваћемо ћелијом. Наравно, ово име смо ми одабрали, није увек случај да се поље у оваквој мрежи назива ћелијом. Ћелије представљене на овај начин, имају своје координате у замишљеном координатном систему. Крајња горња лева ћелија има координате (0, 0), док крајња доња десна ћелија има координате (31, 23).

Код за почетна подешавања

```
1. # Wormy (a Nibbles clone)
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 = 15
10. WINDOWWIDTH = 640
11. WINDOWHEIGHT = 480
12. CELLSIZE = 20
13. assert WINDOWWIDTH % CELLSIZE == 0, "Window width must be a
    multiple of cell size."
14. assert WINDOWHEIGHT % CELLSIZE == 0, "Window height must be a
    multiple of cell size."
15. CELLWIDTH = int(WINDOWWIDTH / CELLSIZE)
16. CELLHEIGHT = int(WINDOWHEIGHT / CELLSIZE)

```

Код на почетку служи за сетовање константи које ће се користити касније. Висини и ширина ћелија су смештене константи ЦЕЛСИЗЕ. Константе `WINDOWWIDTH` и `WINDOWHEIGHT` садрже вредности за ширину и висину прозора. Команда **асерт** која се користи у линијама 13 и 14, служи да осигура да се све ћелије ширински и висински идеално уклопе у димензије прозора. Наиме, уколико израз након команде асерт има вредност **фалсе**, програм се прекида и пријављује се грешка. Уколико је услов задовољен, може се наставити са извршавањем програма. Примера ради, уколико је величина ћелије 10, а ширина прозора 15, само 1.5 ћелија би се уклопило, а то не желимо да дозволимо. Команда асерт осигурава да се само целобројни бројеви ћелија могу користити у мрежи.

```

18. #           R       G       B
19. WHITE      = (255, 255, 255)
20. BLACK      = ( 0,   0,   0)
21. RED        = (255,  0,   0)
22. GREEN      = ( 0, 255,  0)
23. DARKGREEN  = ( 0, 155,  0)
24. DARKGRAY   = ( 40,  40,  40)
25. BGCOLOR = BLACK
26.
27. UP = 'up'
28. DOWN = 'down'
29. LEFT = 'left'
30. RIGHT = 'right'
31.
32. HEAD = 0 # syntactic sugar: index of the worm's head

```

На слици изнад приказане су константе које се користе за дефинисање РГБ вредности боја, као и константе које ће се користити код одређивања смерова кретања змије. Константа `HEAD` ће бити касније објашњена.

Функција `main()`

```

34. def main():
35.     global FPSCLOCK, DISPLAYSURF, BASICFONT
36.
37.     pygame.init()
38.     FPSCLOCK = pygame.time.Clock()

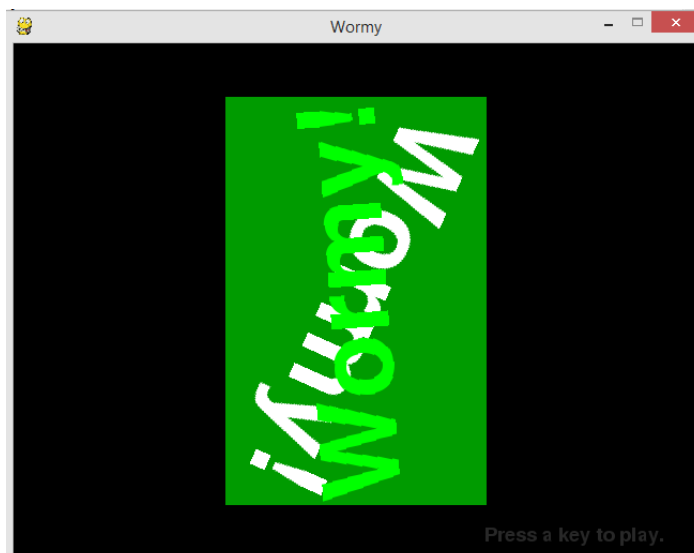
```

```

39.     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH,
        WINDOWHEIGHT))
40.     BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
41.     pygame.display.set_caption('Wormy')
42.
43.     showStartScreen()
44.     while True:
45.         runGame()
46.         showGameOverScreen()

```

Главни део програма наше игрице, сместићемо у функцију **runGame()**. Као што видите на слици изнад, после обављених подешавања, следи део који почиње функцијом **showStartScreen()**. Наиме, жеилмо да када корисник покрене нашу игрицу, на почетку добије екран на ком се ротирајућим словима исписује „Wormy!“.



Овакав приказ постоји само при првом покретању игрице и код који га приказује ћемо сместити у функцију **showStartScreen()**. Као што видите на слици изнад, ову функцију позивамо само једном пре главне **while** петље.

Након приказа почетног екрана, жеилимо да корисник игра игру све док не дође до краја (убије се). То и јесте главна ствар у игрици. Код који ће то омогућити ћемо сместити у функцију **runGame()**. Зато се позив ове функције налази на врху петље. Дакле, из функције **runGame()** се излази када се змија убије. Када се то деси, желимо да кориснику прикажемо Game Over екран.



Притиском било ког тастера, корисник ће поново моћи да игра игру. Функција која приказује Гаме Овер екран се зове **showGameOverScreen()** и из ње се излази када корисник притисне било који тастер. Тада поново желимо да покренемо игру, дакле поново позивамо функцију **runGame()**. Ове две функције се смењују наизменично у главној **while** петљи у линији 44 и тако све док корисник не затвори прозор игрице.

Засебна функција `runGame()`

```
49. def runGame():
50.     # Set a random start point.
51.     startx = random.randint(5, CELLWIDTH - 6)
52.     starty = random.randint(5, CELLHEIGHT - 6)
53.     wormCoords = [{'x': startx,      'y': starty},
54.                   {'x': startx - 1, 'y': starty},
55.                   {'x': startx - 2, 'y': starty}]
56.     direction = RIGHT
57.
58.     # Start the apple in a random place.
59.     apple = getRandomLocation()
```

На самом почетку игрице, обезбедићемо да змија крене од насумичне позиције на табли, а да притом то не буде превише близу ивице прозора. У променљивама `startx` и `starty` ћемо сачувати почетне координате главе змије. Присетимо се да константе `CELLWIDTH` и `CELLHEIGHT` представљају број ћелија ширински и висински које постоје на екрану, не величину екрана у пикселима.

