

## Гурање звезда



### Како да играш Гурање звезда

Гурање звезда је „Sokoban“ или „Box Pusher“ клон. Играч се налази у соби са неколико звезда. Овде се налазе и циљеви са звездицама на које треба гурнути звезде. Играч мора да схвати како да звезде гурне на плочице које на себи имају ознаку звездице. Играч не може да гурне звезду ако је иза ње зид или нека друга звезда. Играч не може повући звезде, па ако звезда буде гурнута у угао, играч ће морати да рестартује ниво. Када су све звезде гуроте у плочице које су означене звездицом, ниво је завршен и прелази се на следећи.

Сваки ниво је сачињен од 2D мреже плочица. Плочице заправо представљају слике исте величине које се могу поставити једна поред друге да би се формирале сложеније слике. Са неколико подних и зидних плочица, можемо створити нивое многих занимљивих облика и величина.

Датотеке нивоа нису укључене у изворни код. Уместо тога, можете сами да направите датотеке нивоа или да их преузмете. Фајл са 201 нивоом се може преузети са <http://invpy.com/starPusherLevels.txt>. Када покренете програм, уверите се да је ова датотека нивоа у истом фолдеру као и датотека `starpusher.py`. У супротном, добићете ову поруку о грешци: `AssertionError: Cannot find the level file: starPusherLevels.txt`

Дизајн нивоа је првобитно направио Давид В. Скинер. Можете преузети више слагалица са његове веб странице на <http://users.bentonrea.com/~sasquatch/sokoban/>.

### Изворни код за Гурање звезда

Овај изворни код можете преузети са <http://invpy.com/starpusher.py>. Ако добијете било какве поруке о грешци, погледајте број линије који је наведен у поруци о грешци и проверите код за било коју грешку. Такође, можете да копирате и налепите свој код у веб форму на адреси <http://invpy.com/diff/starpusher> да видите да ли постоје разлике између вашег кода и кода у књизи.

Датотека нивоа се може преузети са <http://invpy.com/starPusherLevels.txt>. Плочице се могу преузети са <http://invpy.com/starPusherImages.zip>.

Такође, баш као и веверица, трава и непријатељски објекти у игри Веверица Једе Веверице, када кажем „мапни објекат“ , „објекти за игру“ , или „објекти нивоа“ у овом поглављу, не мислим на објекте у смислу објеката у Објектно Оријентисаном Програмирању. Ови „објекти“ су заправо само вредности у речнику, али их је лакше назвати објектима јер представљају ствари у свету игре.

```

1. # Star Pusher (a Sokoban clone)
2. # By Al Sweigart al@inventwithpython.com
3. # http://inventwithpython.com/pygame
4. # Creative Commons BY-NC-SA 3.0 US
5.
6. import random, sys, copy, os, pygame
7. from pygame.locals import *
8.
9. FPS = 30 # frames per second to update the screen
10. WINWIDTH = 800 # width of the program's window, in pixels
11. WINHEIGHT = 600 # height in pixels
12. HALF_WINWIDTH = int(WINWIDTH / 2)
13. HALF_WINHEIGHT = int(WINHEIGHT / 2)
14.
15. # The total width and height of each tile in pixels.
16. TILEWIDTH = 50
17. TILEHEIGHT = 85
18. TILEFLOORHEIGHT = 45
19.
20. CAM_MOVE_SPEED = 5 # how many pixels per frame the camera moves
21.
22. # The percentage of outdoor tiles that have additional
23. # decoration on them, such as a tree or rock.
24. OUTSIDE_DECORATION_PCT = 20
25.
26. BRIGHTBLUE = ( 0, 170, 255)
27. WHITE = (255, 255, 255)
28. BGCOLOR = BRIGHTBLUE
29. TEXTCOLOR = WHITE
30.
31. UP = 'up'
32. DOWN = 'down'
33. LEFT = 'left'
34. RIGHT = 'right'
35.
36.
37. def main():
38.     global FPSCLOCK, DISPLAYSURF, IMAGESDICT, TILEMAPPING,
        OUTSIDEDECOMAPPING, BASICFONT, PLAYERIMAGES, currentImage
39.
40.     # Pygame initialization and basic set up of the global variables.
41.     pygame.init()
42.     FPSCLOCK = pygame.time.Clock()
43.
44.     # Because the Surface object stored in DISPLAYSURF was returned
45.     # from the pygame.display.set_mode() function, this is the
46.     # Surface object that is drawn to the actual computer screen
47.     # when pygame.display.update() is called.
48.     DISPLAYSURF = pygame.display.set_mode((WINWIDTH, WINHEIGHT))
49.
50.     pygame.display.set_caption('Star Pusher')
51.     BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
52.
53.     # A global dict value that will contain all the Pygame
54.     # Surface objects returned by pygame.image.load().
55.     IMAGESDICT = {'uncovered goal': pygame.image.load('RedSelector.png'),
56.                   'covered goal': pygame.image.load('Selector.png'),
57.                   'star': pygame.image.load('Star.png'),
58.                   'corner': pygame.image.load('Wall Block Tall.png'),
59.                   'wall': pygame.image.load('Wood Block Tall.png'),
60.                   'inside floor': pygame.image.load('Plain Block.png'),

```

```

61.     'outside floor': pygame.image.load('Grass Block.png'),
62.     'title': pygame.image.load('star_title.png'),
63.     'solved': pygame.image.load('star_solved.png'),
64.     'princess': pygame.image.load('princess.png'),
65.     'boy': pygame.image.load('boy.png'),
66.     'catgirl': pygame.image.load('catgirl.png'),
67.     'horngirl': pygame.image.load('horngirl.png'),
68.     'pinkgirl': pygame.image.load('pinkgirl.png'),
69.     'rock': pygame.image.load('Rock.png'),
70.     'short tree': pygame.image.load('Tree_Short.png'),
71.     'tall tree': pygame.image.load('Tree_Tall.png'),
72.     'ugly tree': pygame.image.load('Tree_Ugly.png')}
73.
74. # These dict values are global, and map the character that appears
75. # in the level file to the Surface object it represents.
76. TILEMAPPING = {'x': IMAGESDICT['corner'],
77.                '#': IMAGESDICT['wall'],
78.                'o': IMAGESDICT['inside floor'],
79.                '.': IMAGESDICT['outside floor']}
80. OUTSIDEDECOMAPPING = {'1': IMAGESDICT['rock'],
81.                        '2': IMAGESDICT['short tree'],
82.                        '3': IMAGESDICT['tall tree'],
83.                        '4': IMAGESDICT['ugly tree']}
84.
85. # PLAYERIMAGES is a list of all possible characters the player can be.
86. # currentImage is the index of the player's current player image.
87. currentImage = 0
88. PLAYERIMAGES = [IMAGESDICT['princess'],
89.                  IMAGESDICT['boy'],
90.                  IMAGESDICT['catgirl'],
91.                  IMAGESDICT['horngirl'],
92.                  IMAGESDICT['pinkgirl']]
93.
94. startScreen() # show the title screen until the user presses a key
95.
96. # Read in the levels from the text file. See the readLevelsFile() for
97. # details on the format of this file and how to make your own levels.
98. levels = readLevelsFile('starPusherLevels.txt')
99. currentLevelIndex = 0
100.
101. # The main game loop. This loop runs a single level, when the user
102. # finishes that level, the next/previous level is loaded.
103. while True: # main game loop
104.     # Run the level to actually start playing the game:
105.     result = runLevel(levels, currentLevelIndex)
106.
107.     if result in ('solved', 'next'):
108.         # Go to the next level.
109.         currentLevelIndex += 1
110.         if currentLevelIndex >= len(levels):
111.             # If there are no more levels, go back to the first one.
112.             currentLevelIndex = 0
113.         elif result == 'back':
114.             # Go to the previous level.
115.             currentLevelIndex -= 1
116.             if currentLevelIndex < 0:
117.                 # If there are no previous levels, go to the last one.
118.                 currentLevelIndex = len(levels)-1
119.         elif result == 'reset':
120.             pass # Do nothing. Loop re-calls runLevel() to reset the level

```

```

121.
122.
123. def runLevel(levels, levelNum):
124.     global currentImage
125.     levelObj = levels[levelNum]
126.     mapObj = decorateMap(levelObj['mapObj'], levelObj['startState']['player'])
127.     gameStateObj = copy.deepcopy(levelObj['startState'])
128.     mapNeedsRedraw = True # set to True to call drawMap()
129.     levelSurf = BASICFONT.render('Level %s of %s' % (levelObj['levelNum'] + 1, totalNumOfLevels), 1, TEXTCOLOR)
130.     levelRect = levelSurf.get_rect()
131.     levelRect.bottomleft = (20, WINHEIGHT - 35)
132.     mapWidth = len(mapObj) * TILEWIDTH
133.     mapHeight = (len(mapObj[0]) - 1) * (TILEHEIGHT - TILEFLOORHEIGHT) + TILEHEIGHT
134.     MAX_CAM_X_PAN = abs(HALF_WINHEIGHT - int(mapHeight / 2)) + TILEWIDTH
135.     MAX_CAM_Y_PAN = abs(HALF_WINWIDTH - int(mapWidth / 2)) + TILEHEIGHT
136.
137.     levelsComplete = False
138.     # Track how much the camera has moved:
139.     cameraOffsetX = 0
140.     cameraOffsetY = 0
141.     # Track if the keys to move the camera are being held down:
142.     cameraUp = False
143.     cameraDown = False
144.     cameraLeft = False
145.     cameraRight = False
146.
147.     while True: # main game loop
148.         # Reset these variables:
149.         playerMoveTo = None
150.         keyPressed = False
151.
152.         for event in pygame.event.get(): # event handling loop
153.             if event.type == QUIT:
154.                 # Player clicked the "X" at the corner of the window.
155.                 terminate()
156.
157.             elif event.type == KEYDOWN:
158.                 # Handle key presses
159.                 keyPressed = True
160.                 if event.key == K_LEFT:
161.                     playerMoveTo = LEFT
162.                 elif event.key == K_RIGHT:
163.                     playerMoveTo = RIGHT
164.                 elif event.key == K_UP:
165.                     playerMoveTo = UP
166.                 elif event.key == K_DOWN:
167.                     playerMoveTo = DOWN
168.
169.                 # Set the camera move mode.
170.                 elif event.key == K_a:
171.                     cameraLeft = True
172.                 elif event.key == K_d:
173.                     cameraRight = True
174.                 elif event.key == K_w:
175.                     cameraUp = True
176.                 elif event.key == K_s:
177.                     cameraDown = True
178.

