ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

SQUIRREL EAT SQUIRREL

How to Play Squirrel Eat Squirrel

- Squirrel Eat Squirrel is loosely based on the game “Katamari Damacy”.
  - https://en.wikipedia.org/wiki/Katamari_Damacy
- The player controls a small squirrel that hops around the screen eating smaller squirrels and avoiding larger squirrels.
  - Each time the player’s squirrel eats a smaller squirrel grows larger.
  - If the player’s squirrel gets hit by a larger squirrel loses a life point.
- The player
  - wins when the squirrel becomes a monstrously large squirrel called the Omega Squirrel.
  - loses if their squirrel gets hit three times.

Source Code to Squirrel Eat Squirrel

1

- This source code can be downloaded from http://invpy.com/squirrel.py.
- You will also need to download the following image files:
  - http://invpy.com/gameicon.png
  - http://invpy.com/squirrel.png
  - http://invpy.com/grass1.png
  - http://invpy.com/grass2.png
  - http://invpy.com/grass3.png
  - http://invpy.com/grass4.png

The Design of Squirrel Eat Squirrel

- There are three types of data structures represented as **dictionary values**.
  - player squirrel (**only one player squirrel object at a time in the game**),
  - enemy squirrels, and
  - grass objects.
- All objects have the following keys in their dictionary value
  - The 'x' and 'y' key’s value - the coordinates of the top left of the object in ***game world coordinates***.
  - 'rect' key’s value tracks pixel coordinates.
- The player squirrel, enemy squirrel, and grass objects **have other keys.**

The Usual Setup Code

```
# Squirrel Eat Squirrel (a 2D Katamari Damacy clone)
# By Al Sweigart al@inventwithpython.com
# http://inventwithpython.com/pygame
# Creative Commons BY-NC-SA 3.0 US

import random, sys, time, math, pygame
from pygame.locals import *

FPS = 30 # frames per second to update the screen
WINWIDTH = 640 # width of the program's window, in pixels
WINHEIGHT = 480 # height in pixels
HALF_WINWIDTH = int(WINWIDTH / 2)
HALF_WINHEIGHT = int(WINHEIGHT / 2)

GRASSCOLOR = (24, 255, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
```

The Usual Setup Code

19. CAMERASLACK = 90 # how far from the center the squirrel moves before moving the camera

- The camera will begin following the player squirrel when it moves 90 pixels away from the center of the window.

The Usual Setup Code

2

```
MOVERATE = 9 # how fast the player moves
BOUNCERATE = 6 # how fast the player bounces (large is slower)
BOUNCEHEIGHT = 30 # how high the player bounces
STARTSIZE = 25 # how big the player starts off
WINSIZE = 300 # how big the player needs to be to win
INVULNTIME = 2 # how long the player is invulnerable after being hit in seconds
GAMEOVERTIME = 4 # how long the "game over" text stays on the screen in seconds
MAXHEALTH = 3 # how much health the player starts with

NUMGRASS = 80 # number of grass objects in the active area
NUMSQUIRRELS = 30 # number of squirrels in the active area
SQUIRRELMINSPEED = 3 # slowest squirrel speed
SQUIRRELMAXSPEED = 7 # fastest squirrel speed
DIRCHANGEFREQ = 2 # % chance of direction change per frame
LEFT = 'left'
RIGHT = 'right'
```

Describing the Data Structures 3

37. """
38. This program has three data structures to represent the player, enemy squirrels, and grass background objects. The data structures are dictionaries with the following keys:
39.
40. Keys used by all three data structures:
41. 'x' - the left edge coordinate of the object in the game world (not a pixel coordinate on the screen)
42. 'y' - the top edge coordinate of the object in the game world (not a pixel coordinate on the screen)
43. 'rect' - the pygame.Rect object representing where on the screen the object is located.
44. Player data structure keys:
45. 'surface' - the pygame.Surface object that stores the image of the squirrel which will be drawn to the screen.
46. 'facing' - either set to LEFT or RIGHT, stores which direction the player is facing.
47. 'size' - the width and height of the player in pixels. (The width & height are always the same.)
48. 'bounce' - represents at what point in a bounce the player is in. 0 means standing (no bounce), up to BOUNCERATE (the completion of the bounce)
49. 'health' - an integer showing how many more times the player can be hit by a larger squirrel before dying.
50. Enemy Squirrel data structure keys:

Describing the Data Structures

51. 'surface' - the pygame.Surface object that stores the image of the squirrel which will be drawn to the screen.
52. 'movex' - how many pixels per frame the squirrel moves horizontally. A negative integer is moving to the left, a positive to the right.
53. 'movey' - how many pixels per frame the squirrel moves vertically. A negative integer is moving up, a positive moving down.
54. 'width' - the width of the squirrel's image, in pixels
55. 'height' - the height of the squirrel's image, in pixels
56. 'bounce' - represents at what point in a bounce the player is in. 0 means standing (no bounce), up to BOUNCERATE (the completion of the bounce)
57. 'bouncerate' - how quickly the squirrel bounces. A lower number means a quicker bounce.
58. 'bounceheight' - how high (in pixels) the squirrel bounces
59. Grass data structure keys:
60. 'grassImage' - an integer that refers to the index of the pygame.Surface object in GRASSIMAGES used for this grass object
61. """

The main() Function

4

```
def main():
global FPSCLOCK, DISPLAYSURF, BASICFONT, L_SQUIR_IMG, R_SQUIR_IMG,
GRASSIMAGES

pygame.init()
FPSCLOCK = pygame.time.Clock()
pygame.display.set_icon(pygame.image.load('gameicon.png'))
DISPLAYSURF = pygame.display.set_mode((WINWIDTH, WINHEIGHT))
pygame.display.set_caption('Squirrel Eat Squirrel')
BASICFONT = pygame.font.Font('freesansbold.ttf', 32)
```

- The pygame.display.set_icon() is a Pygame function that sets the icon for the game window.
  - The single argument to pygame.display.set_icon() is a Surface object of a small image.
  - The ideal image size is 32 x 32 pixels, although you can use other sized images.
  - The image will just be compressed into a smaller size to be used as the window’s icon.