Тело наше змије, односно, координате ћелија на којима се тренутно налази тело, чуваћемо унутар повезане листе (list of dictionary values) под називом **wormCoords**. За сваки сегмент односно ћелију на којој се налази део тела змије, постојаће један елемент у оквиру наше листе. Сваки елемент ће имати своје кључеве **x** и **y** будући да ћемо памтити по две вредности које ће представљати **XY** координате ћелије на којој се тренутно налази део тела змије.

За почетак, глава змије ће се налазити на координатама **startx** и **starty** које смо насумично одредили изнад. За почетак, змија ће бити дугачка 3 сегмента, односно, заузимаће

3 ћелије. Биће то глава и две наредне ћелије лево од главе, па ћемо одговарајуће елементе убацити одмах у нашу листу.

Глава змије ће се увек налазити на почетку листе **wormCoords** односно, биће представљена са **wormCoords[0]**. Да би наш код био читљивији, на почетку, на линији 32 смо направили константу **HEAD** и сетовали смо јој вредност на 0, тако да ћемо надаље уместо **wormCoords[0]** користити **wormCoords[HEAD]**.

Петља за обраду догађаја

```
61.     while True: # main game loop
62.         for event in pygame.event.get(): # event handling loop
63.             if event.type == QUIT:
64.                 terminate()
65.             elif event.type == KEYDOWN:
66.                 if (event.key == K_LEFT or event.key == K_a) and
direction != RIGHT:
67.                     direction = LEFT
68.                 elif (event.key == K_RIGHT or event.key == K_d) and
direction != LEFT:
69.                     direction = RIGHT
70.                 elif (event.key == K_UP or event.key == K_w) and
direction != DOWN:
71.                     direction = UP
72.                 elif (event.key == K_DOWN or event.key == K_s) and
direction != UP:
73.                     direction = DOWN
74.                 elif event.key == K_ESCAPE:
75.                     terminate()
```

У линији 61 почиње главна петља наше игре. У линији 62, почиње петља за обраду догађаја у игрици. Уколико се јави догађај типа **QUIT**, позивамо функцију **terminate()** која прекида комплетан програм игрице.

У супротном, у питању ће бити промена правца кретања змије. Прва на реду је провера догађаја **KEYDOWN**. У овом случају проверавамо да ли је притиснута стрелица на доле или дугме **С** из WASD контрола. Након ове провере, проверавамо да ли је издата наредба за кретање змије у супротном смеру на истом правцу којим се змија тренутно креће. У овом случају би змија одмах погодила саму себе и игра би се завршила на не баш елегантан начин. Примера ради, уколико се змија креће у лево и играч случајно притисне тастер за скретање у десно, змија би одмах погодила саму себе и игра би била готова одмах. Не желимо да дозволимо овако нешто.

Зато стално проверавамо вредност променљиве **direction**. Уколико играч изда команду којом би змија одмах ујела саму себе, ту команду ћемо једноставно игнорисати.

На самом крају само још проверавамо да ли је стиснуто дугме Есцапе. У том случају такође позивамо функцију **terminate()** и у потпуности прекидамо програм игрице.

Детекција судара

```
77. # check if the worm has hit itself or the edge
78.     if wormCoords[HEAD]['x'] == -1 or wormCoords[HEAD]['x'] ==
CELLWIDTH or wormCoords[HEAD]['y'] == -1 or wormCoords[HEAD]['y'] ==
CELLHEIGHT:
79.         return # game over
80.         for wormBody in wormCoords[1:]:
81.             if wormBody['x'] == wormCoords[HEAD]['x'] and
wormBody['y'] == wormCoords[HEAD]['y']:
82.                 return # game over
```

На слици изнад приказан је део кода којим проверавамо да ли је дошло до краја игре тако што је змија ударила о зид или је глава змије погодила неки сегмент који покрива тело змије, то јест, да ли је змија ујела саму себе. На ова два начина, може се завршити игра.

Прво проверавамо да ли је змија ударила о зид. Као прво, проверићемо да ли је глава змије изашла са леве односно са десне стране са наше координатне мреже која представља екран за игру. Довољно је да проверимо вредност **X** координате главе змије коју можемо да добијемо помоћу **wormCoords[HEAD]['x']**. Уколико је ова вредност једнака **-1**, то значи да је глава змије изашла са **леве** стране. Уколико је ова вредност једнака **CELLWIDTH**, то значи да је глава змије изашла из наше дефинисане мреже са **десне** стране, односно да је ударило о десни зид. Наравно, у оба случаја се прекида партија.

Слично провери левог и десног зида, проверавамо да ли је змија можда ударило о зид са горње односно са доње стране. Информација која нам је потребна јесте вредност **Y** координате главе наше змије коју можемо добити као **wormCoords[HEAD]['y']**. Уколико је ова вредност **-1**, змија је ударила о зид са **горње** стране. Уколико је ова вредност једнака **CELLHEIGHT**, змија је ударила о зид са доње стране. Подсетимо се да консанте **CELLWIDTH** и **CELLHEIGHT** представљају бројеве сегмената на нашој координатној мрежи по ширини односно по висини респективно.

Уколико било који од ових услова буде испуњен, значи да је змија ударила о зид и потребно је прекинути партију. Све што треба урадити је излазак из функције **runGame()** (линија 45) која представља тренутну партију. То ћемо једноставно постићи позивом команде **return**, линија 79. Када се изађе из ове функције, враћамо се у главну петљу, линија 44 и наредна функција која се одмах позива јесте **showGameOverScreen()** на линији 46. Управо ово смо и хтели. Када дође до судара змије са зидом, прекида се партија и приказује се екран за крај партије.

Наравно, поред судара са зидом, потребно је проверити и да ли је змија угризла саму себе. То је следеће што ћемо проверити. На линији 80 налази се **for** петља која ће проћи кроз све сегменте тела наше змије не рачунајући главу. Због тога стоји да **for** петља итерира кроз **wormCoords[1 :]** уместо само кроз **wormCoords**, будући да је на позицији 0 представљен елемент листе који чува информације о глави змије.

Уколико се поклопе координате било ког дела тела змије са главом змије, дакле обе вредности **wormBody['x']** са **wormCoords[HEAD]['x']** и **wormBody['y']** са **wormCoords[HEAD]['y']**, то значи да је змија угризла саму себе и да поново треба прекинути партију изласком из функције **runGame()**. То ћемо поново једноставно урадити позивом команде **return**, линија 82.