```

```

179.     elif event.key == K_n:
180.         return 'next'
181.     elif event.key == K_b:
182.         return 'back'
183.
184.     elif event.key == K_ESCAPE:
185.         terminate() # Esc key quits.
186.     elif event.key == K_BACKSPACE:
187.         return 'reset' # Reset the level.
188.     elif event.key == K_p:
189.         # Change the player image to the next one.
190.         currentImage += 1
191.         if currentImage >= len(PPLAYERIMAGES):
192.             # After the last player image, use the first
193.             currentImage = 0
194.         mapNeedsRedraw = True
195.
196.     elif event.type == KEYUP:
197.         # Unset the camera move mode.
198.         if event.key == K_a:
199.             cameraLeft = False
200.         elif event.key == K_d:
201.             cameraRight = False
202.         elif event.key == K_w:
203.             cameraUp = False
204.         elif event.key == K_s:
205.             cameraDown = False
206.
207.     if playerMoveTo != None and not levelsComplete:
208.         # If the player pushed a key to move, make the move
209.         # (if possible) and push any stars that are pushable.
210.         moved = makeMove(mapObj, gameStateObj, playerMoveTo)
211.
212.         if moved:
213.             # increment the step counter.
214.             gameStateObj['stepCounter'] += 1
215.             mapNeedsRedraw = True
216.
217.         if isLevelFinished(levelObj, gameStateObj):
218.             # level is solved, we should show the "Solved!" image.
219.             levelsComplete = True
220.             keyPressed = False
221.
222.     DISPLAYSURF.fill(BG_COLOR)
223.
224.     if mapNeedsRedraw:
225.         mapSurf = drawMap(mapObj, gameStateObj, levelObj['goals'])
226.         mapNeedsRedraw = False
227.
228.     if cameraUp and cameraOffsetY < MAX_CAM_X_PAN:
229.         cameraOffsetY += CAM_MOVE_SPEED
230.     elif cameraDown and cameraOffsetY > -MAX_CAM_X_PAN:
231.         cameraOffsetY -= CAM_MOVE_SPEED
232.     if cameraLeft and cameraOffsetX < MAX_CAM_Y_PAN:
233.         cameraOffsetX += CAM_MOVE_SPEED
234.     elif cameraRight and cameraOffsetX > -MAX_CAM_Y_PAN:
235.         cameraOffsetX -= CAM_MOVE_SPEED
236.
237.     # Adjust mapSurf's Rect object based on the camera offset.
238.     mapSurfRect = mapSurf.get_rect()
239.     mapSurfRect.center = (HALF_WINWIDTH + cameraOffsetX, HALF_WINHEIGHT + cameraOffsetY)

```

```

240.
241.     # Draw mapSurf to the DISPLAYSURF Surface object.
242.     DISPLAYSURF.blit(mapSurf, mapSurfRect)
243.
244.     DISPLAYSURF.blit(levelSurf, levelRect)
245.     stepSurf = BASICFONT.render('Steps: %s' % (gameStateObj['stepCounter']), 1, TEXTCOLOR)
246.     stepRect = stepSurf.get_rect()
247.     stepRect.bottomleft = (20, WINHEIGHT - 10)
248.     DISPLAYSURF.blit(stepSurf, stepRect)
249.
250.     if levelsComplete:
251.         # is solved, show the "Solved!" image until the player
252.         # has pressed a key.
253.         solvedRect = IMAGESDICT['solved'].get_rect()
254.         solvedRect.center = (HALF_WINWIDTH, HALF_WINHEIGHT)
255.         DISPLAYSURF.blit(IMAGESDICT['solved'], solvedRect)
256.
257.         if keyPressed:
258.             return 'solved'
259.
260.     pygame.display.update() # draw DISPLAYSURF to the screen.
261.     FPSLOCK.tick()
262.
263.
264. def isWall(mapObj, x, y):
265.     """Returns True if the (x, y) position on
266.     the map is a wall, otherwise return False."""
267.     if x < 0 or x >= len(mapObj) or y < 0 or y >= len(mapObj[x]):
268.         return False # x and y aren't actually on the map.
269.     elif mapObj[x][y] in ('#', 'x'):
270.         return True # wall is blocking
271.     return False
272.
273.
274. def decorateMap(mapObj, startxy):
275.     """Makes a copy of the given map object and modifies it.
276.     Here is what is done to it:
277.     * Walls that are corners are turned into corner pieces.
278.     * The outside/inside floor tile distinction is made.
279.     * Tree/rock decorations are randomly added to the outside tiles.
280.
281.     Returns the decorated map object."""
282.
283.     startx, starty = startxy # Syntactic sugar
284.
285.     # Copy the map object so we don't modify the original passed
286.     mapObjCopy = copy.deepcopy(mapObj)
287.
288.     # Remove the non-wall characters from the map data
289.     for x in range(len(mapObjCopy)):
290.         for y in range(len(mapObjCopy[0])):
291.             if mapObjCopy[x][y] in ('$', '!', '@', '+', '*'):
292.                 mapObjCopy[x][y] = ' '
293.
294.     # Flood fill to determine inside/outside floor tiles.
295.     floodFill(mapObjCopy, startx, starty, ' ', 'o')
296.
297.     # Convert the adjoined walls into corner tiles.
298.     for x in range(len(mapObjCopy)):
299.         for y in range(len(mapObjCopy[0])):
300.
301.             if mapObjCopy[x][y] == '#':

```

```

302.         if (isWall(mapObjCopy, x, y-1) and isWall(mapObjCopy, x+1, y)) or \
303.             (isWall(mapObjCopy, x+1, y) and isWall(mapObjCopy, x, y+1)) or \
304.             (isWall(mapObjCopy, x, y+1) and isWall(mapObjCopy, x-1, y)) or \
305.             (isWall(mapObjCopy, x-1, y) and isWall(mapObjCopy, x, y-1)):
306.             mapObjCopy[x][y] = 'x'
307.
308.         elif mapObjCopy[x][y] == ' ' and random.randint(0, 99) < OUTSIDE_DECORATION_PCT:
309.             mapObjCopy[x][y] = random.choice(list(OUTSIDEDECOMAPPING.keys()))
310.
311.     return mapObjCopy
312.
313.
314. def isBlocked(mapObj, gameStateObj, x, y):
315.     """Returns True if the (x, y) position on the map is
316.     blocked by a wall or star, otherwise return False."""
317.
318.     if isWall(mapObj, x, y):
319.         return True
320.
321.     elif x < 0 or x >= len(mapObj) or y < 0 or y >= len(mapObj[x]):
322.         return True # x and y aren't actually on the map.
323.
324.     elif (x, y) in gameStateObj['stars']:
325.         return True # a star is blocking
326.
327.     return False
328.
329.
330. def makeMove(mapObj, gameStateObj, playerMoveTo):
331.     """Given a map and game state object, see if it is possible for the
332.     player to make the given move. If it is, then change the player's
333.     position (and the position of any pushed star). If not, do nothing.
334.
335.     Returns True if the player moved, otherwise False."""
336.
337.     # Make sure the player can move in the direction they want.
338.     playerx, playery = gameStateObj['player']
339.
340.     # This variable is "syntactic sugar". Typing "stars" is more
341.     # readable than typing "gameStateObj['stars']" in our code.
342.     stars = gameStateObj['stars']
343.
344.     # The code for handling each of the directions is so similar aside
345.     # from adding or subtracting 1 to the x/y coordinates. We can
346.     # simplify it by using the xOffset and yOffset variables.
347.     if playerMoveTo == UP:
348.         xOffset = 0
349.         yOffset = -1
350.     elif playerMoveTo == RIGHT:
351.         xOffset = 1
352.         yOffset = 0
353.     elif playerMoveTo == DOWN:
354.         xOffset = 0
355.         yOffset = 1
356.     elif playerMoveTo == LEFT:
357.         xOffset = -1
358.         yOffset = 0
359.
360.     # See if the player can move in that direction.
361.     if isWall(mapObj, playerx + xOffset, playery + yOffset):
362.         return False
363.     else:

```

```

364.     if (playerx + xOffset, playery + yOffset) in stars:
365.         # There is a star in the way, see if the player can push it.
366.         if not isBlocked(mapObj, gameStateObj, playerx + (xOffset*2), playery + (yOffset*2)):
367.             # Move the star.
368.             ind = stars.index((playerx + xOffset, playery + yOffset))
369.             stars[ind] = (stars[ind][0] + xOffset, stars[ind][1] + yOffset)
370.         else:
371.             return False
372.         # Move the player upwards.
373.         gameStateObj['player'] = (playerx + xOffset, playery + yOffset)
374.         return True
375.
376.
377. def startScreen():
378.     """Display the start screen (which has the title and instructions)
379.     until the player presses a key. Returns None."""
380.
381.     # Position the title image.
382.     titleRect = IMAGESDICT['title'].get_rect()
383.     topCoord = 50 # topCoord tracks where to position the top of the text
384.     titleRect.top = topCoord
385.     titleRect.centerx = HALF_WINWIDTH
386.     topCoord += titleRect.height
387.
388.     # Unfortunately, Pygame's font & text system only shows one line at
389.     # a time, so we can't use strings with \n newline characters in them.
390.     # So we will use a list with each line in it.
391.     instructionText = ['Push the stars over the marks.',
392.                        'Arrow keys to move, WASD for camera control, P to change character.',
393.                        'Backspace to reset level, Esc to quit.',
394.                        'N for next level, B to go back a level.']
395.
396.     # Start with drawing a blank color to the entire window:
397.     DISPLAYSURF.fill(BG_COLOR)
398.
399.     # Draw the title image to the window:
400.     DISPLAYSURF.blit(IMAGESDICT['title'], titleRect)
401.
402.     # Position and draw the text.
403.     for i in range(len(instructionText)):
404.         instSurf = BASICFONT.render(instructionText[i], 1, TEXT_COLOR)
405.         instRect = instSurf.get_rect()
406.         topCoord += 10 # 10 pixels will go in between each line of text.
407.         instRect.top = topCoord
408.         instRect.centerx = HALF_WINWIDTH
409.         topCoord += instRect.height # Adjust for the height of the line.
410.         DISPLAYSURF.blit(instSurf, instRect)
411.
412.     while True: # Main loop for the start screen.
413.         for event in pygame.event.get():
414.             if event.type == QUIT:
415.                 terminate()
416.             elif event.type == KEYDOWN:
417.                 if event.key == K_ESCAPE:
418.                     terminate()
419.                 return # user has pressed a key, so return.
420.
421.         # Display the DISPLAYSURF contents to the actual screen.
422.         pygame.display.update()
423.         FPSCLOCK.tick()
424.

```



```

425.
426. def readLevelsFile(filename):
427.     assert os.path.exists(filename), 'Cannot find the level file: %s' % (filename)
428.     mapFile = open(filename, 'r')
429.     # Each level must end with a blank line
430.     content = mapFile.readlines() + ['\r\n']
431.     mapFile.close()
432.
433.     levels = [] # Will contain a list of level objects.
434.     levelNum = 0
435.     mapTextLines = [] # contains the lines for a single level's map.
436.     mapObj = [] # the map object made from the data in mapTextLines
437.     for lineNum in range(len(content)):
438.         # Process each line that was in the level file.
439.         line = content[lineNum].rstrip('\r\n')
440.
441.         if ';' in line:
442.             # Ignore the ; lines, they're comments in the level file.
443.             line = line[:line.find(';')]
444.
445.         if line != '':
446.             # This line is part of the map.
447.             mapTextLines.append(line)
448.         elif line == '' and len(mapTextLines) > 0:
449.             # A blank line indicates the end of a level's map in the file.
450.             # Convert the text in mapTextLines into a level object.
451.
452.             # Find the longest row in the map.
453.             maxWidth = -1
454.             for i in range(len(mapTextLines)):
455.                 if len(mapTextLines[i]) > maxWidth:
456.                     maxWidth = len(mapTextLines[i])
457.             # Add spaces to the ends of the shorter rows. This
458.             # ensures the map will be rectangular.
459.             for i in range(len(mapTextLines)):
460.                 mapTextLines[i] += ' ' * (maxWidth - len(mapTextLines[i]))
461.
462.             # Convert mapTextLines to a map object.
463.             for x in range(len(mapTextLines[0])):
464.                 mapObj.append([])
465.             for y in range(len(mapTextLines)):
466.                 for x in range(maxWidth):
467.                     mapObj[x].append(mapTextLines[y][x])
468.
469.             # Loop through the spaces in the map and find the @, ., and $
470.             # characters for the starting game state.
471.             startx = None # The x and y for the player's starting position
472.             starty = None
473.             goals = [] # list of (x, y) tuples for each goal.
474.             stars = [] # list of (x, y) for each star's starting position.
475.             for x in range(maxWidth):
476.                 for y in range(len(mapObj[x])):
477.                     if mapObj[x][y] in ('@', '+'):
478.                         # '@' is player, '+' is player & goal
479.                         startx = x
480.                         starty = y
481.                     if mapObj[x][y] in ('.', '+', '*'):
482.                         # '.' is goal, '*' is star & goal
483.                         goals.append((x, y))
484.                     if mapObj[x][y] in ('$ ', '*'):
485.                         # '$' is star