The pygame.transform.flip() function

```
# load the image files
L_SQUIR_IMG = pygame.image.load('squirrel.png')
R_SQUIR_IMG = pygame.transform.flip(L_SQUIR_IMG, True, False)
GRASSIMAGES = []
for i in range(1, 5):
GRASSIMAGES.append(pygame.image.load('grass%s.png' % i))
while True:
runGame()
```

The pygame.transform.flip() function

- The image for the player and enemy squirrels is loaded from *squirrel.png*.
  - Make sure that this PNG file is in the same folder as *squirrel.py*
- The image in *squirrel.png* is of a squirrel facing to the left.
  - We also need a Surface object that contains a picture of the squirrel facing to the right.
  - Instead of creating a second PNG image file, we call the pygame.transform.flip() function.
  - This function has three parameters:
    - the Surface object with the image to flip,
    - a Boolean value to do a horizontal flip, and
    - a Boolean value to do a vertical flip.
  - By passing True for the second parameter and False for the third parameter, the Surface object that returns has the image of the squirrel facing to the right.
  - The original Surface object in L_SQUIR_IMG that we passed in is unchanged.

Original

Horizontal Flip

Vertical Flip

Horizontal and Vertical Flip

A More Detailed Game State than Usual

```
def runGame():
# set up variables for the start of a new game
invulnerableMode = False # if the player is invulnerable
invulnerableStartTime = 0 # time the player became invulnerable
gameOverMode = False # if the player has lost
gameOverStartTime = 0 # time the player lost
winMode = False # if the player has won
```

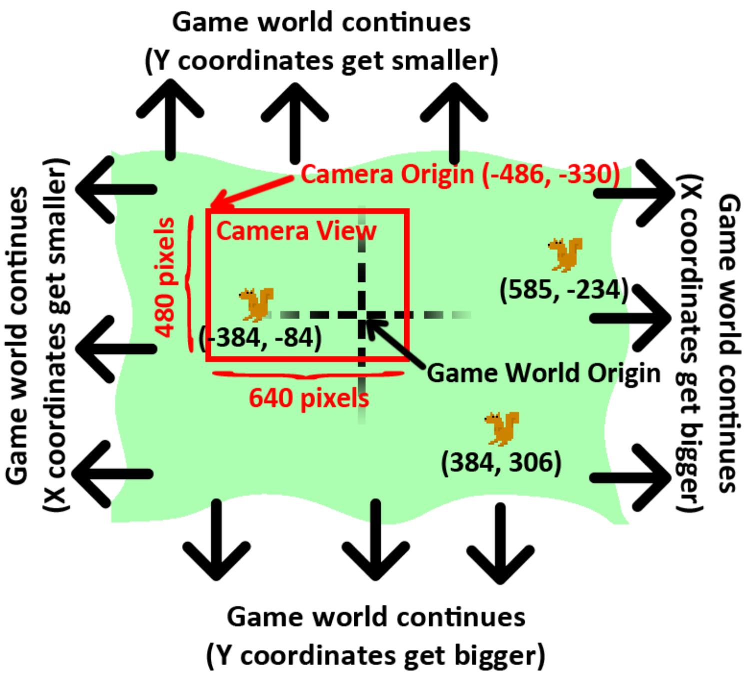
The Usual Text Creation Code

```
# create the surfaces to hold game text
gameOverSurf = BASICFONT.render('Game Over', True, WHITE)
gameOverRect = gameOverSurf.get_rect()
gameOverRect.center = (HALF_WINWIDTH, HALF_WINHEIGHT)

winSurf = BASICFONT.render('You have achieved OMEGA SQUIRREL!', True, WHITE)
winRect = winSurf.get_rect()
winRect.center = (HALF_WINWIDTH, HALF_WINHEIGHT)

winSurf2 = BASICFONT.render('(Press "r" to restart.)', True, WHITE)
winRect2 = winSurf2.get_rect()
winRect2.center = (HALF_WINWIDTH, HALF_WINHEIGHT + 30)
```

Cameras

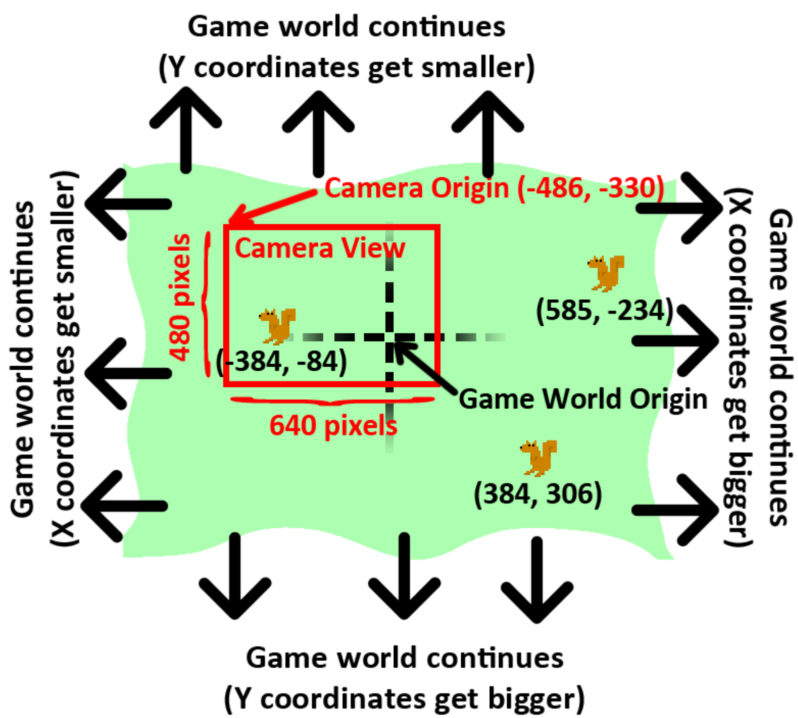


105. # camerax and cameray are where the middle of the camera view is
106. camerax = 0
107. cameray = 0