Детекција судара са циљем (црвеном тачком)

```
84. # check if worm has eaten an apply
85.     if wormCoords[HEAD]['x'] == apple['x'] and
        wormCoords[HEAD]['y'] == apple['y']:
86.         # don't remove worm's tail segment
87.         apple = getRandomLocation() # set a new apple somewhere
88.     else:
89.         del wormCoords[-1] # remove worm's tail segment
```

У овом делу кода, проверавамо да ли је дошло до судара змије са циљем који треба да поједе. Урадићемо слично као и до сада, проверу координата главе змије са координатама циља. Уколико се поклопе вредности X и Y координата, змија је погодила циљ и сада одређујемо ново насумично место на екрану на ком ће се појавити нови циљ. Дакле, насумично одређујемо нове координате циља за шта ћемо користити функцију `getRandomLocation()`, линија 87.

Уколико се глава змије није сударила са циљем, обрисаћемо последњи елемент у листи `wormCoords`, дакле, последњи сегмент тела змије, линија 89. Подсетимо се да негативни индекси броје од краја листе. Како је 0 индекс првог елемента, 1 индекс другог, тако је -1 индекс **последњег** елемента, -2 претпоследњег и тако даље.

Код између линија 91 и 100, приказан на наредној слици, описује кретање змије. Дакле, без обзира на то да ли је змија погодила циљ или не, додаћемо нови сегмент тела змији (нову главу) у правцу кретања змије. Уколико је змија погодила циљ, као што смо рекли, нећемо обрисати последњи сегмент њеног тела, па додавањем још једног сегмента на почетку (нова глава), змија ће постати за један сегмент дужа, што нам је и био циљ. Са друге стране, уколико није дошло до судара са циљем, обрисаћемо последњи сегмент тела змије, линија 89, тако да ће после додавања новог сегмента на почетку змије, дужина само змије остати непромењена. Како нову главу додајемо змији у правцу њеногкретања, а бришемо последњи сегмент, добијамо визуелни ефекат кретања змије о екрану.

Кретање змије

```
91. # move the worm by adding a segment in the direction it is moving
92.     if direction == UP:
93.         newHead = {'x': wormCoords[HEAD]['x'], 'y':
            wormCoords[HEAD]['y'] - 1}
94.     elif direction == DOWN:
95.         newHead = {'x': wormCoords[HEAD]['x'], 'y':
            wormCoords[HEAD]['y'] + 1}
96.     elif direction == LEFT:
97.         newHead = {'x': wormCoords[HEAD]['x'] - 1, 'y':
            wormCoords[HEAD]['y']}
98.     elif direction == RIGHT:
99.         newHead = {'x': wormCoords[HEAD]['x'] + 1, 'y':
            wormCoords[HEAD]['y']}
```

```
100. wormCoords.insert(0, newHead)
```

Као што смо већ рекли, кретање змије постижемо додавањем новог сегмента на почетку листе **wormCoords** који ће представљати нову главу змије. Јако је битно да нову главу додамо на основу правца у ком се змија креће. Нова глава ће дакле, сигурно бити додата ћелију која је суседна ћелији на којој се тренутна глава налази. У зависности од смера и правца кретања змије, нова глава ће се сместити на ћелију изнад, испод, лево или десно од ћелије на којој се тренутна глава налази. Дакле, координате нове главе ће се разликовати од координата тренутне за једну вредност. Биће то вредност X или Y координате, повећана односно смањена за 1 у зависности од смера и правца кретања змије. Кретање на горе значиће смањење Y за 1, кретање на доле значиће повећање Y за 1, кретање на лево смањује X за 1, док кретања на десно повећава X за 1. Координате нове главе памтимо помоћу **newHead**.

Нову главу додајемо на почетак листе **wormCoords** помоћу функције **insert()**, линија 100.

Функција за рад са листама - **insert()**

За разлику од **append()** функције која додаје елементе искључиво на крај листе, функција **insert()** може додати елемент на било коју позицију у листи. Ова функција прихвата два параметра. Први параметар означава индекс позиције на коју се додаје нови елемент (по дефалт-у, сви наредни елементи имају индексе повећане за један у односу на претходни). Уколико је прослеђени параметар већи од дужине саме листе, нови елемент се једноставно додаје на крај листе као што **append()** ради. Други параметар је елемент који се додаје у листу. На следећој слици је приказано пар примера како бисте се боље упознали са функцијом **insert()**. Откуцајте их у интерактивном шелл-у.

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(0, 'frog')
>>> spam
['frog', 'cat', 'dog', 'bat']
>>> spam.insert(10, 42)
>>> spam
['frog', 'cat', 'dog', 'bat', 42]
>>> spam.insert(2, 'horse')
>>> spam
['frog', 'cat', 'horse', 'dog', 'bat', 42]
```

Исцртавање екрана

```
101. DISPLAYSURF.fill(BG_COLOR)
102. drawGrid()
103. drawWorm(wormCoords)
104. drawApple(apple)
105. drawScore(len(wormCoords) - 3)
106. pygame.display.update()
107. FPSCLOCK.tick(FPS)
```

Код за исцртавање екрана налази се у оквиру функције **runGame()** и јако је једноставан. Команда у линији 101 испуњава екран позадинском бојом. У линијама 102 – 105 исцртавају

мрежу, змију, циљ (црвену јабуку) и број освојених поена (сцоре) на екрану. Позивом **pygame.display.update()**, заправо исцртавамо на стварном екрану све што смо дефинисали функцијама изнад.

Исписивање „Press a key“ текста на екрану

```
109. def drawPressKeyMsg():
110.     pressKeySurf = BASICFONT.render('Press a key to play.', True,
        DARKGRAY)
111.     pressKeyRect = pressKeySurf.get_rect()
112.     pressKeyRect.topleft = (WINDOWWIDTH - 200, WINDOWHEIGHT - 30)
113.     DISPLAYSURF.blit(pressKeySurf, pressKeyRect)
```

Приликом приказивања анимације на почетном екрану, приликом првог стартовања игре, као и приликом приказивања екрана за крај игре (Гаме Овер), у доњем десном углу екрана се налази текст „Пресс а кеу то плаи“. Како не бисмо исти код куцали два пута у функцијама **showStartScreen()** и **showGameOverScreen()**, овај код смо сместили у засебну функцију **drawPressKeyMsg()** коју ћемо касније једноставно позивати када год нам је потребно исписивање ове поруке на екрану.

Функција **checkForKeyPress()**