```

```

486.         stars.append((x, y))
487.
488.     # Basic level design sanity checks:
489.     assert startx != None and starty != None, 'Level %s (around
line %s) in %s is missing a "@" or "+" to mark the start point.' % (levelNum+1,
lineNum, filename)
490.     assert len(goals) > 0, 'Level %s (around line %s) in %s must
have at least one goal.' % (levelNum+1, lineNum, filename)

491.     assert len(stars) >= len(goals), 'Level %s (around line %s) in
%s is impossible to solve. It has %s goals but only %s stars.' % (levelNum+1,
lineNum, filename, len(goals), len(stars))
492.
493.     # Create level object and starting game state object.
494.     gameStateObj = {'player': (startx, starty),
495.                     'stepCounter': 0,
496.                     'stars': stars}
497.     levelObj = {'width': maxWidth,
498.                 'height': len(mapObj),
499.                 'mapObj': mapObj,
500.                 'goals': goals,
501.                 'startState': gameStateObj}
502.
503.     levels.append(levelObj)
504.
505.     # Reset the variables for reading the next map.
506.     mapTextLines = []
507.     mapObj = []
508.     gameStateObj = {}
509.     levelNum += 1
510.     return levels
511.
512.
513. def floodFill(mapObj, x, y, oldCharacter, newCharacter):
514.     """Changes any values matching oldCharacter on the map object to
515.     newCharacter at the (x, y) position, and does the same for the
516.     positions to the left, right, down, and up of (x, y), recursively."""
517.
518.     # In this game, the flood fill algorithm creates the inside/outside
519.     # floor distinction. This is a "recursive" function.
520.     # For more info on the Flood Fill algorithm, see:
521.     # http://en.wikipedia.org/wiki/Flood\_fill
522.     if mapObj[x][y] == oldCharacter:
523.         mapObj[x][y] = newCharacter
524.
525.     if x < len(mapObj) - 1 and mapObj[x+1][y] == oldCharacter:
526.         floodFill(mapObj, x+1, y, oldCharacter, newCharacter) # call right
527.     if x > 0 and mapObj[x-1][y] == oldCharacter:
528.         floodFill(mapObj, x-1, y, oldCharacter, newCharacter) # call left
529.     if y < len(mapObj[x]) - 1 and mapObj[x][y+1] == oldCharacter:
530.         floodFill(mapObj, x, y+1, oldCharacter, newCharacter) # call down
531.     if y > 0 and mapObj[x][y-1] == oldCharacter:
532.         floodFill(mapObj, x, y-1, oldCharacter, newCharacter) # call up
533.
534.
535. def drawMap(mapObj, gameStateObj, goals):
536.     """Draws the map to a Surface object, including the player and
537.     stars. This function does not call pygame.display.update(), nor
538.     does it draw the "Level" and "Steps" text in the corner."""
539.
540.     # mapSurf will be the single Surface object that the tiles are drawn
541.     # on, so that it is easy to position the entire map on the DISPLAYSURF

```

```

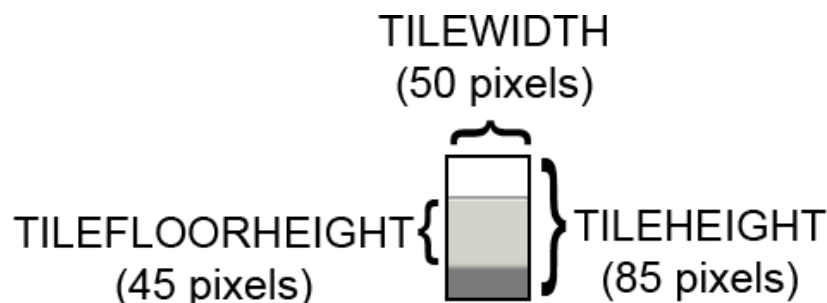
542. # Surface object. First, the width and height must be calculated.
543. mapSurfWidth = len(mapObj) * TILEWIDTH
544. mapSurfHeight = (len(mapObj[0]) - 1) * (TILEHEIGHT - TILEFLOORHEIGHT) + TILEHEIGHT
545. mapSurf = pygame.Surface((mapSurfWidth, mapSurfHeight))
546. mapSurf.fill(BG_COLOR) # start with a blank color on the surface.
547.
548. # Draw the tile sprites onto this surface.
549. for x in range(len(mapObj)):
550.     for y in range(len(mapObj[x])):
551.         spaceRect = pygame.Rect((x * TILEWIDTH, y * (TILEHEIGHT -
TILEFLOORHEIGHT), TILEWIDTH, TILEHEIGHT))
552.         if mapObj[x][y] in TILEMAPPING:
553.             baseTile = TILEMAPPING[mapObj[x][y]]
554.         elif mapObj[x][y] in OUTSIDEDECOMAPPING:
555.             baseTile = TILEMAPPING[' ']
556.
557.         # First draw the base ground/wall tile.
558.         mapSurf.blit(baseTile, spaceRect)
559.
560.         if mapObj[x][y] in OUTSIDEDECOMAPPING:
561.             # Draw any tree/rock decorations that are on this tile.
562.             mapSurf.blit(OUTSIDEDECOMAPPING[mapObj[x][y]], spaceRect)
563.         elif (x, y) in gameStateObj['stars']:
564.             if (x, y) in goals:
565.                 # A goal AND star are on this space, draw goal first.
566.                 mapSurf.blit(IMAGESDICT['covered goal'], spaceRect)
567.                 # Then draw the star sprite.
568.                 mapSurf.blit(IMAGESDICT['star'], spaceRect)
569.             elif (x, y) in goals:
570.                 # Draw a goal without a star on it.
571.                 mapSurf.blit(IMAGESDICT['uncovered goal'], spaceRect)
572.
573.         # Last draw the player on the board.
574.         if (x, y) == gameStateObj['player']:
575.             # Note: The value "currentImage" refers
576.             # to a key in "PLAYERIMAGES" which has the
577.             # specific player image we want to show.
578.             mapSurf.blit(PLAYERIMAGES[currentImage], spaceRect)
579.
580.     return mapSurf
581.
582.
583. def isLevelFinished(levelObj, gameStateObj):
584.     """Returns True if all the goals have stars in them."""
585.     for goal in levelObj['goals']:
586.         if goal not in gameStateObj['stars']:
587.             # Found a space with a goal but no star on it.
588.             return False
589.     return True
590.
591.
592. def terminate():
593.     pygame.quit()
594.     sys.exit()
595.
596.
597. if name == 'main':
598.     main()

```

## Почетно подешавање

```
1. # Star Pusher (a Sokoban clone)
2. # By Al Sweigart al@inventwithpython.com
3. # http://inventwithpython.com/pygame
4. # Creative Commons BY-NC-SA 3.0 US
5.
6. import random, sys, copy, os, pygame
7. from pygame.locals import *
8.
9. FPS = 30 # frames per second to update the screen
10. WINWIDTH = 800 # width of the program's window, in pixels
11. WINHEIGHT = 600 # height in pixels
12. HALF_WINWIDTH = int(WINWIDTH / 2)
13. HALF_WINHEIGHT = int(WINHEIGHT / 2)
14.
15. # The total width and height of each tile in pixels.
16. TILEWIDTH = 50
17. TILEHEIGHT = 85
18. TILEFLOORHEIGHT = 45
19.
20. CAM_MOVE_SPEED = 5 # how many pixels per frame the camera moves
21.
22. # The percentage of outdoor tiles that have additional
23. # decoration on them, such as a tree or rock.
24. OUTSIDE_DECORATION_PCT = 20
25.
26. BRIGHTBLUE = ( 0, 170, 255)
27. WHITE = (255, 255, 255)
28. BGCOLOR = BRIGHTBLUE
29. TEXTCOLOR = WHITE
30.
31. UP = 'up'
32. DOWN = 'down'
33. LEFT = 'left'
34. RIGHT = 'right'
```

Ове константе се користе у различитим деловима програма. Променљиве `TILEWIDTH` и `TILEHEIGHT` показују да је свака од слика у ширини 50 пиксела и висока 85 пиксела. Међутим, ове плочице се преклапају једна са другом када су нацртане на екрану. (Ово је објашњено касније.) `TILEFLOORHEIGHT` се односи на чињеницу да је део плочице који представља под има висину 45 пиксела. Ево дијаграма слике равног пода:



Травнате плочице изван просторије ће понекад имати додатне декорације, као што су дрвеће или камење. Константа `OUTSIDE_DECORATION_PCT` показује који проценат ових плочица ће насумично имати ове декорације.

```

37. def main():
38.     global FPSLOCK, DISPLAYSURF, IMAGESDICT, TILEMAPPING,
        OUTSIDEDECOMAPPING, BASICFONT, PLAYERIMAGES, currentImage
39.
40.     # Pygame initialization and basic set up of the global variables.
41.     pygame.init()
42.     FPSLOCK = pygame.time.Clock()
43.
44.     # Because the Surface object stored in DISPLAYSURF was returned
45.     # from the pygame.display.set_mode() function, this is the
46.     # Surface object that is drawn to the actual computer screen
47.     # when pygame.display.update() is called.
48.     DISPLAYSURF = pygame.display.set_mode((WINWIDTH, WINHEIGHT))
49.
50.     pygame.display.set_caption('Star Pusher')
51.     BASICFONT = pygame.font.Font('freesansbold.ttf', 18)

```

Ово су уобичајена Pygame подешавања која се извршавају на почетку програма.

```

53. # A global dict value that will contain all the Pygame
54. # Surface objects returned by pygame.image.load().
55. IMAGESDICT = {'uncovered goal': pygame.image.load('RedSelector.png'),
56.               'covered goal': pygame.image.load('Selector.png'),
57.               'star': pygame.image.load('Star.png'),
58.               'corner': pygame.image.load('Wall Block Tall.png'),
59.               'wall': pygame.image.load('Wood Block Tall.png'),
60.               'inside floor': pygame.image.load('Plain Block.png'),
61.               'outside floor': pygame.image.load('Grass Block.png'),
62.               'title': pygame.image.load('star_title.png'),
63.               'solved': pygame.image.load('star_solved.png'),
64.               'princess': pygame.image.load('princess.png'),
65.               'boy': pygame.image.load('boy.png'),
66.               'catgirl': pygame.image.load('catgirl.png'),
67.               'horngirl': pygame.image.load('horngirl.png'),
68.               'pinkgirl': pygame.image.load('pinkgirl.png'),
69.               'rock': pygame.image.load('Rock.png'),
70.               'short tree': pygame.image.load('Tree_Short.png'),
71.               'tall tree': pygame.image.load('Tree_Tall.png'),
72.               'ugly tree': pygame.image.load('Tree_Ugly.png')}

```

IMAGESDICT је речник где се чувају учитане слике. Ово олакшава употребу у другим функцијама, јер само IMAGESDICT променљива мора бити глобална. Ако смо сваку од ових слика сачували у одвојеним променљивама, онда би свих 18 променљивих (за 18 слика које се користе у овој игри) морало бити глобално. Речник који садржи све површинске објекте са сликама је лакши за руковање.

```

74. # These dict values are global, and map the character that appears
75. # in the level file to the Surface object it represents.
76. TILEMAPPING = {'x': IMAGESDICT['corner'],
77.                '#': IMAGESDICT['wall'],
78.                'o': IMAGESDICT['inside floor'],
79.                ' ': IMAGESDICT['outside floor']}