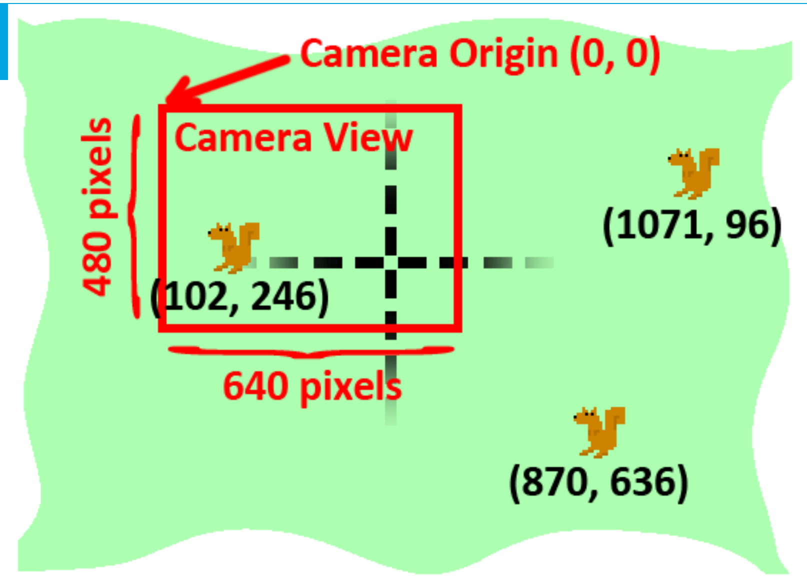
- The camerax and cameray variables track the game coordinates of the —camera.
  - Imagine the game world as an infinite 2D space.
  - This could, of course, never fit on any screen.
  - We can only draw a portion of the infinite 2D space on the screen.
  - We call the area of this portion a **camera**

Cameras

- The game world origin is where the (0, 0) game world coordinates are.
- The three squirrels are located (in game world coordinates) at
  - (-384, -84),
  - (384, 306), and
  - (585, -234).
- We can only display 640 x 480 pixel area on the screen, so we need to track where the camera origin is located in game world coordinates.
- In the picture, the camera origin is placed at (-486, -330) in game world coordinates.

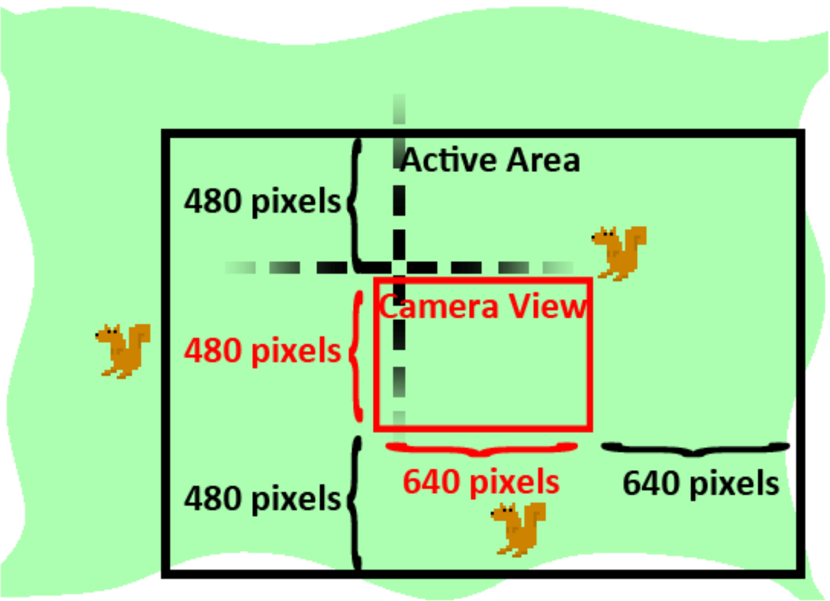


Cameras



- The picture shows the same field and squirrels in camera coordinates.
- The area that the camera can see (the camera view) has its origin at the game world coordinates (-486, -330).
- Since what the camera sees is displayed on the player's screen, the —camera coordinates are the same as the —pixel coordinates.
- To find out the pixel coordinates of the squirrels, take the game coordinates of the squirrel and subtract the game coordinates of the camera's origin.
- So the squirrel on the left has game world coordinates of (-384, -84) but appears at (102, 246) on the screen in pixel coordinates.
  - For the X coordinate, -384 - -486 = 102 and
  - for the Y coordinate, -84 - -330 = 246.
- When we do the same calculation to find the pixel coordinates of the other two squirrels, we find that they exist outside of the range of the screen.
- This is why they don't appear in the camera's view.

The "Active Area"



- The —active areall is just a name used to describe the area of the game world that the camera views plus an area around it the size of the camera area
- When we create new enemy squirrel or grass objects, we don't want them to be created inside the view of the camera, since it'll appear that they just pop out of nowhere.
  - But we also don't want to create them too far away from the camera, because then they may never wander into the camera's view.
  - Inside the active area but outside the camera is where squirrel and grass objects can safely be created.
- When squirrel and grass objects are beyond the border of the active area then they are far away enough to delete so that they don't take up memory any more.
  - Objects that far away aren't needed since it is much less likely that they'll come back into view of the camera.

Keeping Track of the Location of Things in the Game World

```
grassObjs = [] # stores all the grass objects in the game
squirrelObjs = [] # stores all the non-player squirrel objects
# stores the player object:
playerObj = {'surface': pygame.transform.scale(L_SQUIR_IMG, (STARTSIZE, STARTSIZE)),
'facing': LEFT,
'size': STARTSIZE,
'x': HALF_WINWIDTH,
'y': HALF_WINHEIGHT,
'bounce':0,
'health': MAXHEALTH}

moveLeft = False
moveRight = False
moveUp = False
moveDown = False
```

Starting Off with Some Grass 5

```
# start off with some random grass images on the screen
for i in range(10):
    grassObjs.append(makeNewGrass(camerax, cameray))
    grassObjs[i]['x'] = random.randint(0, WINWIDTH)
    grassObjs[i]['y'] = random.randint(0, WINHEIGHT)
```

- The active area should start off with a few grass objects visible on the screen.
  - The makeNewGrass() function creates and returns a grass object randomly located somewhere in the active area but outside the camera view.
  - This is what we normally want when we call makeNewGrass(), but since we want to make sure the first few grass objects are on the screen, the X and Y coordinates are overwritten.