```
116. def checkForKeyPress():
117.     if len(pygame.event.get(QUIT)) > 0:
118.         terminate()
119.
120.     keyUpEvents = pygame.event.get(KEYUP)
121.     if len(keyUpEvents) == 0:
122.         return None
123.     if keyUpEvents[0].key == K_ESCAPE:
124.         terminate()
125.     return keyUpEvents[0].key
```

У овој функцији, проверавамо шта се налази у реду догађаја који постоји дефинисан у позадини и у који се пакују сви догађаји који се јављају. За почетак, проверавамо да ли постоји неки догађај типа **QUIT**. Позив **pygame.event.get()** на линији 117 враћа листу догађаја типа **QUIT** будући да смо проследили параметар **QUIT** функцији. Иначе би вратио листу свих догађаја. Једноставно проверавамо дужину враћене листе. Уколико је дужина већа од 0, значи да постоји **QUIT** догађај и онда прекидамо програм функцијом **terminate()**, линија 118. Уколико нема **QUIT** догађаја, овај позив ће вратити празну листу.

Након овога, претражујемо ред догађаја у потрази за догађајима типа **KEYUP** који се јављају пуштањем притиснутог тастера. Поново проверавамо дужину враћене листе, линија 121.

Уколико је 0, нема излаза из функције, враћа се вредност **None**. Уколико није 0, проверавамо да ли је у питању тастер Есц. Уколико јесте, прекидамо програм, линија 124. Уколико није у питању Есц тастер, враћамо догађај који смо преузели из реда догађаја као излаз из функције **checkForKeyPress()**. Као што видите, враћамо први догађај из листе (`keyUpEvents[0]`). У листи ће свакако постојати само тај један догађај будући да се догађај обрађује чим се јави, пре него што се јави неки наредни KEYUP догађај.

Почетни екран

```
128. def showStartScreen():
129.     titleFont = pygame.font.Font('freesansbold.ttf', 100)
130.     titleSurf1 = titleFont.render('Wormy!', True, WHITE, DARKGREEN)
131.     titleSurf2 = titleFont.render('Wormy!', True, GREEN)
132.
133.     degrees1 = 0
134.     degrees2 = 0
135.     while True:
136.         DISPLAYSURF.fill(BGCOLOR)
```

Сада ћемо приказати функцију **showStartScreen()** коју смо помињали раније и у којој ћемо обезбедити приказ почетног екрана са ротирајућим словима.

Приликом првог покретања игре, артија не почиње одмах, већ се играчу приказује почетни екран. На овај начин, играчу се пружају додатне информације о самој игрици, а такође, играчу се даје време да се припреми за партију. Уколико би партија одмах почела, играч би могао да изгуби живот у игрици јак брзо јер није био спреман да крене да игра.

На почетном екрану имаћемо два објекта на којима ће бити исписан текст „Wormy“. Креираћемо их позивом уграђене функције **render()**. Текст ће бити велики. **Font** конструктором креирамо фонт величине 100 пт. Наглашавамо и који фонт користимо за слова. Први натпис Wormy ће бити обијен у белу боју и имаће тамно зелену позадину, док ће други бити зелене боје са провидном позадином. Овиме смо креирали објекте које ћемо приказати.

Међутим, није нам довољно што смо само креирали објекте будући да не желимо да се они приказују статично попут слике. Желимо да ова два објекта на екрану ротирају све време док је приказан почетни екран. Линијом 135 креће петља којом ћемо вртети анимацију коју ћемо направити како бисмо добили ротацију слова.

Ротирање слова на почетном екрану

```
137.     rotatedSurf1 = pygame.transform.rotate(titleSurf1, degrees1)
138.     rotatedRect1 = rotatedSurf1.get_rect()
139.     rotatedRect1.center = (WINDOWWIDTH / 2, WINDOWHEIGHT / 2)
140.     DISPLAYSURF.blit(rotatedSurf1, rotatedRect1)
141.
142.     rotatedSurf2 = pygame.transform.rotate(titleSurf2,
143.         degrees2)
143.     rotatedRect2 = rotatedSurf2.get_rect()
```