```

Структура података за мапу је само 2D листа појединачних низова знакова. Речник TILEMAPPING повезује знакове који се користе са сликама које представљају. (Ово ће постати јасније у објашњењу функције drawMap()).

```
80. OUTSIDEDECOMAPPING = {'1': IMAGESDICT['rock'],
81.                        '2': IMAGESDICT['short tree'],
82.                        '3': IMAGESDICT['tall tree'],
83.                        '4': IMAGESDICT['ugly tree']}
```

OUTSIDEDECOMAPPING је такође речник који повезује знакове који се користе са сликама које су уčitане. Сlike „спољашње декорације“ су нацртане на врху травнатих плочица.

```
85. # PLAYERIMAGES is a list of all possible characters the player can be.
86. # currentImage is the index of the player's current player image.
87. currentImage = 0
88. PLAYERIMAGES = [IMAGESDICT['princess'],
89.                 IMAGESDICT['boy'],
90.                 IMAGESDICT['catgirl'],
91.                 IMAGESDICT['horngirl'],
92.                 IMAGESDICT['pinkgirl']]
```

Листа PLAYERIMAGES чува слике које се користе за играча. currentImage променљива прати индекс тренутно одабране слике играча. На пример, када је currentImage подешен на 0, онда се PLAYERIMAGES[0], која је „принцеза“ слика играча приказује на екрану.

```
94. startScreen() # show the title screen until the user presses a key
95.
96. # Read in the levels from the text file. See the readLevelsFile() for
97. # details on the format of this file and how to make your own levels.
98. levels = readLevelsFile('starPusherLevels.txt')
99. currentLevelIndex = 0
```

Функција startScreen() ће наставити да приказује почетни екран (који такође има упутства за игру) док играч не притисне тастер. Када играч притисне тастер, функција startScreen() се враћа, а затим чита нивое из фајла. Играч креће на првом нивоу, што је објекат у листи нивоа са индексом 0.

```
101. # The main game loop. This loop runs a single level, when the user
102. # finishes that level, the next/previous level is loaded.
103. while True: # main game loop
104.     # Run the level to actually start playing the game:
105.     result = runLevel(levels, currentLevelIndex)
```

Функција runLevel() обрађује све акције за игру. Када играч заврши са играњем нивоа, runLevel() ће вратити један од следећих низова: 'solved' (јер је играч завршио стављање свих звезда на циљеве), 'next' (јер играч жели да пређе на наредни ниво), 'back' (јер играч жели да се врати на претходни ниво), и 'reset' (јер играч жели да поново почне да игра тренутни ниво, можда зато што је гурнуо звезду у ћошак).

```

107. if result in ('solved', 'next'):
108.     # Go to the next level.
109.     currentLevelIndex += 1
110.     if currentLevelIndex >= len(levels):
111.         # If there are no more levels, go back to the first one.
112.         currentLevelIndex = 0
113. elif result == 'back':
114.     # Go to the previous level.
115.     currentLevelIndex -= 1
116.     if currentLevelIndex < 0:
117.         # If there are no previous levels, go to the last one.
118.         currentLevelIndex = len(levels)-1

```

Ако функција runLevel() врати стрингове 'solved' или 'next', онда морамо повећати levelNum за 1. Ако се levelNum повећа преко броја нивоа, онда се levelNum поставља насад на 0.

Ако се врати 'back', онда се levelNum смањује за 1. Ако levelNum има вредност мању од 0, онда се поставља на задњи ниво (који је len(levels)-1).

```

119. elif result == 'reset':
120.     pass # Do nothing. Loop re-calls runLevel() to reset the level

```

Ако је повратна вредност 'reset', онда код не ради ништа. Ставка pass не ради ништа (као коментар), али је потребна зато што Python интерпретер очекује увучену линију кода након elif исказа.

Можемо потпуно уклонити линије 119 и 120 из изворног кода, а програм ће и даље радити исто. Разлог због којег их овде укључујемо је читљивост кода, тако да ако касније правимо измене, нећемо заборавити да runLevel() такође може да врати стринг 'reset'.

```

123. def runLevel(levels, levelNum):
124.     global currentImage
125.     levelObj = levels[levelNum]
126.     mapObj = decorateMap(levelObj['mapObj'], levelObj['startState']['player'])
127.     gameStateObj = copy.deepcopy(levelObj['startState'])

```

Листа нивоа садржи све објекте нивоа који су учитани из датотеке нивоа. Објекат нивоа за тренутн ниво (што је онај који је сетован у levelNum) је смештен у променљивој levelObj. Објекат мапе (који прави разлику између унутарњих и спољњих плочица, и декорације плочица) враћа се из функције decorateMap(). Да би пратили стање игре док играч игра овај ниво, копија објекта стања игре која је смештена у levelObj је направљена помоћу функције copy.deepcopy().

Копија објекта стања игре је направљена зато што објекат стања игре сачуван у levelObj['startState'] представља стање игре на самом почетку нивоа, а ми не желимо да га модификујемо. У супротном, ако играч поново покрене ниво, оригинално стање игре за тај ниво ће бити изгубљено.

Функција copy.deepcopy() се користи зато што је објекат стања игре речник који има торке. Али технички, речник садржи референце на торке. (Референце су детаљно објашњене на <http://inypy.com/references>.) Користећи израз за доделу да направите копију референци, али не и вредности на које се односе, тако да се и копија и оригинални речник идаље односе на исте „тупле“.

Функција copy.deepcopy() решава овај проблем тако што прави копије стварних туплова у речнику. На овај начин можемо гарантовати да промена једног речника неће утицати на други речник.

```

128. mapNeedsRedraw = True # set to True to call drawMap()
129. levelSurf = BASICFONT.render('Level %s of %s' % (levelObj['levelNum'] + 1, totalNumOfLevels), 1, TEXTCOLOR)
130. levelRect = levelSurf.get_rect()
131. levelRect.bottomleft = (20, WINHEIGHT - 35)
132. mapWidth = len(mapObj) * TILEWIDTH
133. mapHeight = (len(mapObj[0]) - 1) * (TILEHEIGHT - TILEFLOORHEIGHT) + TILEHEIGHT
134. MAX_CAM_X_PAN = abs(HALF_WINHEIGHT - int(mapHeight / 2)) + TILEWIDTH
135. MAX_CAM_Y_PAN = abs(HALF_WINWIDTH - int(mapWidth / 2)) + TILEHEIGHT
136.
137. levelsComplete = False
138. # Track how much the camera has moved:
139. cameraOffsetX = 0
140. cameraOffsetY = 0
141. # Track if the keys to move the camera are being held down:
142. cameraUp = False
143. cameraDown = False
144. cameraLeft = False
145. cameraRight = False

```

Више варијабли је постављено на почетку играња нивоа. `mapWidth` и `mapHeight` променљиве представљају величину мапа у пикселима. Израз за израчунавање `mapHeight` је помало компликован јер се плочице међусобно преклапају. Само је доњи ред плочица пуна висина (која представља део израза + `TILEHEIGHT`), сви остали редови плочица (чији је број `(len(mapObj[0]) - 1)`) мало преклопљен. То значи да су ефективно сваки `(TILEHEIGHT - TILEFLOORHEIGHT)` високи пиксели.

Камера у апликацији се може померати независно од кретања играча по мапи. Због тога је камери потребан сопствени скуп „помичних“ променљивих: `cameraUp`, `cameraDown`, `cameraLeft`, и `cameraRight`. Променљиве `cameraOffsetX` и `cameraOffsetY` прате положај камере.

```

147. while True: # main game loop
148.     # Reset these variables:
149.     playerMoveTo = None
150.     keyPressed = False
151.
152.     for event in pygame.event.get(): # event handling loop
153.         if event.type == QUIT:
154.             # Player clicked the "X" at the corner of the window.
155.             terminate()
156.

```

Променљива `playerMoveTo` ће бити серована на константу правца у ком играч намерава да се премести на мапи. `keyPressed` променљива прати да ли је било који тастер притиснут током ове итерације петље. Ова променљива се проверава касније када играч реши ниво.

```

157.     elif event.type == KEYDOWN:
158.         # Handle key presses
159.         keyPressed = True
160.         if event.key == K_LEFT:
161.             playerMoveTo = LEFT
162.         elif event.key == K_RIGHT:
163.             playerMoveTo = RIGHT
164.         elif event.key == K_UP:
165.             playerMoveTo = UP
166.         elif event.key == K_DOWN:
167.             playerMoveTo = DOWN
168.

```



```

169.     # Set the camera move mode.
170.     elif event.key == K_a:
171.         cameraLeft = True
172.     elif event.key == K_d:
173.         cameraRight = True
174.     elif event.key == K_w:
175.         cameraUp = True
176.     elif event.key == K_s:
177.         cameraDown = True
178.
179.     elif event.key == K_n:
180.         return 'next'
181.     elif event.key == K_b:
182.         return 'back'
183.
184.     elif event.key == K_ESCAPE:
185.         terminate() # Esc key quits.
186.     elif event.key == K_BACKSPACE:
187.         return 'reset' # Reset the level.
188.     elif event.key == K_p:
189.         # Change the player image to the next one.
190.         currentImage += 1
191.         if currentImage >= len(PLAYERIMAGES):
192.             # After the last player image, use the first
193.             currentImage = 0
194.         mapNeedsRedraw = True
195.
196. elif event.type == KEYUP:
197.     # Unset the camera move mode.
198.     if event.key == K_a:
199.         cameraLeft = False
200.     elif event.key == K_d:
201.         cameraRight = False
202.     elif event.key == K_w:
203.         cameraUp = False
204.     elif event.key == K_s:
205.         cameraDown = False

```

Овај код одређује шта ће се догодити у односу на притиснут тастер.

```

207. if playerMoveTo != None and not levelsComplete:
208.     # If the player pushed a key to move, make the move
209.     # (if possible) and push any stars that are pushable.
210.     moved = makeMove(mapObj, gameStateObj, playerMoveTo)
211.
212.     if moved:
213.         # increment the step counter.
214.         gameStateObj['stepCounter'] += 1
215.         mapNeedsRedraw = True
216.
217.     if isLevelFinished(levelObj, gameStateObj):
218.         # level is solved, we should show the "Solved!" image.
219.         levelsComplete = True
220.         keyPressed = False

```

Ако променљива `playerMoveTo` више није постављена на `None`, онда знамо да играч намерава да се помери. Позив функције `makeMove()` управља променом XY координата позиције играча у `gameStateObj`, као и померањем било које звезде. Повратна вредност функције `makeMove()` се складишти у `moved`. Ако је ова вредност `True`, онда је играч померен у том правцу. Ако је вредност била `False`, онда је играч покушао да се помери на плочицу која је била зид, или да

гурне звезду која има неку препреку иза себе. У овом случају, играч не може да се помера и на мапи се ништа не мења.

```
222. DISPLAYSURF.fill(BG_COLOR)
223.
224. if mapNeedsRedraw:
225.     mapSurf = drawMap(mapObj, gameStateObj, levelObj['goals'])
226.     mapNeedsRedraw = False
```

Мапа не треба поново да се исцртава на свакој итерацији кроз петљу игре. У ствари, овај програм је довољно компликован да би то зазвало лагано (али приметно) успоравање у игри. Мапа треба да се поново нацрта када се нешто промени (као што је играч који се креће или звезда која се гурла). Тако је Surface објекат у mapSurf променљивој ажуриран само позивом drawMap() функције када је mapNeedsRedraw променљива постављена на True.