The Game Loop

131. while True: # main game loop

- The game loop does
  - event handling,
  - updating the game state, and
  - drawing everything to the screen.

Checking to Disable Invulnerability

```
132. # Check if we should turn off invulnerability
133. if invulnerableMode and time.time() - invulnerableStartTime > INVULNTIME:
134. invulnerableMode = False
```

- When the player gets hit by an enemy squirrel and does not die, we make the player invulnerable for a couple seconds (INVULNTIME = 2).
- During this time, the player's squirrel flashes and the won't take any damage from other squirrels.

Moving the Enemy Squirrels

```
# move all the squirrels
for sObj in squirrelObjs:
# move the squirrel, and adjust for their bounce
sObj['x'] += sObj['movex']
sObj['y'] += sObj['movey']
```

- The enemy squirrels all move according to the values in their 'movex' and 'movey' keys.
  - If these values are positive, the squirrels move right or down.
  - If these values are negative, they move left or up.
- The larger the value, the farther they move on each iteration through the game loop (which means they move faster).

Moving the Enemy Squirrels

```
sObj['bounce'] += 1
if sObj['bounce'] > sObj['bouncerate']:
sObj['bounce'] = 0 # reset bounce amount
```

- The value in sObj['bounce'] is incremented on each iteration of the game loop for each squirrel.
  - When this value is 0, the squirrel is at the very beginning of its bounce.
  - When this value is equal to the value in sObj['bouncerate'] the value is at its end.
  - This is why a smaller sObj['bouncerate'] value makes for a faster bounce.
  - If sObj['bouncerate'] is 3, then it only takes three iterations through the game loop for the squirrel to do a full bounce.
  - When sObj['bounce'] gets larger than sObj['bouncerate'], then it needs to be reset to 0.

Moving the Enemy Squirrels


```
145. # random chance they change direction
146. if random.randint(0, 99) < DIRCHANGEFREQ:
147. sObj['movex'] = getRandomVelocity()
148. sObj['movey'] = getRandomVelocity()
149. if sObj['movex'] > 0: # faces right
150. sObj['surface'] = pygame.transform.scale(R_SQUIR_IMG, (sObj['width'], sObj['height']))
151. else: # faces left
152. sObj['surface'] = pygame.transform.scale(L_SQUIR_IMG, (sObj['width'], sObj['height']))
```

- There is a 2% chance on each iteration through the game loop that the squirrel will randomly change speed and direction.
  - On line 146 the random.randint(0, 99) call randomly selects an integer out of 100 possible integers.
  - If this number is less than DIRCHANGEFREQ (which we set to 2) then a new value will be set for sObj['movex'] and sObj['movey'].
  - Because this means the squirrel might have changed direction, the Surface object in sObj['surface'] should be replaced by a new one that is properly facing left or right and scaled to the squirrel’s size.

Removing the Far Away Grass and Squirrel Objects

```
# go through all the objects and see if any need to be deleted.
for i in range(len(grassObjs) - 1, -1, -1):
if isOutsideActiveArea(camerax, cameray, grassObjs[i]):
del grassObjs[i]
for i in range(len(squirrelObjs) - 1, -1, -1):
if isOutsideActiveArea(camerax, cameray, squirrelObjs[i]):
del squirrelObjs[i]
```

- During each iteration of the game loop, the code will check all grass and enemy squirrel objects to see if they are outside the — **active area.**
  - The isOutsideActiveArea() function takes the current coordinates of the camera and the grass/enemy squirrel object, and returns True if the object is not located in the active area.
- Deleting squirrel and grass objects is done with the del operator.
- This ensures that there is always a number of squirrels and grass objects near the player.

Python staff

- The for loop pass arguments to the range() function so that the numbering starts at the index of the last item and then decrements by -1 until it reaches the number -1.
  - We are iterating backwards over the list's indexes compared to how it is normally done.
  - This is done because we are iterating over the list that we are also deleting items from.
- To see why this reverse order is needed, say we had the following code:

```
animals = ['cat', 'mouse', 'dog', 'horse’]
for i in range(len(animals)):
        if animals[i] == 'dog':
                del animals[i]
```

- But if we ran this code, we would get an IndexError error that looks like this:

```
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
IndexError: list index out of range
```

Python staff

- Why this error happens?
  - First, the animals list = ['cat', 'mouse', 'dog', 'horse'] and len(animals) = 4.
  - This means that the call to range(4) would cause the for loop to iterate with the values 0, 1, 2, and 3.
  - When the for loop iterates with i set to 2, the if statement's condition will be True and the del animals[i] statement will delete animals[2].
  - This means that afterwards the animals list will be ['cat', 'mouse', 'horse'].
  - The indexes of all items after 'dog' are all shifted down by one because the 'dog' value was removed.
  - But on the next iteration through the for loop, i is set to 3.
  - But animals[3] is out of bounds because the valid indexes of the animals list is no longer 0 to 3 but 0 to 2.
  - The original call to range() was for a list with 4 items in it.
  - **The list changed in length, but the for loop is set up for the original length. (Python staff)**

Python staff

- However, if we iterate from the last index of the list to 0, we don’t run into this problem.
- The following program deletes the 'dog' string from the animals list without causing an IndexError error:

```
animals = ['cat', 'mouse', 'dog', 'horse']
for i in range(len(animals) - 1, -1, -1):
        if animals[i] == 'dog':
                del animals[i]
```

- The reason this code doesn’t cause an error is because the for loop iterates over 3, 2, 1, and 0.
- On the first iteration, the code checks if animals[3] is equal to 'dog’ and because it isn’t (animals[3] is 'horse') so the code moves on to the next iteration.
- Then animals[2] is checked if it equals 'dog’, so animals[2] is deleted.
- After animals[2] is deleted, the animals list is set to ['cat', 'mouse', 'horse'].
- On the next iteration, i is set to 1. There is a value at animals[1] (the 'mouse' value), so no error is caused.