```

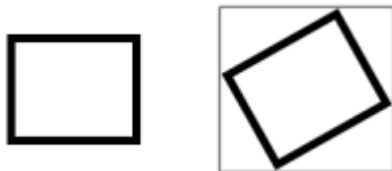
144.         rotatedRect2.center = (WINDOWWIDTH / 2, WINDOWHEIGHT / 2)
145.         DISPLAYSURF.blit(rotatedSurf2, rotatedRect2)
146.
147.         drawPressKeyMsg()
148.
149.         if checkForKeyPress():
150.             pygame.event.get() # clear event queue
151.             return
152.         pygame.display.update()
153.         FPSLOCK.tick(FPS)

```

Као што смо већ рекли, ова функција ће нам омогућити ротацију слова на почетном екрану. За ротирање објеката, користи се уграђена функција **pygame.transform.rotate(n1, n2)**. Први параметар ове функције представља објекат који се ротира, док други представља број степени за колико желимо да извршимо ротацију. Битно је напоменути да ова функција не мења стварно објекат који јој прослеђујемо као први параметар, већ прави копију тог објекта заротирану за број степени који јој прослеђујемо као други параметар.

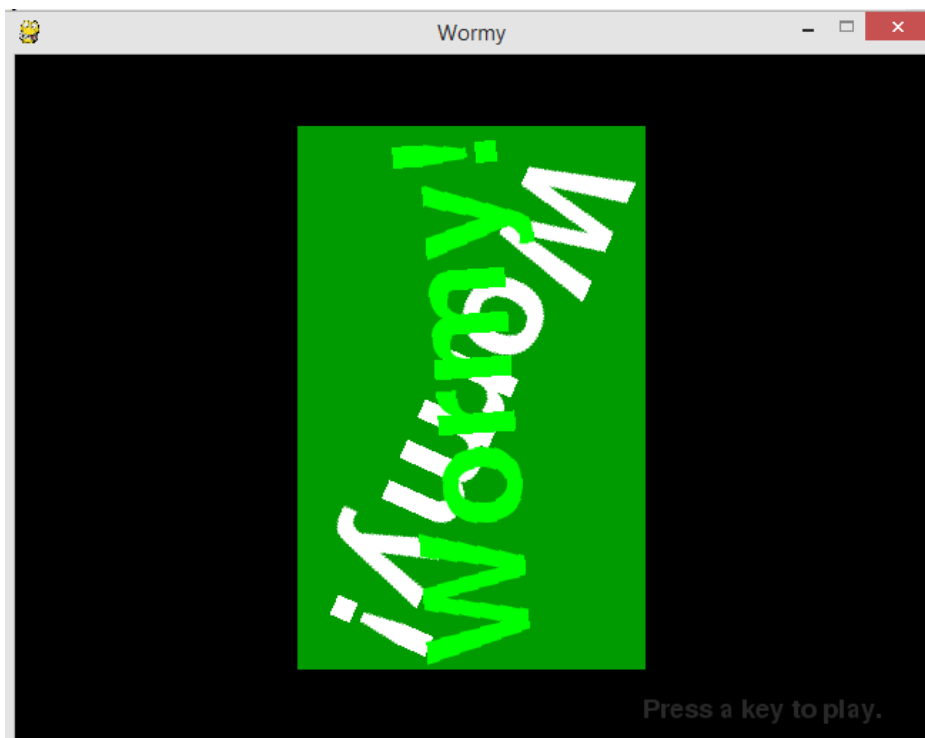
Поред слова која исписујемо, желимо да ротирамо и позадину око тих слова, тачније читав објекат на којем су слова „Wormy“ исписана. Сви објекти представљени су правоугаоником. У линијама 138 и 143 издвајамо правоугаонике који представљају позадине наших слова.

Приметите да ће позадина коју заузимају наши објекти са словима, после ротације постати већа. Наиме, сви објекти представљени су правоугаоником. После ротације оригиналног правоугаоника, његова темена ће прећи границе на којима су била ширински и висински пре ротације. Слика испод говори о овој чињеници.



Оригинални квадрат лево заузима мање простора него такав исти квадрат после ротације. Будући да ротирамо слова, као и позадину објеката, дакле, слова и квадрате који представљају позадину, визуелно ће се, док траје анимација, видети како квадрати у позадини мењају своје димензије. Међутим, присетимо се да један од објеката које смо креирали има зелену позадину са белим словима на себи, док други има зелена слова са провидном позадином, линија 131. Дакле, приликом трајања анимације, само зелени квадрат који представља позадину првог објекта ће се видети како ротира, а због чињенице коју смо описали изнад, визуелно ће се добити ефекат да тај зелени квадрат мења своју величину непрестано будући да копије заузимају више, односно мање места од оригинала после ротација непрестано у круг.

На линијама 140 и 145, уграђеном функцијом **blit()**, спајамо и приказујемо слике слова и квадрата после одрађених ротација на истом месту у исто време. Дакле, постоје 4 слике које се приказују, али будући да један квадрат има провидну позадину, видеће се три слике. Кроз петљу непрестано ротирамо слова и квадрате који мењају своје димензије, тачније однос ширине и висине.

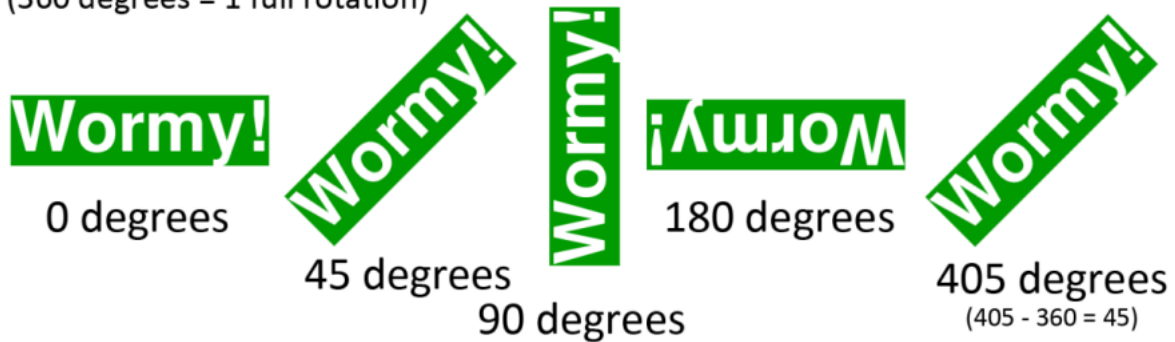


Величина ротације која ће се обавити дата је у степенима и прослеђује се као други параметар функцији **pygame.transform.rotate(n1, n2)**. Као што знате пун круг сарџи 360 степени. Уколико се проследи 0, неће се ротирати уопште. Ротација за једну четвртину круга у смеру супротном од смера кретања казаљке на сату, добила би се прослеђивањем вредности 90. Смер кретања казаљке на сату би се добио прослеђивањем негативне вредности броја степени ротације. Ротирање слике за 360 степени би дало за резултат почетну слику, дакле као да је ротирана за 0 степени. Заправо, уколико се проследи број који је већи или једнак 360, ПуГаме ће аутоматски одузимати 360 од те вредности све док не добије вредност мању од 360 и за ту вредност ће онда извршити ротацију.

На слици испод су приказани примери ротација за одређен број степени.

Rotation Examples:

(360 degrees = 1 full rotation)



На линији 147 позивамо функцију **drawPressKeyMsg()** којом исписујемо на екрану поруку „Пресс а кеу то плау“, о чему је било речи раније. Петља наше анимације, линија 135 ће наставити да ради све док функција **checkForKeyPress()** коју смо раније објаснили, врати вредност која није **None** што ће се десити када играч притисне неки тастер на тастатури који није Есц тастер којим се програм протпунп зауставља. Уколико услов на линији 149 прође, значи да је играч спреман да започне партију. Позив функције **pygame.event.get()** служи да очисти ред догађаја потпуно од догачаја који су се можда десили током приказивања почетног екрана пре него што се отпочне са партијом. након тога, команда **return** излази из петље и функција **showStartScreen()** се овим завршава. Након тога се улази у главну петљу програма, линија 44 и надаље се смењују функције **runGame()** и **showGameOverScreen()**. Уколико услов на линији 149 не прође, наставља се са приказом почетног екрана и са ротирањем наших објеката односно, анимација и даље траје.

Реализација ротација. Ротације које нису савршене

Вероватно сте приметили да приликом извршавања ротација, на линијама 137 и 142, користимо нове променљиве **rotatedSurf1** и **rotatedSurf2** за чување ротираних слика објеката који се приказују. Дакле, ми у свакој итерацији петље у ствари ротирамо оригиналне објекте **titleSurf1** и **titleSurf2** за број степени који **повећавамо** у свакој наредној итерацији петље. Можда сте се запитали, зашто не бисмо чували сваки ротирани објекат па у наредној итерацији ротирали за исти број степени тај који смо добили ротацијом у претходној итерацији петље? Тако не бисмо морали да повећавамо степене ротације и имали бисмо само 2 променљиве за чување објеката уместо 4.

За ово постоје 2 разлога:

1. Ротирање 2Д слике никада није савршено. Ротирајућа слика никада није ротирана за тачно онолико степени колико смо задали, већ приближно томе. Примера ради, уколико бисте неку слику ротирали за 10 степени у једном смеру, а потом ту ротирану копију ротирали за 10 степени у супротном смеру, не бисте добили слику идентичну оној од које сте почели, то јест, оригиналну слику. Овај проблем можете слично посматрати као проблем фотокопирања. Уколико бисте фотокопирали неки документ, па потом фотокопирали копију коју сте добили претходним фотокопирањем и тако даље, квалитет сваке наредне добијене копије би се константно смањивао. Једини случај када овај проблем не би постојао јесте ротирање слике за неки умножак од 90 степени, као на пример, 0, 90, 180, 270, 360 степени. Једино у овом случају, пиксели могу да се ротирају савршено без искривљености. Код нас међутим, ово неће бити случај.
2. Уколико ротирате 2Д слику, добијена копија ће бити мало већа од оригинала. Уколико потом ту копију поново ротирате, добићете још већу копију. Ако овако наставите, после неког времена, копија ће бити превелика за приказ. У нашем случају, објекат ће бити превелики и добићете грешку **pygame.error: Width or height is too large**.