Након што је мапа нацртана на линији 225, променљива mapNeedsRedraw је постављена на False. Ако желите да видите како програм успорава цртање на свакој итерацији кроз петљу, ставите линију 226 под коментар и поново покрените програм. Приметићете да је померање камере значајно спорије.

```
228. if cameraUp and cameraOffsetY < MAX_CAM_X_PAN:
229.     cameraOffsetY += CAM_MOVE_SPEED
230. elif cameraDown and cameraOffsetY > -MAX_CAM_X_PAN:
231.     cameraOffsetY -= CAM_MOVE_SPEED
232. if cameraLeft and cameraOffsetX < MAX_CAM_Y_PAN:
233.     cameraOffsetX += CAM_MOVE_SPEED
234. elif cameraRight and cameraOffsetX > -MAX_CAM_Y_PAN:
235.     cameraOffsetX -= CAM_MOVE_SPEED
```

Ако су променљиве за кретање камере постављене на True и камера није прошла границе постављене са MAX\_CAM\_X\_PAN и MAX\_CAM\_Y\_PAN, локација камере (сачувана у cameraOffsetX и cameraOffsetY) би се требала помакнути за CAM\_MOVE\_SPEED пиксела.

Имајте на уму да постоје наредбе if и elif на линијама 228 и 230 за померање камере горе и доле, а затим одвојене if и elif наредбе на линијама 232 и 234. На овај начин корисник може померити камеру вертикално и хоризонтално у истом тренутку. То не би било могуће да је линија 232 била elif наредба.

```
237. # Adjust mapSurf's Rect object based on the camera offset.
238. mapSurfRect = mapSurf.get_rect()
239. mapSurfRect.center = (HALF_WINWIDTH + cameraOffsetX, HALF_WINHEIGHT + cameraOffsetY)
240.
241. # Draw mapSurf to the DISPLAYSURF Surface object.
242. DISPLAYSURF.blit(mapSurf, mapSurfRect)
243.
244. DISPLAYSURF.blit(levelSurf, levelRect)
245. stepSurf = BASICFONT.render('Steps: %s' % (gameStateObj['stepCounter']), 1, TEXT_COLOR)
246. stepRect = stepSurf.get_rect()
247. stepRect.bottomleft = (20, WINHEIGHT - 10)
248. DISPLAYSURF.blit(stepSurf, stepRect)
249.
250. if levelsComplete:
251.     # is solved, show the "Solved!" image until the player
252.     # has pressed a key.
253.     solvedRect = IMAGESDICT['solved'].get_rect()
254.     solvedRect.center = (HALF_WINWIDTH, HALF_WINHEIGHT)
255.     DISPLAYSURF.blit(IMAGESDICT['solved'], solvedRect)
256.
```

```

257.     if keyPressed:
258.         return 'solved'
259.
260.     pygame.display.update() # draw DISPLAYSURF to the screen.
261.     FPSLOCK.tick()
262.
263.

```

Линије од 237 до 261 позиционирају камеру и цртају мапу и друге графике на приказану површину објекта у DISPLAYSURF. Ако је ниво решен, онда је графика победе такође нацртана изнад свега осталог. Променљива keyPressed ће бити постављена на True ако је корисник притиснуо тастер током ове итерације, када ће се извршити функција runLevel().

```

264. def isWall(mapObj, x, y):
265.     """Returns True if the (x, y) position on
266.     the map is a wall, otherwise return False."""
267.     if x < 0 or x >= len(mapObj) or y < 0 or y >= len(mapObj[x]):
268.         return False # x and y aren't actually on the map.
269.     elif mapObj[x][y] in ('#', 'x'):
270.         return True # wall is blocking
271.     return False

```

Функција isWall() враћа True ако постоји зид на мапи на XY координатама које се прослеђују функцији. Зидни објекти су представљени као 'x' или '#' стрингови у мапи објекта.

```

274. def decorateMap(mapObj, startxy):
275.     """Makes a copy of the given map object and modifies it.
276.     Here is what is done to it:
277.     * Walls that are corners are turned into corner pieces.
278.     * The outside/inside floor tile distinction is made.
279.     * Tree/rock decorations are randomly added to the outside tiles.
280.
281.     Returns the decorated map object."""
282.
283.     startx, starty = startxy # Syntactic sugar
284.
285.     # Copy the map object so we don't modify the original passed
286.     mapObjCopy = copy.deepcopy(mapObj)

```

Функција decorateMap() мења mapObj структуру података тако да није тако јасна као што се појављује у датотеци мапе. Три ствари које decorateMa() мења су објашњене у коментару на врху функције.

```

288.     # Remove the non-wall characters from the map data
289.     for x in range(len(mapObjCopy)):
290.         for y in range(len(mapObjCopy[0])):
291.             if mapObjCopy[x][y] in ('$', '.', '@', '+', '*'):
292.                 mapObjCopy[x][y] = ' '

```

Објекат мапе има знакове који представљају положај играча, циљеве и звезде. Они су неопходни за објекат мапе (они се чувају у другим структурама података након што је датотека са мапом прочитана) тако да е конвертују у празнине.

```

294.     # Flood fill to determine inside/outside floor tiles.
295.     floodFill(mapObjCopy, startx, starty, ' ', 'o')

```

Функција floodFill() ће променити све плочице унутар зидова са знаковима ' ' на знакове 'o'. То се ради помоћу концепта програмирања званог рекурзија, који је објашњен у одељку „рекурзивне функције“ у овом поглављу.

```

297. # Convert the adjoined walls into corner tiles.
298. for x in range(len(mapObjCopy)):
299.     for y in range(len(mapObjCopy[0])):
300.
301.         if mapObjCopy[x][y] == '#':
302.             if (isWall(mapObjCopy, x, y-1) and isWall(mapObjCopy, x+1, y)) or \
303.                 (isWall(mapObjCopy, x+1, y) and isWall(mapObjCopy, x, y+1)) or \
304.                 (isWall(mapObjCopy, x, y+1) and isWall(mapObjCopy, x-1, y)) or \
305.                 (isWall(mapObjCopy, x-1, y) and isWall(mapObjCopy, x, y-1)):
306.                 mapObjCopy[x][y] = 'x'
307.
308.         elif mapObjCopy[x][y] == ' ' and random.randint(0, 99) < OUTSIDE_DECORATION_PCT:
309.             mapObjCopy[x][y] = random.choice(list(OUTSIDEDECOMAPPING.keys()))
310.
311.     return mapObjCopy

```

Велики, вишелинијски израз на линији 301 проверава да ли је зидна плочица на тренутним XY координатама угаона зидна плочица проверавајући да ли постоје зидне плочице поред ње које чине облик угла. Ако је тако, стринг '#' у објекту мапе који представља нормалан тид је промењен у 'x' стринг који представља угаону зидну плочицу.

```

314. def isBlocked(mapObj, gameStateObj, x, y):
315.     """Returns True if the (x, y) position on the map is
316.     blocked by a wall or star, otherwise return False."""
317.
318.     if isWall(mapObj, x, y):
319.         return True
320.
321.     elif x < 0 or x >= len(mapObj) or y < 0 or y >= len(mapObj[x]):
322.         return True # x and y aren't actually on the map.
323.
324.     elif (x, y) in gameStateObj['stars']:
325.         return True # a star is blocking
326.
327.     return False

```

Постоје три случаја у којима би простор на мапи био блокиран: ако постоји звезда, зид или су координате простора прошле ивице мапе. Функција isBlocked() проверава ова три случаја и враћа True ако су XY координате блокиране, а False ако није.

```

330. def makeMove(mapObj, gameStateObj, playerMoveTo):
331.     """Given a map and game state object, see if it is possible for the
332.     player to make the given move. If it is, then change the player's
333.     position (and the position of any pushed star). If not, do nothing.
334.
335.     Returns True if the player moved, otherwise False."""
336.
337.     # Make sure the player can move in the direction they want.
338.     playerx, playery = gameStateObj['player']
339.
340.     # This variable is "syntactic sugar". Typing "stars" is more
341.     # readable than typing "gameStateObj['stars']" in our code.
342.     stars = gameStateObj['stars']
343.
344.     # The code for handling each of the directions is so similar aside
345.     # from adding or subtracting 1 to the x/y coordinates. We can
346.     # simplify it by using the xOffset and yOffset variables.
347.     if playerMoveTo == UP:

```

```

348.     xOffset = 0
349.     yOffset = -1
350. elif playerMoveTo == RIGHT:
351.     xOffset = 1
352.     yOffset = 0
353. elif playerMoveTo == DOWN:
354.     xOffset = 0
355.     yOffset = 1
356. elif playerMoveTo == LEFT:
357.     xOffset = -1
358.     yOffset = 0
359.
360. # See if the player can move in that direction.
361. if isWall(mapObj, playerx + xOffset, playery + yOffset):
362.     return False
363. else:
364.     if (playerx + xOffset, playery + yOffset) in stars:
365.         # There is a star in the way, see if the player can push it.
366.         if not isBlocked(mapObj, gameStateObj, playerx + (xOffset*2), playery + (yOffset*2)):
367.             # Move the star.
368.             ind = stars.index((playerx + xOffset, playery + yOffset))
369.             stars[ind] = (stars[ind][0] + xOffset, stars[ind][1] + yOffset)
370.         else:
371.             return False
372.     # Move the player upwards.
373.     gameStateObj['player'] = (playerx + xOffset, playery + yOffset)
374.     return True

```

Функција `makeMove()` проверава да ли је померање играча у одређеном смеру исправан потез. Све док не постоји зид који блокира путању, или звезда која има зид или другу звезду иза ње, играч ће моћи да се креће у том правцу. Променљива `gameStateObj` ће бити ажурирана како би одражавала ово, и вредост `True` ће бити враћена да каже позиваоцу функције да је играч премештен.

Ако је на плочици на коју је играч желео да се помери постојала звезда, положај те звезде се такође мења и ова информација се ажурира у променљивој `gameStateObj`. На овај начин функционише гурање звезде.

Ако је играч блокиран да се креће у жељеном правцу, онда `gameStateObj` није модификован и функција враћа `False`.

```

377. def startScreen():
378.     """Display the start screen (which has the title and instructions)
379.     until the player presses a key. Returns None."""
380.
381.     # Position the title image.
382.     titleRect = IMAGESDICT['title'].get_rect()
383.     topCoord = 50 # topCoord tracks where to position the top of the text
384.     titleRect.top = topCoord
385.     titleRect.centerx = HALF_WINWIDTH
386.     topCoord += titleRect.height
387.
388.     # Unfortunately, Pygame's font & text system only shows one line at
389.     # a time, so we can't use strings with \n newline characters in them.
390.     # So we will use a list with each line in it.
391.     instructionText = ['Push the stars over the marks.',
392.                        'Arrow keys to move, WASD for camera control, P to change character.',
393.                        'Backspace to reset level, Esc to quit.',
394.                        'N for next level, B to go back a level.']

```

Функција `startScreen()` треба да прикаже неколико различитих делова текста низ средину прозора. Сваки ред ћемо сачувати као стринг у листи `instructionText`. Слика наслова (смештена у `IMAGESDICT['title']` (која је изворно учитана из `star_title.png` фајла)) ће бити позиционирана 50 пиксела од врха прозора. То је зато што је цео број 50 сачуван у променљивој `topCoord` на линији 383. Променљива `topCoord` ће пратити позиционирање Y осе слике наслова и текста са инструкцијама. X оса ће увек бити подешена тако да се слике и текст центрирају, као у линији 385 за насловну слику.