Adding New Grass and Squirrel Objects

```
# add more grass & squirrels if we don't have enough.
while len(grassObjs) < NUMGRASS:
grassObjs.append(makeNewGrass(camerax, cameray))
while len(squirrelObjs) < NUMSQUIRRELS:
squirrelObjs.append(makeNewSquirrel(camerax, cameray))
```

- The NUMGRASS constant was set to 80 and the NUMSQUIRRELS constant was set to 30 at the beginning of the program.
  - These variables are set so that we can be sure there are always plenty of grass and squirrel objects in the active area all time.
  - If the length of the grassObjs or squirrelObjs drops below NUMGRASS or NUMSQUIRRELS respectively, then new grass and squirrel objects are created.

Camera Slack, and Moving the Camera View

7

```
# adjust camerax and cameray if beyond the "camera slack"
playerCenterx = playerObj['x'] + int(playerObj['size'] / 2)
playerCentery = playerObj['y'] + int(playerObj['size'] / 2)
if (camerax + HALF_WINWIDTH) - playerCenterx > CAMERASLACK:
camerax = playerCenterx + CAMERASLACK - HALF_WINWIDTH
elif playerCenterx – (camerax + HALF_WINWIDTH) > CAMERASLACK:
camerax = playerCenterx – CAMERASLACK - HALF_WINWIDTH
if (cameray + HALF_WINHEIGHT) - playerCentery > CAMERASLACK:
cameray = playerCentery + CAMERASLACK - HALF_WINHEIGHT
elif playerCentery – (cameray + HALF_WINHEIGHT) > CAMERASLACK:
cameray = playerCentery – CAMERASLACK - HALF_WINHEIGHT
```

- The camera's position needs to be updated when the player moves over.
- I've called the number of pixels the player can move before the camera gets updated the —camera slack.
- The CAMERASLACK constant is set to 90, which our program will take to mean that the player squirrel can move 90 pixels from the center before the camera position gets updated to follow the squirrel.
- (camerax + HALF_WINWIDTH) and (cameray + HALF_WINHEIGHT) are the XY game world coordinates currently at the center of the screen.
- The playerCenterx and playerCentery is set to the middle of the player's squirrel's position, also in game world coordinates.

Drawing the Background

```
181. # draw the green background
182. DISPLAYSURF.fill(GRASSCOLOR)
```

- A green color for the background will be painted over all of the previous contents of the Surface so that we can start drawing the frame from scratch.

Drawing the Grass

```
# draw all the grass objects on the screen
for gObj in grassObjs:
    gRect = pygame.Rect( (gObj['x'] - camerax,
    gObj['y'] - cameray,
    gObj['width'],
    gObj['height']) )
    DISPLAYSURF.blit(GRASSIMAGES[gObj['grassImage']], gRect)
```

- The Rect object is stored in a variable named gRect.
  - gRect is used in the blit() method call to draw the grass image on the display Surface.
- gObj['grassImage'] only contains an integer that is an index to GRASSIMAGES.
  - GRASSIMAGES is a list of Surface objects that contain all grass images.
  - Surface objects take up much more memory than just a single integer, and all grass objects with similar gObj['grassImage'] values look identical.
  - So it makes sense to only have each grass image stored once in GRASSIMAGES and simply store integers in the grass objects themselves.

Drawing the Squirrels

```
# draw the other squirrels
for sObj in squirrelObjs:
 sObj['rect'] = pygame.Rect( (sObj['x'] - camerax,
    sObj['y'] - cameray - getBounceAmount(sObj['bounce'], sObj['bouncerate'], sObj['bounceheight']),
    sObj['width'],
    sObj['height']) )
    DISPLAYSURF.blit(sObj['surface'], sObj['rect'])
```

- The for loop that draws all enemy squirrel game objects is similar to the previous for loop, except that the Rect object it creates is saved in the 'rect' key’s value of the squirrel dictionary.
  - The reason the code does this is because we will use this Rect object later to check if the enemy squirrels have collided with the player squirrel.
  - The top parameter for the Rect constructor is not just sObj['y'] - cameray but sObj['y'] - cameray - getBounceAmount(sObj['bounce'], sObj['bouncerate'], sObj['bounceheight']).
  - The getBounceAmount() function will return the number of pixels that the top value should be raised.
- There is no common list of Surface objects of the squirrel images
  - Each enemy squirrel game object has its own Surface object stored in the 'surface' key.
  - This is because the squirrel images can be scaled to different sizes.

Drawing the Squirrels and Health Meter

8

```
202. # draw the player squirrel
203. flashIsOn = round(time.time(), 1) * 10 % 2 == 1
```

- After drawing the grass and enemy squirrels, the code will draw the player's squirrel.
- However, there is one case where we would skip drawing the player's squirrel.
- When the player collides with a larger enemy squirrel, the player takes damage and flashes for a little bit to indicate that the player is temporarily invulnerable.
  - This flashing effect is done by drawing the player squirrel on some iterations through the game loop but not on others.
  - The player squirrel will be drawn on game loop iterations for a tenth of a second, and then not drawn on the game loop iterations for a tenth of second.
  - This repeats over and over again as long as the player is invulnerable (which, in the code, means that the invulnerableMode variable is set to True).
  - Our code will make the flashing last for two seconds, since 2 was stored in the INVULNTIME constant.