```
154. degrees1 += 3 # rotate by 3 degrees each frame
155.         degrees2 += 7 # rotate by 7 degrees each frame
```

Због описаних проблема, ми ћемо ротације увек вршити над оригиналним објектима тако што ћемо у свакој наредној итерацији петље, повећавати вредност броја степени за који узвршавамо ротацију. Када ти бројеви постану већи од 360, ПуГаме ће одузимати вредност 360 од броја степени који се задаје, као што смо објаснили раније. Број степениза које ћемо вршити ротације ћемо чувати у променљивама **degrees1** и **degrees2**.

Још једна ствар коју желимо да обезбедимо јесте да се текст обојен белом бојом ротира спорије од текста обојеног зеленом бојом. Зато променљиву **degrees2** повећавамо за 7 у свакој наредној итерацији док променљиву **degrees1** повећавамо за 3. На овај начин постижемо да визуелно изгледа као да се зелена слова ротирају брже.

```
158. def terminate():
159.     pygame.quit()
160.     sys.exit()
```

Функција **terminate()** зауставља потпуно коректно програм игрице тако што прво гаси библиотеку ПуГаме, а потом и затвара прозор игрице и потпуно прекида њен програм.

Нова појављивања циља (јабуке)

```
163. def getRandomLocation():
164.     return {'x': random.randint(0, CELLWIDTH - 1), 'y':
        random.randint(0, CELLHEIGHT - 1)}
```

Функција **getRandomLocation()** служи да насумично одреди нове координате циља када преходни циљ буде погођен. Она се позива када годје потребно приказати циљ на новом месту на екрану. Она враћа уређени пар XY координата у виду елемента који се може одмах паковати у листо **wormCoords**.

Екран за крај игре (Game Over Screen)

```
167. def showGameOverScreen():
168.     gameOverFont = pygame.font.Font('freesansbold.ttf', 150)
169.     gameSurf = gameOverFont.render('Game', True, WHITE)
170.     overSurf = gameOverFont.render('Over', True, WHITE)
171.     gameRect = gameSurf.get_rect()
172.     overRect = overSurf.get_rect()
173.     gameRect.midtop = (WINDOWWIDTH / 2, 10)
174.     overRect.midtop = (WINDOWWIDTH / 2, gameRect.height + 10 + 25)
175.
```

```

176.     DISPLAYSURF.blit(gameSurf, gameRect)
177.     DISPLAYSURF.blit(overSurf, overRect)
178.     drawPressKeyMsg()
179.     pygame.display.update()

```

Екран за крај игре направљен је слично екрану за почетак игре, осим што овај сада нема анимација на себи. Једноставно, две речи **Game** и **Over** су исписане на два објекта која су после исцртана на екрану. Дакле, идентично објектима са речима **Wormy** из претходног наслова, уз чињеницу да у овом случају немамо никаквих анимација већ само прост приказ на екрану (линије 176 и 177).

```

180. pygame.time.wait(500)
181.     checkForKeyPress() # clear out any key presses in the event
    queue
182.
183.     while True:
184.         if checkForKeyPress():
185.             pygame.event.get() # clear event queue
186.             return

```

Екран за крај игре приказује се када играч изгуби живот у игри. Овај екран остаје приказан све док играч не притисне било који тастер осим Есц тастера на тастатури.

Обратимо сада пажњу на линије 180 и 181. Ове две линије имају улогу да спрече случајно покретање нове партије. У линији 180, функцијом **pygame.time.wait()**, паузирали смо извршавање програма на 500 милисекунди како бисмо обезбедили да играч не притисне неки тастер превише рано.

Након тога, функција **checkForKeyPress()** је позвана како би обезбедила да сви догађаји (кеу евентс) који су се десили од почетка функције **showGameOverScreen()** буду игнорисани. Пауза коју смо увели, као и позив функције **checkForKeyPress()** су убачени у програм како би омогућили да играч не притисне случајно превише рано неки тастер и тако започне неспреман нову партију. Размотримо следећу ситуацију: Рецимо да играч прилази зиду и покушава да избегне зид тако што притисне неку од стрелица. Међутим, деси се да је играч притиснуо стрелицу касно и већ је ударио о зид. Истог тренутка се позива функција **showGameOverScreen()** и догађај притиснутог тастера би се заправо регистровао **након** позива функције **showGameOverScreen()**. Тада би овај догађај довео до моменталног гашења екрана за крај игре и до покретања нове партије истог тренутка као што смо и дефинисали раније. Једноставно, притисак тастера би се регистровао у погрешно време. Нова партија би почела прерано и играч не би био спреман да крене да игра. Линије 180 и 181 служе да би се избегле овакве ситуације, то јест, да би игра била више „усер фриендлу“ то јест, захвалнија за играње.

Када прођу линије 180 и 181, у петљи на линији 183 се чека притисак било ког тастера, али не случајно, како би се отпочела нова партија.

Функције за исцртавања

Код за исцртавање резултата, змије, циља (јабуке) и мреже у позадини, смештени су у засебним функцијама.

```

188. def drawScore(score):
189.     scoreSurf = BASICFONT.render('Score: %s' % (score), True,
    WHITE)
190.     scoreRect = scoreSurf.get_rect()
191.     scoreRect.topleft = (WINDOWWIDTH - 120, 10)
192.     DISPLAYSURF.blit(scoreSurf, scoreRect)

```

Функција за испис резултата на екрану је јако једноставна. Она просто исписује резултат који се чува у променљивој **score**.