На линији 386, променљива `topCoord` се повећава без обзира на висину те слике. На тај начин можемо да изменимо слику и да почетни екран не мора да се мења.

```
396. # Start with drawing a blank color to the entire window:
397. DISPLAYSURF.fill(BG_COLOR)
398.
399. # Draw the title image to the window:
400. DISPLAYSURF.blit(IMAGESDICT['title'], titleRect)
401.
402. # Position and draw the text.
403. for i in range(len(instructionText)):
404.     instSurf = BASICFONT.render(instructionText[i], 1, TEXT_COLOR)
405.     instRect = instSurf.get_rect()
406.     topCoord += 10 # 10 pixels will go in between each line of text.
407.     instRect.top = topCoord
408.     instRect.centerx = HALF_WINWIDTH
409.     topCoord += instRect.height # Adjust for the height of the line.
410.     DISPLAYSURF.blit(instSurf, instRect)
```

Линија 400 је место где се слика наслова повезује са приказом површине. За петљу која почиње на линији 403 ће се приказати, позиционирати и блитирати сваки инструкцијски стринг у `instructionText`. Променљива `topCoord` ће увек бити увећана за величину претходно приказаног текста (линија 409) и 10 додатних пиксела (на линији 406, тако да ће бити размак од 10 пиксела између линија текста).

```
412. while True: # Main loop for the start screen.
413.     for event in pygame.event.get():
414.         if event.type == QUIT:
415.             terminate()
416.         elif event.type == KEYDOWN:
417.             if event.key == K_ESCAPE:
418.                 terminate()
419.             return # user has pressed a key, so return.
420.
421. # Display the DISPLAYSURF contents to the actual screen.
422. pygame.display.update()
423. FPSLOCK.tick()
```

У функцији `startScreen()` се налази петља која почиње на линији 412 и обрађује догађаје који указују да ли програм треба да се заврши или да се врати из функције. Док играч не учини исто, петља ће наставити да позива `pygame.display.update()` и `FPSLOCK.tick()` да би се почетни екран приказао на екрану.

## Структуре података у игри Гурање звезда

Гурање звезда има специфичан формат за нивое, мапе и структуре података о стању игре.

### Структура података за стање игре

Објекат за стање игре ће бити речник са три кључа: `'player'`, `'stepCounter'`, и

'stars'.

- Вредност кључа 'player' ће бити тупла од два цела броја за тренутну XY позицију играча.
- Вредност кључа 'stepCounter' ће бити цео број који прати колико је потеза играч направио на овом нивоу (тако да играч у будућности може покушати да реши слагалицу са мањим бројем корака)
- Вредност кључа 'stars' је листа вредности два целобројна тупла XY за сваку од звезда на тренутном нивоу.

### Структура података за мапу

Структура података за мапу је једноставна 2D листа листа где два коришћена индекса представљају X и Y координате мапе. Вредност сваког индекса у листи листа је низ знакова који представља наслов који се налази на тој мапи у сваком простору:

- '#' - Дрвени зид.
- 'x' - Угаони зид.
- '@' - Почетни простор за играча на овом нивоу.
- '.' - Простор циља.
- '\$' - Простор где се звезда налази на почетку нивоа.
- '+' - Простор са циљем и почетним простором играча.
- '\*' - Простор са циљем и звездом на почетку нивоа.
- ' ' - Травнати спољни простор.
- 'o' - Унутрашњи простор- (Ово је мало слово О, а не нула.)
- '1' - Камен на трави.
- '2' - Кратко дрво на трави.
- '3' - Високо дрво на трави.
- '4' - Ружно дрво на трави.

### Структура података за нивое

Објекат нивоа садржи објекат стања игре (који ће бити стање када се први пут покрене), објекат мапе и неколико других вредности. Сам објекат нивоа је речник са следећим кључевима:

- Вредност кључа 'width' је цео број плочица које имају ширину читаве мапе.
- Вредност кључа 'height' је цео број плочица које имају висину читаве мапе.
- Вредност кључа 'mapObj' је објекат мапе за овај ниво.
- Вредност на кључу 'goals' је листа дводелних тупла за XY координате сваког циља на мапи.
- Вредност кључа 'startState' је објекат стања игре који се користи за приказ почетне позиције звезда и играча на почетку нивоа.

### Читање и писање текстуалних датотека

Python има функције за читање датотека са хард диска. Ово ће бити корисно ако имате засебну датотеку која чува све податке за сваки ниво. Ово је такође добра идеја, јер да би добили нове нивое, играч не мора да мења изворни код игре, већ може само да преузме нове датотеке нивоа.

### Текстуалне и бинарне датотеке

Текстуалне датотеке су датотеке које садрже једноставне текстуалне податке. Текстуалне датотеке можете креирати користећи Notepad на Windows оперативном систему, Gedit ако користите Ubuntu, и TextEdit за Mac OS X. Постоје и многи други програми који се називају текстуални едитори и који могу да креирају и мењају текстуалне датотеке. IDLE-ов едитор датотека је едитор текста.

Разлика између текстуалних едитора и процесора текста (као што су Microsoft Word, OpenOffice Writer или iWork Pages) је да текстални едитори имају само текст. Не можете да поставите фонт, величину или боју текста. (IDLE аутоматски поставља боју текста на основу врсте Python кода, али то не можете сами да промените, тако да је и даље текст едитор.) Разлика између текстуалних и бинарних датотека није важна за овај програм, али о томе можете читати на <http://invpy.com/textbinary>. Све што требате знати је ово поглавље, а програм Гурање звезда се бави само текстуалним датотекама.

### Писање у датотеке

Да бисте креирали датотеку, позовите `open()` функцију са два аргумента: стринг за име датотеке и стринг 'w' како би функцији саопштили да желите да пишете у датотеку. Функција `open()` враћа објекат датотеке:

```
>>> textFile = open('hello.txt', 'w')
>>>
```

Ако покренете овај код из интерактивног шела (interactive shell), датотека `hello.txt` коју ова функција креира ће бити креирана у истом фолдеру у којем се налази програм `python.exe` (на Windows оперативном систему то ће вероватно бити `C:\Python32`). Ако је функција `open()` позвана из `.py` програма, датотека се креира у истом фолдеру у којој се налази и `.py` датотека.

Режим „write“ каже функцији `open()` да креира датотеку ако она не постоји. Ако постоји, онда ће `open()` обрисати ту датотеку и креирати нову, празну датотеку. Ово је исто као и начин на који израз за доделу може да креира нову променљиву или препише тренутну вредност у већ постојећој променљивој. Ово може бити веома опасно. Ако случајно за име датотеке пошаљете неку важну датотеку `open()` функцији са 'w' као другим параметром, она ће бити избрисана. То би могло резултирати поновним инсталирањем оперативног система на вашем рачунару и/или лансирањем нуклеарних пројектила.

Објекат датотеке има методу `write()` која се може користити за писање текста у датотеку. Само му додајте стринг као што бисте проследили стринг `print()` функцији. Разлика је у томе што `write()` не додаје аутоматски нови ред (`'\n'`) на крај стринга. Ако желите додати нови ред, мораће те га укључити у стринг:

```
>>> textFile = open('hello.txt', 'w')
>>> textFile.write('This will be the content of the file.\nHello world!\n')
>>>
```

Да бисте у програмском језику Python рекли да сте завршили писање садржаја у ову датотеку, требате позвати `close()` методу објекта датотеке. (Иако ће Python аутоматски затворити све отворене објекте датотека када се програм заврши.)

```
>>> textFile.close()
```

### Читање из датотека

Да бисте прочитали садржај датотеке, пошаљите стринг 'r' уместо 'w' као други параметар функције `open()`. Затим позовите `readlines()` методу за објекат датотеке да прочитате садржај датотеке. На крају, затворите датотеку позивањем методе `close()`.

```
>>> textFile = open('hello.txt', 'r')
>>> content = textFile.readlines()
>>> textFile.close()
```



Метода `readlines()` враћа листу стрингова: један стринг за сваки ред текста у датотеци:

```
>>> content
['This will be the content of the file.\n', 'Hello world!\n']
>>>
```

Ако желите да поново прочитате садржај те датотеке, мораћете да позовете `close()` методу објекта датотеке и поново га отворите.

Као алтернативу за `readlines()`, такође можете позвати `read()` методу, која ће вратити цео садржај датотеке као један стринг:

```
>>> textFile = open('hello.txt', 'r')
>>> content = textFile.read()
>>> content
'This will be the content of the file.\nHello world!\n'
```

Са друге стране, ако изоставите други параметар за функцију `open()`, Python ће претпоставити да желите да отворите датотеку у режиму за читање. `open('foobar.txt', 'r')` и `open('foobar.txt')` раде исту ствар.

### О формату датотеке мапе за игру Гурање звезда

Потребно је да текстуална датотека нивоа буде у одређеном формату. Који знакови представљају зидове или звезде или почетну позицију играча? Ако имамо мапе за више нивоа, како можемо да знамо када се мапа једног нивоа завршава и да почиње следећа?

Срећом, формат датотек мапе који чемо користити је већ дефинисан за нас. Постоји много „Sokoban“ игара (можете наћи више на <http://invpy.com/sokobanclones>), и сви они користе исти формат датотеке мапе. Ако преузмете датотеку нивоа са <http://invpy.com/starPusherLevels.txt> и отворите је у текст едитору, видећете нешто слично овоме:

```
; Star Pusher (Sokoban clone)
; http://inventwithpython.com/blog
; By Al Sweigart al@inventwithpython.com
;
; Everything after the ; is a comment and will be ignored by the game
that
; reads in this file.
;
; The format is described at:
; http://sokobano.de/wiki/index.php?title=Level_format
; @ - The starting position of the player.
; $ - The starting position for a pushable star.
; . - A goal where a star needs to be pushed.
; + - Player & goal
; * - Star & goal
; (space) - an empty open space.
; # - A wall.
;
; Level maps are separated by a blank line (I like to use a ; at the
start
; of the line since it is more visible.)
;
```

```
; I tried to use the same format as other people use for their Sokoban
games,
; so that loading new levels is easy. Just place the levels in a text
file
; and name it "starPusherLevels.txt" (after renaming this file, of
course).
```

```
; Starting demo level: #####
##  #
#   .   #
#  $    # # .$$ . # ##### $  #
# .   #
#    ##
#####
```

Коментари на врху датотеке објашњавају формат датотеке. Када учитате први ниво, добијате следећи изглед:



```
426. def readLevelsFile(filename):
427.     assert os.path.exists(filename), 'Cannot find the level file: %s' % (filename)
```

Функција `os.path.exists()` ће вратити `True` ако постоји датотека која је названа као стринг који је прослеђен функцији. Ако не постоји, `os.path.exists()` враћа `False`.

```
428. mapFile = open(filename, 'r')
429. # Each level must end with a blank line
430. content = mapFile.readlines() + ['\r\n']
431. mapFile.close()
432.
433. levels = [] # Will contain a list of level objects.
434. levelNum = 0
435. mapTextLines = [] # contains the lines for a single level's map.
436. mapObj = [] # the map object made from the data in mapTextLines
```

Објекат датотеке нивоа која је отворена за читање се смешта у `mapFile`. Сав текст из датотеке се чува као листа стрингова у променљивој `content`, са празном линијом која се додаје на крај. (Разлога за то је објашњен касније.)