Drawing the Squirrels and Health Meter

```
if not gameOverMode and not (invulnerableMode and flashIsOn):
    playerObj['rect'] = pygame.Rect( (playerObj['x'] - camerax,
        playerObj['y'] – cameray - getBounceAmount(playerObj['bounce'], BOUNCERATE, BOUNCEHEIGHT),
        playerObj['size'],
        playerObj['size']) )
        DISPLAYSURF.blit(playerObj['surface'], playerObj['rect'])


# draw the health meter
drawHealthMeter(playerObj['health'])
```

- The drawHealthMeter() function draws the indicator at the top left corner of the screen that tells the player how many times the player squirrel can be hit before dying.

The Event Handling Loop

- 215. `for event in pygame.event.get(): # event handling loop`
- 216. `    if event.type == QUIT:`
- 217. `        terminate()`

- The first thing that is checked in the event handling loop is if the QUIT event has been generated.
- If so, then the program should be terminated.

The Event Handling Loop

```
elif event.type == KEYDOWN:
    if event.key in (K_UP, K_w):
        moveDown = False
        moveUp = True
    elif event.key in (K_DOWN, K_s):
        moveUp = False
        moveDown = True
```

- If the up or down arrow keys have been pressed (or their WASD equivalents), then the move variable for that direction should be set to True and the move variable for the opposite direction should be set to False.

The Event Handling Loop

```
            elif event.key in (K_LEFT, K_a):
                moveRight = False
                moveLeft = True
                if playerObj['facing'] == RIGHT: # change player image
                    playerObj['surface'] = pygame.transform.scale(L_SQUIR_IMG, (playerObj['size'], playerObj['size']))
                playerObj['facing'] = LEFT
            elif event.key in (K_RIGHT, K_d):
                moveLeft = False
                moveRight = True
            if playerObj['facing'] == LEFT: # change player image
               playerObj['surface'] = pygame.transform.scale(R_SQUIR_IMG, (playerObj['size'], playerObj['size']))
               playerObj['facing'] = RIGHT
            elif winMode and event.key == K_r:
                return
```

The Event Handling Loop

```
        elif event.type == KEYUP:
            # stop moving the player's squirrel
            if event.key in (K_LEFT, K_a):
                moveLeft = False
            elif event.key in (K_RIGHT, K_d):
                moveRight = False
            elif event.key in (K_UP, K_w):
                moveUp = False
            elif event.key in (K_DOWN, K_s):
                moveDown = False
            elif event.key == K_ESCAPE:
                terminate()
```

Moving the Player, and Accounting for Bounce

```
if not gameOverMode:
    # actually move the player
    if moveLeft:
        playerObj['x'] -= MOVERATE
    if moveRight:
        playerObj['x'] += MOVERATE
    if moveUp:
        playerObj['y'] -= MOVERATE
    if moveDown:
        playerObj['y'] += MOVERATE
    if (moveLeft or moveRight or moveUp or moveDown) or playerObj['bounce'] != 0:
        playerObj['bounce'] += 1
```

Moving the Player, and Accounting for Bounce

```
        if playerObj['bounce'] > BOUNCERATE:
            playerObj['bounce'] = 0 # reset bounce amount
```

- The value in playerObj['bounce'] keeps track of at what point in bouncing the player is at.
  - This variable stores an integer value from 0 to BOUNCERATE.
  - A playerObj['bounce'] value of 0 means the player squirrel is at the start of a bounce and a value of BOUNCERATE means the player squirrel is at the end of the bounce.
  - The player squirrel will bounce whenever the player is moving, or if the player has stopped moving but the squirrel hasn’t finished its current bounce.
- If any of the move variables is set to True or the current playerObj['bounce'] is not 0 (which means the player is currently in a bounce), then the variable should be incremented.
  - Because the playerObj['bounce'] variable should only be in the range of 0 to BOUNCERATE, if incrementing it makes it larger than BOUNCERATE, it should be reset back to 0.

Collision Detection: Eat or Be Eaten


```
        # check if the player has collided with any squirrels
        for i in range(len(squirrelObjs)-1, -1, -1):
            sqObj = squirrelObjs[i]
            if 'rect' in sqObj and playerObj['rect'].colliderect(sqObj['rect']):
                # a player/squirrel collision has occurred

                if sqObj['width'] * sqObj['height'] <= playerObj['size']**2:
                    # player is larger and eats the squirrel
                    playerObj['size'] += int( (sqObj['width'] * sqObj['height'])**0.2 ) + 1
                    del squirrelObjs[i]
```

- If the player’s squirrel is equal or larger than the size of the enemy squirrel it has collided with, then the player’s squirrel will eat that squirrel and grow.
- The number that is added to the 'size' key in the player object is calculated based on the enemy squirrel’s size.
- The larger squirrels cause more growth.
  - According to the chart, eating a squirrel that has a width and height of 45 (that is, an area of 1600 pixels) would cause the player to grow 5 pixels wider and taller.

Collision Detection: Eat or Be Eaten

```
        if playerObj['facing'] == LEFT:
            playerObj['surface'] = pygame.transform.scale(L_SQUIR_IMG, (playerObj['size'], playerObj['size']))
        if playerObj['facing'] == RIGHT:
            playerObj['surface'] = pygame.transform.scale(R_SQUIR_IMG, (playerObj['size'], playerObj['size']))
```

- The player’s squirrel image needs to be updated now that the squirrel is larger.
  - This can be done by passing the original squirrel image in L_SQUIR_IMG or R_SQUIR_IMG to the pygame.transform.scale() function, which will return an enlarged version of the image.
  - Depending on whether playerObj['facing'] is equal to LEFT or RIGHT determines which original squirrel image we pass to the function.

Collision Detection: Eat or Be Eaten

```
        if playerObj['size'] > WINSIZE:
            winMode = True # turn on "win mode"
    elif not invulnerableMode:
        # player is smaller and takes damage
        invulnerableMode = True
        invulnerableStartTime = time.time()
        playerObj['health'] -= 1
        if playerObj['health'] == 0:
            gameOverMode = True # turn on "game over mode"
            gameOverStartTime = time.time()
```

- The way the player wins the game is by getting the squirrel to have a size larger than the integer stored in the WINSIZE constant variable and the winMode variable is set to True.
- If the player’s area was not equal to or larger than the area of the enemy squirrel, and invulnerableMode was not set to True, then the player will take damage from colliding with this larger squirrel.
- To prevent the player from being damaged several times by the same squirrel immediately, we will briefly make the player invulnerable to further squirrel attacks by setting invulnerableMode to True.