```

195. def drawWorm(wormCoords):
196.     for coord in wormCoords:
197.         x = coord['x'] * CELLSIZE
198.         y = coord['y'] * CELLSIZE
199.         wormSegmentRect = pygame.Rect(x, y, CELLSIZE, CELLSIZE)
200.         pygame.draw.rect(DISPLAYSURF, DARKGREEN, wormSegmentRect)
201.         wormInnerSegmentRect = pygame.Rect(x + 4, y + 4, CELLSIZE -
    8, CELLSIZE - 8)
202.         pygame.draw.rect(DISPLAYSURF, GREEN, wormInnerSegmentRect)

```

Функција **drawWorm()** је задужена за исцртавање тела змије на екрану. Она ће исцртати зелени квадратић за сваки сегмент тела змије. Тело змије, тачније координате ћелија позадинске мреже на којима се налази тело змије, прослеђује се у виду листе **wormCoords** која садржи елементе са кључевима **x** и **y**. Петља на линији 196 пролази кроз све елементе листе.

Међутим, да бисмо исцртали квадрате, нама није довољно да знамо координате сегмената у мрежи, већ њихове координате у пикселима на екрану. Како наша мрежа заузима цео прозор игрице и такође почиње од вредности 0, јако је једноставно претворити координате у мрежу у координате у пикселима на екрану. Пошто су сви сегменти једнаке величине и њихова димензија је сачувана у константи **CELLSIZE**, једноставно ћемо на линијама 197 и 198, помножити координате сегмента тела змије у мрежи, константом **CELLSIZE** и тиме ћемо ове координате превести у координате у пикселима.

Сада је потребно креирати објекат који ћемо исцртати на координатама које смо израчунали. Како су сви сегменти једнаких димензија и квадратног су облика, јасно је да је потребно да креирамо квадрат на израчунатим координатама који ћемо потом исцртати. На линији 199, креирамо квадрат који ће бити исцртан као сегмент тела змије. Квадрат креирамо на израчунатим координатама, а његова димензија ће бити **CELLSIZE** будући да квадрат треба да покрије цео сегмент у мрежи. Потом, исцртавамо квадрат који смо креирали користећи функцију **pygame.draw.rect()** на линији 200. Дакле, исцртавамо тамно зелени квадрат преко сегмента који представља један део тела змије. Након тога, да би наша змија изгледала лепше, креирамо и мањи, светло зелени квадрат **преко** тамно зеленог који смо прво креирали тако да изгледа као да је наша змија састављена од сегмената светло зелене боје са тамно зеленим оквиром.

Унутрашњи светло зелени квадрат исцртавамо 4 пиксела на доле и 4 пиксела на десно од горњег левог темена ћелије у којој га цртамо. Ширина и висина овог квадрата су за 8 пиксела мањи од величине ћелије тако да ћемо добити маргину од 4 пиксела на десној и на доњој страни такође.

```

205. def drawApple(coord):
206.     x = coord['x'] * CELL_SIZE
207.     y = coord['y'] * CELL_SIZE
208.     appleRect = pygame.Rect(x, y, CELL_SIZE, CELL_SIZE)
209.     pygame.draw.rect(DISPLAYSURF, RED, appleRect)

```

Функција **drawApple()** служи за исцртавање циља (јабуке) на екрану. Она је јако слична функцији за исцртавање змије. У основи, раде исто, осим што у овом случају имамо само један квадрат који је отребно да исцртамо уместо листе као у првој функцији. Координате циља се прослеђују као параметар функцији и њен задатак је само да конвертује координате које је добила у координате у пикселима, линије 206 и 207. Све што је остало јесте да се креира квадрат на израчунатим координатама са димензијом **CELL_SIZE**, линија 208 и да се исцрта да екрану црвеном бојом, линија 209, наравно, користећи функцију **pygame.draw.rect()**.

```

212. def drawGrid():
213.     for x in range(0, WINDOWWIDTH, CELL_SIZE): # draw vertical lines
214.         pygame.draw.line(DISPLAYSURF, DARKGRAY, (x, 0), (x,
WINDOWHEIGHT))
215.     for y in range(0, WINDOWHEIGHT, CELL_SIZE): # draw horizontal
lines
216.         pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, y),
(WINDOWWIDTH, y))

```

Функција **drawGrid()** служи за исцртавање мреже на екрану. Мрежу исцртавамо само да би визуелно екран изгледао боље и да би олакшао игру играчу. Наравно, мрежа се састоји из линија, тако да је једноставно потребно исцртати их. Користићемо функцију **pygame.draw.line()**.

На слици изнад, представљен је код који исцртава мрежу употребом 2 **for** петље и то је најбољи начин за обављање овог посла. Касније ћемо објаснити тај код. Прво ћемо приказати код који би исцртао мрежу без употребе петљи, линију по линију како би било потпуно јасно шта у ствари радимо и да бисте лакше приметили шаблон по ком се мењају параметри. Дакле, како имамо 32 вертикалне линије, биће нам потребна 32 позива функције **pygame.draw.line()** са одговарајућим параметрима

```

pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 0), (0, WINDOWHEIGHT))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (20, 0), (20, WINDOWHEIGHT))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (40, 0), (40, WINDOWHEIGHT))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (60, 0), (60, WINDOWHEIGHT))
...skipped for brevity...
pygame.draw.line(DISPLAYSURF, DARKGRAY, (560, 0), (560, WINDOWHEIGHT))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (580, 0), (580, WINDOWHEIGHT))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (600, 0), (600, WINDOWHEIGHT))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (620, 0), (620, WINDOWHEIGHT))

```

Приметите шаблом по ком се мењају параметри. Дакле, потребне су нам 32 тачке са горње стране мреже и 32 тачке са доње стране мреже које ћемо спојити паралелним линијама. Присетите се да се функцији **pygame.draw.line()** прослеђују две тачке између којих се црта линија. Горње тачке почињу од тачке (0, 0), а завршавају се тачком (620, 0). Дакле, X координате ових тачака се мењају за по 20, почев од 0 (горњи леви угао) па све до 620 (горњи десни угао), док су Y координате увек 0 будући да су тачке које нам требају на самом врху екрана. Што се тиче тачака са доње стране, ситуација је иста као са тачкама са горње стране уз разлику у томе што је сада Y координата ових тачака фиксирана на вредност **WINDOWHEIGHT**, дакле на самом дну екрана. Свака од назначених тачака се спаја линијом са тачком наспрам себе и на тај начин се исцртавају паралелне вертикалне линије за мрежу. То значи да **for** петља треба да итерира почев од вредности 0, до вредности 640 са кораком 20. Тачније, наша петља треба да ради над опсегом **range(0, WINDOWHEIGHT, CELLSIZE)**.

За исцртавање хоризонталних линија, потребни су следећи позиви:

```
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 0), (WINDOWWIDTH, 0))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 20), (WINDOWWIDTH, 20))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 40), (WINDOWWIDTH, 40))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 60), (WINDOWWIDTH, 60))
...skipped for brevity...
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 400), (WINDOWWIDTH, 400))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 420), (WINDOWWIDTH, 420))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 440), (WINDOWWIDTH, 440))
pygame.draw.line(DISPLAYSURF, DARKGRAY, (0, 460), (WINDOWWIDTH, 460))
```

Ситуација је слична као и у исцртавању вертикалних линија. Овога пута, X координате обе тачке линије су фиксирани, прво на вредности 0, крајња лева ивица екрана, а потом на вредности константе **WINDOWWIDTH**, што представља X координату тачака на крајњој десној иници екрана. Вредности Y координате се мењају оба пута почев од 0, па сев до 460 са кораком 20. Наравно, и код исцртавања хоризонталних линија користимо **for** петљу која ће итерирати над опсегом **range(0, WINDOWHEIGHT, CELLSIZE)**.

Искористићемо ову прилику да нагласимо значај петљи у програмирању. Коришћење петљи је основна ствар у програмирању. Кодови су много читљивији и елегантнији и наравно, штеде нас непотребног куцања. Замислите да је неку командру потребно извршити 100 000 или више пута. Било би бесмислено искуцати исту команду толико пута, то би нас коштало много времена иако би у принципу, извршавање кода било исто као и без коришћења петљи.

```
219. if __name__ == '__main__':
220.     main()
```

Након имплементирања свих функција, дефинисања глобалних променљивих и константи, функција **main()** се позива како би се стартовала игра.

Немојте користити исте променљиве више пута

Погледајте још једном функцију за исцртавање змије **drawWorm()**.

```
199. wormSegmentRect = pygame.Rect(x, y, CELL_SIZE, CELL_SIZE)
200.     pygame.draw.rect(DISPLAYSURF, DARKGREEN, wormSegmentRect)
201.     wormInnerSegmentRect = pygame.Rect(x + 4, y + 4, CELL_SIZE -
      8, CELL_SIZE - 8)
202.     pygame.draw.rect(DISPLAYSURF, GREEN, wormInnerSegmentRect)
```

Приметите поново да смо креирали два различита објекта квадрата на линијама 199 и 201. Квадрат креиран на линији 199 смо сачували у локалној променљивој **wormSegmentRect** и проследили смо га функцији **pygame.draw.rect()** на линији 200 како бисмо га исцртали. Квадрат који смо креирали на линији 201 смо сачували у локалној променљивој **wormInnerSegmentRect** и проследили смо га функцији **pygame.draw.rect()** на линији 202 на исцртавање. Подсетимо се само да се ради о квадратима којима смо цртали тело змије, тамном већем и светлијем мањем.

Сваки пут када креирамо неку променљиву, у меморији рачунара се одвоји простор за ту променљиву. Можда сте у неком тренутку помислили да је паметније да смо за чување ова два објекта квадрата користили исту променљиву уместо две различите променљиве. На пример, можда вам се чинило да је паметније да смо променљиву **wormSegmentRect** искористили за оба објекта уместо што смо креирали нову променљиву **wormInnerSegmentRect** и тиме трошили више меморије. На слици испод је приказан код у ком користимо само једну променљиву за чување оба објекта који вам се можда чини бољим. Наравно, различите објекте бисмо чували у различито време. Није могуће да једна променљива има у исто време две различите вредности.

```
199. wormSegmentRect = pygame.Rect(x, y, CELL_SIZE, CELL_SIZE)
200.     pygame.draw.rect(DISPLAYSURF, DARKGREEN, wormSegmentRect)
201.     wormInnerSegmentRect = pygame.Rect(x + 4, y + 4, CELL_SIZE -
      8, CELL_SIZE - 8)
202.     pygame.draw.rect(DISPLAYSURF, GREEN, wormInnerSegmentRect)
```

Како први објекат који смо запамтили у променљивој на линији 199 неће више бити потребан после линије 200 на којој смо га исцртали, технички је потпуно исправно да ову променљиву сада искористимо за чување другог објекта који смо креирали на линији 201 без икаквих сметњи. Дакле, пошто сада користимо мање променљивих, чувамо меморију и наш програм је бољи. Да ли је ово тачно?

Технички, ово јесте тачно и коришћењем мањег броја променљивих стварно штедимо меморију и чинимо програм ефикаснијим по ресурсе рачунара. Међутим, да ли на овај начин негде губимо нешто?

Увођењем једне променљиве уместо две, меморија коју ћемо уштедети је свега неколико бајтова. На модерним рачунарима, таква уштеда би била потпуно не приметна. Са друге стране, коришћењем једне променљиве, знатно смањујемо читљивост нашег програма и ту је оно што губимо, а што је јако важно. Уколико би програмер после неког времена читао

овај код, видевши позиве на линијама 200 и 202, могао би да помисли да се ради о истом објекту или нешто слично томе и то би доста могло да отежа поновно читање и разумевање кода.

Мале ствари попут ове су јако важан фактор у програмирању. Јако често, ваш код ће читати други програмери и зато је јако важно да код буде што читљивији како би другима било лакше да разумеју како тачно ради. Уколико после неколико недеља погледате свој код, највероватније се нећете сећати шта како ради и биће као да читате код који је неко други писао. Зато је читљивост кода изузетно битна ствар у програмирању, далеко битнија од уштеде пар бајтова меморије.

Наравно, потребно је водити рачуна и о уштеди меморије у смислу избегавања непотребног разбацивања меморијског простора. Наиме, уколико није неопходно, не треба користити додатне променљиве и трошити меморију непотребно. Обично су то ситуације када се користе огромне листе или низови огромних дужина, поготову када су у питању велике структуре података. Тада треба избегавати њихово умножавање уколико није неопходно и слично. У тим ситуацијама, уштеда меморије може бити значајна и важно је обавити је на што бољи начин и у што већој мери.

Отклоните багове у програму

За додатну вежбу, уколико сте потпуно разумели сваки део програма ове игре, на линку испод можете скинути верзију игрице са уметнутим баговима. Ваш задатак је да пробате да отклоните багове и поправите игрицу тако да ради како треба. Наравно, прво морате да откријете саме багове.

<http://invpy.com/buggy/wormy>

Срећно и уживајте у програмирању!