Након што се креирају објекти нивоа, они ће бити сачувани у листи нивоа. Променљива `levelNum` ће пратити колико је нивоа пронађено унутар датотеке нивоа. Листа `mapTextLines` ће бити листа стрингова из листе `content` за једну мапу (за разлику од тога како `content` складишти стрингове свих мапа у датотеци нивоа). Променљива `mapObj` ће бити 2D листа.

```
437. for lineNum in range(len(content)):
438.     # Process each line that was in the level file.
439.     line = content[lineNum].rstrip('\r\n')
```

For петља на линији 437 ће проћи кроз сваку линију која је прочитана из нивоа датотеке по ред по ред. Број линије ће бити сачуван у lineNum и стринг текста за линију ће бити сачуван у line. Сви знакови новог реда на крају низа биће уклоњени.

```
441. if ';' in line:
442.     # Ignore the ; lines, they're comments in the level file.
443.     line = line[:line.find(';')]
```

Сваки текст који стоји након ; у датотеци мапе третира се као коментар и занемарује се. Ово је као знак # за Python коментаре. Да бисмо били сигурни да наш код не мисли да је коментар део мапе, променљива line се мења тако да се састоји само од текста до (али не укључујући) знака ;. (Запамтите да ово мења само стринг у листи content. Не мења датотеку нивоа на хард диску.)

```
445. if line != '':
446.     # This line is part of the map.
447.     mapTextLines.append(line)
```

У датотеци мапе могу постојати мапе за више нивоа. Листа mapTextLines ће садржати линије текста из датотек емапе за тренутни ниво који се учитава. Све док тренутна линија није празна, линија ће се додати на крај mapTextLines листе.

```
448. elif line == '' and len(mapTextLines) > 0:
449.     # A blank line indicates the end of a level's map in the file.
450.     # Convert the text in mapTextLines into a level object.
```

Када постоји празан ред у датотеци мапе, то значи да се мапа за тренутни ниво завршила. А будуће линије текста ће бити за касније нивое. Обратите пажњу да мора бити бар један ред у mapTextLines тако да се више празних редова не рачуна као почетак и заустављање више нивоа.

```
452.     # Find the longest row in the map.
453.     maxWidth = -1
454.     for i in range(len(mapTextLines)):
455.         if len(mapTextLines[i]) > maxWidth:
456.             maxWidth = len(mapTextLines[i])
```

Сви стрингови у mapTextLines листи морају бити исте дужине (тако да формирају правоугаоник), тако да би требало да буду обогаћени додатним празним размацама све до најдужег стринга. For петља пролази кроз сваки од стрингова у mapTextLines и ажурира maxWidth када пронађе нови најдужи стринг. Након завршетка ове петље, променљива maxWidth ће бити постављена на дужину најдужег стринга у mapTextLines.

```
457.     # Add spaces to the ends of the shorter rows. This
458.     # ensures the map will be rectangular.
459.     for i in range(len(mapTextLines)):
460.         mapTextLines[i] += ' ' * (maxWidth - len(mapTextLines[i]))
```

For петља на линији 459 поново пролази кроз стрингове у mapTextLines, овај пут да би додали довољно знакова за размак како би били дуги као maxWidth.

```

462.     # Convert mapTextLines to a map object.
463.     for x in range(len(mapTextLines[0])):
464.         mapObj.append([])
465.     for y in range(len(mapTextLines)):
466.         for x in range(maxWidth):
467.             mapObj[x].append(mapTextLines[y][x])

```

Променљива `mapTextLines` само чува листу стрингова. (Сваки стринг у листи представља ред, а сваки знак у стрингу представља знак у различитој колони. Због тога линија 467 има Y и X индексе обрнуте, баш као и структура података SHAPES у игри Тетромينو.) Објекат мапе ће морати да буде листа листе једноцифрених стрингова тако да `mapObj[x][y]` означава плочицу на XY координатама. За петљу на линији 463 додаје се празна листа `mapObj` за сваку колону у `mapTextLines`.

Угњежене петље на линијама 465 и 466 ће попунити ове листе стринговима са једним знаком з апредстављање сваке плочице на мапи. Ово ствара објекат мапе коју играч користи.

```

469.     # Loop through the spaces in the map and find the @, ., and $
470.     # characters for the starting game state.
471.     startx = None # The x and y for the player's starting position
472.     starty = None
473.     goals = [] # list of (x, y) tuples for each goal.
474.     stars = [] # list of (x, y) for each star's starting position.
475.     for x in range(maxWidth):
476.         for y in range(len(mapObj[x])):
477.             if mapObj[x][y] in ('@', '+'):
478.                 # '@' is player, '+' is player & goal
479.                 startx = x
480.                 starty = y
481.             if mapObj[x][y] in ('.', '*', '$'):
482.                 # '.' is goal, '*' is star & goal
483.                 goals.append((x, y))
484.             if mapObj[x][y] in ('$ ', '*'):
485.                 # '$' is star
486.                 stars.append((x, y))

```

Након креирања објекта мапе, угњежене петље на линијама 475 и 476 проћи ће кроз сваки простор како би пронашли XY координате три ствари:

1. Стартна позиција играча. Ово ће бити сачувано у `startx` и `starty` променљивама, које ће касније бити сачуване у објекту стања игре на линији 494.
2. Почетне позиције свих звезда. Оне ће бити сачуване у листи `stars`, која ће касније бити смештена у објекат стања игре на линији 496.
3. Положај свих циљева. Они ће бити сачувани у листи `goals`, који ће касније бити смештени у објекат `level` на линији 500.

Апаметите, објекат стања игре садржи све ствари које се могу променити. Због тога је у њему сачувана позиција играча (зато што се играч може кретати) и зашто су звезде сачуване у њему (зато што их играч може гурати окол). Али циљеви се чувају у објекту нивоа, јер се они никад неће кретати.

```

488.     # Basic level design sanity checks:
489.     assert startx != None and starty != None, 'Level %s (around
line %s) in %s is missing a "@" or "+" to mark the start point.' % (levelNum+1, lineNum, filename)
490.     assert len(goals) > 0, 'Level %s (around line %s) in %s must
have at least one goal.' % (levelNum+1, lineNum, filename)
491.     assert len(stars) >= len(goals), 'Level %s (around line %s) in
%s is impossible to solve. It has %s goals but only %s stars.' % (levelNum+1, lineNum, filename, len(goals), len(stars))

```

У овом тренутку, ниво је прочитан и обрађен. Да бисте били сигурни да ће овај ниво функционисати исправно, мора проћи неколико тврдњи. Ако је било који од услова за ове тврдње False, онда ће Python произвести грешку (користећи стринг из тврдње) говорећи шта није у реду са датотеком нивоа.

Прва тврдња на линији 489 проверава да ли постоји полазна тачка играча наведена негде на мапи. Друга тврдња на линији 490 проверава да ли постоји бар један циљ (или више) негде на мапи. И трећа тврдња на линији 491 проверава да ли постоји барем једна звезда за сваки циљ (али има више звезда од циљева.)

```
493.     # Create level object and starting game state object.
494.     gameStateObj = {'player': (startx, starty),
495.                     'stepCounter': 0,
496.                     'stars': stars}
497.     levelObj = {'width': maxWidth,
498.                'height': len(mapObj),
499.                'mapObj': mapObj,
500.                'goals': goals,
501.                'startState': gameStateObj}
502.
503.     levels.append(levelObj)
```

Коначно, ови објекти се чувају у објекту стања игре, који је сам сачуван у објекту нивоа. Објекат нивоа се додаје листи објеката нивоа на линији 503. Ова листа нивоа ће бити враћена од стране функције readLevelsFile() када су све мапе обрађене.

```
505.     # Reset the variables for reading the next map.
506.     mapTextLines = []
507.     mapObj = []
508.     gameStateObj = {}
509.     levelNum += 1
510.     return levels
```

Сада када је завршена обрада за овај ниво, променљиве mapTextLines, mapObj и gameStateObj треба да се ресетују на прасне вредности за следећи ниво који ће се читати из датотеке нивоа. Променљива levelNum се такође повећава за 1 и представља број следећег нивоа.

### Рекурзивне функције

Пре него што научите како ради floodFill() функција, морате научити нешто о рекурзији. Рекурзија је једноставан концепт: рекурзивна функција је само функција која позива себе, попут једне у следећем програму: (не уписуј слова на почетку сваког реда)

```
A. def passFortyTwoWhenYouCallThisFunction(param):
B.     print('Start of function.')
C.     if param != 42:
D.         print('You did not pass 42 when you called this function.')
E.         print('Fine. I will do it myself.')
F.         passFortyTwoWhenYouCallThisFunction(42) # this is the recursive call
G.     if param == 42:
H.         print('Thank you for passing 42 when you called this function.')
I.         print('End of function.') J.
K.     passFortyTwoWhenYouCallThisFunction(41)
```

(У вашим програмима, немојте да функције имају дугачка имена попут passFortyTwoWhenYouCallThisFunction()).

Када покренете овај програм, функција се дефинише када се изврши `def` израз на линији А. Следећа линија кода која се извршава је линија К, која позива `passFortyTwoWhenYouCallThisFunction()` и прихвата као параметар 41. Као резултат функција позива себе на линији F и као параметар има 42. Овај позив називамо рекурзивни позив.

Ово је излаз вашег програма:

```
Start of function.  
You did not pass 42 when you called this function. Fine. I will do it myself.  
Start of function.  
Thank you for passing 42 when you called this function.  
End of function.  
End of function.
```

Обратите пажњу да се текст „Start of function“ и „End of function“ појављује два пута. Хајде да схватимо шта се тачно дешава и којим редоследом се дешава.

На линији К, функција је позвана и 41 је прослеђено као параметар. Линија В штампа „Start of function“. Услов на линији С ће бити `True` (`41 != 42`) па ће линије С и D одштампати своје поруке. Линија F ће затим рекурзивно позвати функцију и као параметар добија 42. Дакле, извршење се поново покреће на линији В и штампа „Start of function“. Услов линије С овај пут је `False`, тако да прелази на линију G и налази да је услов `True`. Ово узрокује позивање линије H и на екрану приказује „Thank you...“. Тада ће се задња линија функције, линија I, извршити и исписати „End of function“ и функција ће се вратити на линију која ју је позвала.

Али запамтите, линија кода која је позвала функцију је била линија F. И у овом оригиналном позиву, параметар је постављен на 41. Код иде до линије G и проверава услов, који је `False` (`41==41` је `False`) тако да прескаче позив `print()` функције иде се на линију I која по други пут штампа „End of function“.

Пошто је достигао крај функције, враћа се на линију кода која је позвала функцију, која је била линија К. Нема више линија кода после линије К, тако да се програм завршава.

Имајте на уму да локалне променљиве нису само локалне за функцију, већ и за специфичан позив функције.

### Преливање стека

Сваки пут када се позове функција, Python интерпретер памти која је линија кода извршила позив. На тај начин када функција врати Python зна где да настави извршавање. Памћење овога заузима мало меморије. То обично није велика ствар, али погледајте следећи код:

```
def funky():  
    funky()  
  
funky()
```

Ако покренете овај програм, добићете велику количину излаза који изгледају овако:

```
...
File "C:\test67.py", line 2, in funky
    funky()
File "C:\test67.py", line 2, in funky
    funky()
File "C:\test67.py", line 2, in funky
    funky()
File "C:\test67.py", line 2, in funky
    funky()
File "C:\test67.py", line 2, in funky
    funky()
RuntimeError: maximum recursion depth exceeded
```

Функција `funky()` не ради ништа друго осим што позива сама себе. А онда у том позиву, функција се поново позива. Онда се поново позива, и опрт, и поново. Сваки пут када се позове, Python мора да запамти која линија кода је направила позив како би знао где да се врати након извршавања. Али `funky()` функција се никад не враћа, веч само наставља да позива себе.

Ово је као бесконачна петља са петљама, где програм наставља да ради и никада не престаје. Да би се спречило да му понестане меморије, Python ће узроковати грешку након 1000 позива у дубину и прекинути програм. Овај тип грешке се назива преливањ њ стека (stack overflow).