The Game Over Screen

```
    else:
        # game is over, show "game over" text
        DISPLAYSURF.blit(gameOverSurf, gameOverRect)
        if time.time() - gameOverStartTime > GAMEOVERTIME:
            return # end the current game
```

- When the player has died, the “Game Over” text will be shown on the screen for the number of seconds that is in the GAMEOVERTIME constant.
  - Once this amount of time has elapsed, then the runGame() function will return.
  - This lets the enemy squirrels continue to be animated and moving around for a few seconds after the player dies and before the next game starts.
  - The “game over screen” in Squirrel Eat Squirrel does not wait until the player presses a key before a new game starts.

Winning

```
# check if the player has won.
if winMode:
    DISPLAYSURF.blit(winSurf, winRect)
    DISPLAYSURF.blit(winSurf2, winRect2)

pygame.display.update()
FPSCLOCK.tick(FPS)
```

Drawing a Graphical Health Meter

```
def drawHealthMeter(currentHealth):
    for i in range(currentHealth): # draw red health bars
        pygame.draw.rect(DISPLAYSURF, RED,   (15, 5 + (10 * MAXHEALTH) - i * 10, 20, 10))
    for i in range(MAXHEALTH): # draw the white outlines
        pygame.draw.rect(DISPLAYSURF, WHITE, (15, 5 + (10 * MAXHEALTH) - i * 10, 20, 10), 1)
```

The Same Old terminate() Function

```
def terminate():
    pygame.quit()
    sys.exit()
```

- The terminate() function works the same as in the previous game programs.

The Mathematics of the Sine Function

```
def getBounceAmount(currentBounce, bounceRate, bounceHeight):
    # Returns the number of pixels to offset based on the bounce.
    # Larger bounceRate means a slower bounce.
    # Larger bounceHeight means a higher bounce.
    # currentBounce will always be less than bounceRate
    return int(math.sin( (math.pi / float(bounceRate)) * currentBounce ) * bounceHeight)

```

- If you want to make the bounce higher, than increase the BOUNCEHEIGHT constant.
- If you want to make the bounce slower, than increase the BOUNCERATE constant.

Backwards Compatibility with Python Version 2

- The reason we call float() to convert bounceRate to a floating point number is simply so that this program will work in Python version 2.
- In Python version 3, the division operator will evaluate to a floating point value even if both of the operands are integers, like this:
- >>> # Python version 3
- >>> 10 / 5
- 2.0
- >>> 10 / 4
- 2.5

Backwards Compatibility with Python Version 2

- However, in Python version 2, the / division operator will only evaluate to a floating point value if one of the operands is also a floating point value.
- If both operands are integers, then Python 2’s division operator will evaluate to an integer value (rounding down if needed), like this:
- >>> # Python version 2
- >>> 10 / 5
- 2
- >>> 10 / 4
- 2
- >>> 10 / 4.0
- 2.5
- >>> 10.0 / 4
- 2.5
- >>> 10.0 / 4.0
- 2.5

Backwards Compatibility with Python Version 2

- If we always convert one of the values to a floating point value with the float() function, then the division operator will evaluate to a float value no matter which version of Python runs this source code.
- Making these changes so that our code works with older versions of software is called backwards compatibility.
- It is important to maintain backwards compatibility, because not everyone will always be running the latest version of software and you want to ensure that the code you write works with as many computers as possible.
- **You can't always make your Python 3 code backwards backwards compatible with Python 2, but if it's possible then you should do it.**

The getRandomVelocity() Function

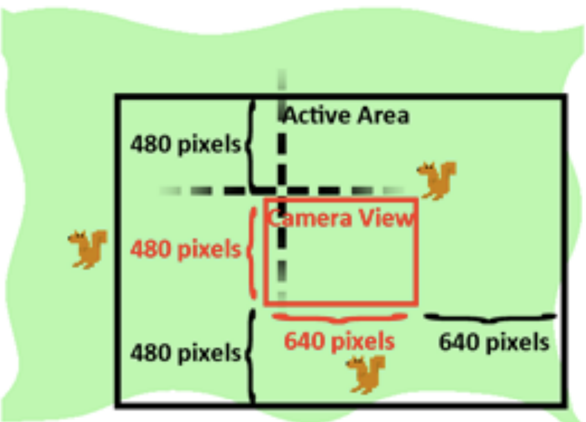
```
def getRandomVelocity():
    speed = random.randint(SQUIRRELMINSPEED, SQUIRRELMAXSPEED)
    if random.randint(0, 1) == 0:
        return speed
    else:
        return -speed
```

9

- The getRandomVelocity() function is used to randomly determine how fast an enemy squirrel will move.
  - The range of this velocity is set in the SQUIRRELMINSPEED and SQUIRRELMAXSPEED constants, but on top of that, the speed is either negative (indicating the squirrel goes to the left or up) or positive (indicating the squirrel goes to the right or down).
  - There is a fifty-fifty chance for the random speed to be positive or negative.

Finding a Place to Add New Squirrels and Grass

```
def getRandomOffCameraPos(camerax, cameray, objWidth, objHeight):
    # create a Rect of the camera view
    cameraRect = pygame.Rect(camerax, cameray, WINWIDTH, WINHEIGHT)
    while True:
        x = random.randint(camerax - WINWIDTH, camerax + (2 * WINWIDTH))
        y = random.randint(cameray - WINHEIGHT, cameray + (2 * WINHEIGHT))
        # create a Rect object with the random coordinates and use colliderect()
        # to make sure the right edge isn't in the camera view.
        objRect = pygame.Rect(x, y, objWidth, objHeight)
        if not objRect.colliderect(cameraRect):
            return x, y
```



Creating Enemy Squirrel Data Structures

```
def makeNewSquirrel(camerax, cameray):
    sq = {}
    generalSize = random.randint(5, 25)
    multiplier = random.randint(1, 3)
    sq['width']  = (generalSize + random.randint(0, 10)) * multiplier
    sq['height'] = (generalSize + random.randint(0, 10)) * multiplier
    sq['x'], sq['y'] = getRandomOffCameraPos(camerax, cameray, sq['width'], sq['height'])
    sq['movex'] = getRandomVelocity()
    sq['movey'] = getRandomVelocity()
```

Creating Enemy Squirrel Data Structures

- Creating enemy squirrel game objects is similar to making the grass game objects.
- The data for each enemy squirrel is also stored in a dictionary.
- The width and height are set to random sizes.
  - The generalSize variable is used so that the width and height of each squirrel aren't too different from each other.
  - Using completely random numbers for width and height could give us very tall and skinny squirrels or very short and wide squirrels.
  - The width and height of the squirrel are this general size with a random number from 0 to 10 added to it (for slight variation), and then multiplied by the multiplier variable.
- The original XY coordinate position of the squirrel will be a random location that the camera cannot see, to prevent the squirrels from just "popping" into existence on the screen.
- The speed and direction are also randomly selected by the getRandomVelocity() function.

Flipping the Squirrel Image

```
if sq['movex'] < 0: # squirrel is facing left
    sq['surface'] = pygame.transform.scale(L_SQUIR_IMG, (sq['width'], sq['height']))
else: # squirrel is facing right
    sq['surface'] = pygame.transform.scale(R_SQUIR_IMG, (sq['width'], sq['height']))
sq['bounce'] = 0
sq['bouncerate'] = random.randint(10, 18)
sq['bounceheight'] = random.randint(10, 50)
return sq
```

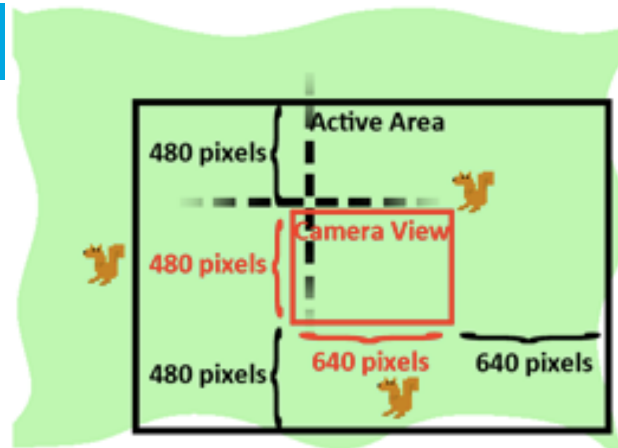
- The L_SQUIR_IMG and R_SQUIR_IMG constants contain Surface objects with left-facing and right-facing squirrel images on them.
- New Surface objects will be made using the pygame.transform.scale() function to match the squirrel’s width and height (stored in sq['width'] and sq['height’] respectively).
- After that, the three bounce-related values are randomly generated (except for sq['bounce'] which is 0 because the squirrel always starts at the beginning of the bounce) and the dictionary is returned

Creating Grass Data Structures

```
def makeNewGrass(camerax, cameray):
    gr = {}
    gr['grassImage'] = random.randint(0, len(GRASSIMAGES) - 1)
    gr['width']  = GRASSIMAGES[0].get_width()
    gr['height'] = GRASSIMAGES[0].get_height()
    gr['x'], gr['y'] = getRandomOffCameraPos(camerax, cameray, gr['width'], gr['height'])
    gr['rect'] = pygame.Rect( (gr['x'], gr['y'], gr['width'], gr['height']) )
    return gr
```

- The grass game objects are dictionaries with the usual 'x', 'y', 'width', 'height', and 'rect' keys but also a 'grassImage' key which is a number from 0 to one less than the length of the GRASSIMAGES list.
- This number will determine what image the grass game object has.
- For example, if the value of the grass object’s 'grassImage' key is 3, then it will se the Surface object stored at GRASSIMAGES[3] for its image.

Checking if Outside the Active Area




```
def isOutsideActiveArea(camerax, cameray, obj):
    # Return False if camerax and cameray are more than
    # a half-window length beyond the edge of the window.
    boundsLeftEdge = camerax - WINWIDTH
    boundsTopEdge = cameray - WINHEIGHT
    boundsRect = pygame.Rect(boundsLeftEdge, boundsTopEdge, WINWIDTH * 3, WINHEIGHT * 3)
    objRect = pygame.Rect(obj['x'], obj['y'], obj['width'], obj['height'])
    return not boundsRect.colliderect(objRect)
```

- The isOutsideActiveArea() will return True if the object you pass it is outside of the "active area" that is dictated by the camerax and cameray parameters.

Final code

- 395. if __name__ == '__main__':
- 396.   main()

- Finally, after all functions have been defined, the program will run the main() function and start the game.

Summary

- Squirrel Eat Squirrel was our first game to have multiple enemies moving around the board at once.
  - The key to having several enemies was using a dictionary value with identical keys for each enemy squirrel, so that the same code could be run on each of them during an iteration through the game loop.

Summary

10

- The concept of the camera was also introduced.
  - Cameras weren't needed for our previous games because the entire game world fit onto one screen.
  - However, when you make your own games that involve a player moving around a large game world, you will need code to handle converting between the game world's coordinate system and the screen's pixel coordinate system.
- Finally, the mathematical sine function was introduced to give realistic squirrel hops (no matter how tall or long each hop was).
  - You don't need to know a lot of math to do programming.
  - In most cases, just knowing addition, multiplication, and negative numbers is fine.
  - However, if you study mathematics, you'll often find several uses for math to make your games cooler.

- For additional programming practice, you can download buggy versions of Squirrel Eat Squirrel from http://invpy.com/buggy/squirrel and try to figure out how to fix the bugs.