Овај код такође доводи до исте грешке, иако нема рекурзивних функција:

```
def spam():
    eggs()

def eggs():
    spam()

spam()
```

Када покренете овај програм, добија се следећа грешка:

```
...
File "C:\test67.py", line 2, in spam
    eggs()
File "C:\test67.py", line 5, in eggs
    spam()
File "C:\test67.py", line 2, in spam
    eggs()
File "C:\test67.py", line 5, in eggs
    spam()
File "C:\test67.py", line 2, in spam
    eggs()
RuntimeError: maximum recursion depth exceeded
```

### Спречавање преливања стека основним случајем

Да бисте спречили грешке у преливању стека, морате имати основни случај где функција зауставља нове рекурзивне позиве. Ако не постоји основни случај, позиви функција никад неће престати и долази до преливања стека. Ево примера рекурзивне функције са основним случајем. Основни случај је када је параметар `param` једнак 2.

```
def fizz(param):  
    print(param)  
    if param == 2:  
        return  
    fizz(param - 1)  
  
fizz(5)
```

Када покренете програм, добићете следећи излаз:

```
5  
4  
3  
2
```

Овај програм нема грешку преливања стека јер једном кад се параметар `param` сетује на 2, `if` израз ће бити `True` и функција ће се вратити, а затим ће се и остали позиви вратити.

Ако ваш код никад не достигне основни случај, то ће изазвати преливање стека. Ако променимо `fizz(5)` позив на `fizz(0)`, излаз програма би изгледао овако:

```
 , ,
```

Рекурзивни позив и основни случајеви ће се користити за извођење Flood Fill алгоритма, који је описан у наставку.

### Flood Fill алгоритам

Flood Fill алгоритам се у програму Гутање звезда користи за промену свих подних плочица унутар зидова нивоа да би се користиле унутрашње подне плочице уместо спољашњих. Оригинални `floodFill()` позив је на линији 295. Он ће претворити све плочице представљене стрингом `' '` (који представља спољашњи под) у стринг `'o'` (који представља унутрашњи под).



```

513. def floodFill(mapObj, x, y, oldCharacter, newCharacter):
514.     """Changes any values matching oldCharacter on the map object to
515.     newCharacter at the (x, y) position, and does the same for the
516.     positions to the left, right, down, and up of (x, y), recursively."""
517.
518.     # In this game, the flood fill algorithm creates the inside/outside
519.     # floor distinction. This is a "recursive" function.
520.     # For more info on the Flood Fill algorithm, see:
521.     # http://en.wikipedia.org/wiki/Flood\_fill
522.     if mapObj[x][y] == oldCharacter:
523.         mapObj[x][y] = newCharacter

```

Линије 522 и 523 конвертују плочицу на XY координатама која је прослеђена у floodFill() на newCharacter стринг ако је оригинално био исти као и oldCharacter стринг.

```

525.     if x < len(mapObj) - 1 and mapObj[x+1][y] == oldCharacter:
526.         floodFill(mapObj, x+1, y, oldCharacter, newCharacter) # call right
527.     if x > 0 and mapObj[x-1][y] == oldCharacter:
528.         floodFill(mapObj, x-1, y, oldCharacter, newCharacter) # call left
529.     if y < len(mapObj[x]) - 1 and mapObj[x][y+1] == oldCharacter:
530.         floodFill(mapObj, x, y+1, oldCharacter, newCharacter) # call down
531.     if y > 0 and mapObj[x][y-1] == oldCharacter:
532.         floodFill(mapObj, x, y-1, oldCharacter, newCharacter) # call up

```

Ове четири наредбе if проверавају да ли је почица десно, лево, горе и доле од XY координата иста као oldCharacter, и ако је тако, направљен је рекурзивни позив функције floodFill() са тим координатама.

Да би боље разумели како ради функција floodFill(), овде је верзија која не користи рекурзивне позиве, али уместо тога користи листу XY координата да би пратила које просторе на мапи треба да провери и њвњнтуално промени у newCharacter.

```

def floodFill(mapObj, x, y, oldCharacter, newCharacter):
    spacesToCheck = []
    if mapObj[x][y] == oldCharacter:
        spacesToCheck.append((x, y))
    while spacesToCheck != []:
        x, y = spacesToCheck.pop()
        mapObj[x][y] = newCharacter

        if x < len(mapObj) - 1 and mapObj[x+1][y] == oldCharacter:
            spacesToCheck.append((x+1, y)) # check right
        if x > 0 and mapObj[x-1][y] == oldCharacter:
            spacesToCheck.append((x-1, y)) # check left
        if y < len(mapObj[x]) - 1 and mapObj[x][y+1] == oldCharacter:
            spacesToCheck.append((x, y+1)) # check down
        if y > 0 and mapObj[x][y-1] == oldCharacter:
            spacesToCheck.append((x, y-1)) # check up

```

Ако желите да прочитате детаљнији водич о рекурзији који користи мачке и зомбије за пример, идите на <http://invpy.com/recursivezombies>.

## Цртање мапе

```
535. def drawMap(mapObj, gameStateObj, goals):
536.     """Draws the map to a Surface object, including the player and
537.     stars. This function does not call pygame.display.update(), nor
538.     does it draw the "Level" and "Steps" text in the corner."""
539.
540.     # mapSurf will be the single Surface object that the tiles are drawn
541.     # on, so that it is easy to position the entire map on the DISPLAYSURF
542.     # Surface object. First, the width and height must be calculated.
543.     mapSurfWidth = len(mapObj) * TILEWIDTH
544.     mapSurfHeight = (len(mapObj[0]) - 1) * (TILEHEIGHT - TILEFLOORHEIGHT) + TILEHEIGHT
545.     mapSurf = pygame.Surface((mapSurfWidth, mapSurfHeight))
546.     mapSurf.fill(BG_COLOR) # start with a blank color on the surface.
```

Функција drawMap() ће вратити Surface објекат са целом мапом (и играчем и звездама) на њој. Ширина и висина потребне за ову површину морају се израчунати из mapObj (што се ради на линијама 543 и 544). Surface објекат на којем ће се све нацртати креира се на линији 545. За почетак, цео Surface објекат је обојен у боју позадине на линији 546.

```
548. # Draw the tile sprites onto this surface.
549. for x in range(len(mapObj)):
550.     for y in range(len(mapObj[x])):
551.         spaceRect = pygame.Rect((x * TILEWIDTH, y * (TILEHEIGHT -
TILEFLOORHEIGHT), TILEWIDTH, TILEHEIGHT))
```

Скуп угњеждених for петљи на линијама 549 и 550 ће проћи кроз све могуће XY координате на мапи и нацртати одговарајућу слику плочице на тој локацији.

```
552.     if mapObj[x][y] in TILEMAPPING:
553.         baseTile = TILEMAPPING[mapObj[x][y]]
554.     elif mapObj[x][y] in OUTSIDEDECOMAPPING:
555.         baseTile = TILEMAPPING[' ']
556.
557.     # First draw the base ground/wall tile.
558.     mapSurf.blit(baseTile, spaceRect)
559.
```

Променљива baseTile је постављена на Surface објекат слике која се црта на тренутној XY координати итерације. Ако је јједнозначни стринг у OUTSIDEDECOMAPPING речнику, користиће се TILEMAPPING[' '] (стринг од једног знака за основни спољни под).

```
560.     if mapObj[x][y] in OUTSIDEDECOMAPPING:
561.         # Draw any tree/rock decorations that are on this tile.
562.         mapSurf.blit(OUTSIDEDECOMAPPING[mapObj[x][y]], spaceRect)
```

Порде тога, ако се плочица налази у речнику OUTSIDEDECOMAPPING одговарајућа слика дрвета или камена треба бити нацртана на врху плочице која је управо нацртана на тој XY координати.

```
563.     elif (x, y) in gameStateObj['stars']:
564.         if (x, y) in goals:
565.             # A goal AND star are on this space, draw goal first.
566.             mapSurf.blit(IMAGESDICT['covered goal'], spaceRect)
567.             # Then draw the star sprite.
568.             mapSurf.blit(IMAGESDICT['star'], spaceRect)
```

Ако постоји звезда лоцирана на овој XY координати на мапи (која се може сазнати проверавањем за (x, y) у листи gameStateObj['stars']), онда треба да се нацрта звезда на позицији XY (која се врши на линији 568). Пре него што је звезда нацртана, код треба прво проверити да ли постоји и циљ на овој локацији, у ком случају, прво треба исцртати плочицу „covered goal“.

```
569.     elif (x, y) in goals:
570.         # Draw a goal without a star on it.
571.         mapSurf.blit(IMAGESDICT['uncovered goal'], spaceRect)
```

Ако постоји циљ на XY координати, онда се „uncovered goal“ треба нацртати на врху плочице. „uncovered goal“ је нацртан јер ако је извршење достигло elif израз на линији 569, знамо да је стање elif израза на линији 563 било False и да нема звезде која је такође на овој XY координати.

```
573.     # Last draw the player on the board.
574.     if (x, y) == gameStateObj['player']:
575.         # Note: The value "currentImage" refers
576.         # to a key in "PLAYERIMAGES" which has the
577.         # specific player image we want to show.
578.         mapSurf.blit(PLAYERIMAGES[currentImage], spaceRect)
579.
580.     return mapSurf
```

Конечно, функција drawMap() проверава да ли се играч налази на овој XY координати, а ако је тако, слика играча је нацртана преко плочице. Линија 580 се налази изван угњеждених петљи које су почеле на линији 549 и 550, тако да је до тренутка враћања Surface објекта нацртана цела мапа.

### Провера да ли је ниво завршен

```
583. def isLevelFinished(levelObj, gameStateObj):
584.     """Returns True if all the goals have stars in them."""
585.     for goal in levelObj['goals']:
586.         if goal not in gameStateObj['stars']:
587.             # Found a space with a goal but no star on it.
588.             return False
589.     return True
```

Функција isLevelFinished() враћа True ако су сви циљеви покривени звездама. Неки нивои могу имати више звезда од циљева, па је важно да проверите да ли су сви циљеви покривени звездама, уместо да проверите да ли су све звезде изнад циљева.

For петља на линији 585 пролази кроз циљеве у levelObj['goals'] (што је листа торки XY координата за сваки циљ) и проверава да ли постоји звезда у листи gameStateObj['stars'] која има оне исте XY координате. Када први пут пронађете циљ без звезде на истој позицији, функција враћа False.

Ако прође кроз све циљеве и пронађе звезду на свакој од њих, isLevelFinished() вратиће True.

```
592. def terminate():
593.     pygame.quit()
594.     sys.exit()
```

Ова функција terminate() је иста као у свим претходним програмима.

```
597. if name == 'main':
598.     main()
```

Након дефинисања свих функција, функција `main()` се позива на линији 602 да започне игру.

### Сажетак

У игри Веверица једе Веверицу, свет игре је био прилично једноставан: само бесконачна зелена равница са сликама траве насумично разбацаним око ње. Игра Гурање звезда је увела нешто ново: има јединствено дизајниране нивое са графичким плочицама. Да би се ови нивои чували у формату који рачунар може да чита, они се откуцају у текстуалну датотеку и код у програму чита те датотеке и креира структуре података за ниво.

Стварно, радије него да само направите једноставну игру са једном мапом, Гурање звезда је више систем за учитавање прилагођених мапа на основу датотеке нивоа. Само мењајући датотеку нивоа, можемо да променимо где се зидови, звезде и циљеви појављују у свету игре. Програм Гурање звезда може да обради било коју конфигурацију коју је датотека нивоа поставила (све док прослеђује изразе који обезбеђују да мапа има смисла).

Не морате чак ни да знате како да програмирате у језику Python да направите сопствене нивое. Програм за уређивање текста који мења датотеку `starPusherLevels.txt` је све што је потребно да свако може да буде креатор нивоа за ову игру.

За додатну праксу програмирања, можете преузети `buggy` верзије игре са <http://invpy.com/buggy/starpusher> и покушати схватити како да исправите грешке.