Squirrel Game

Frame

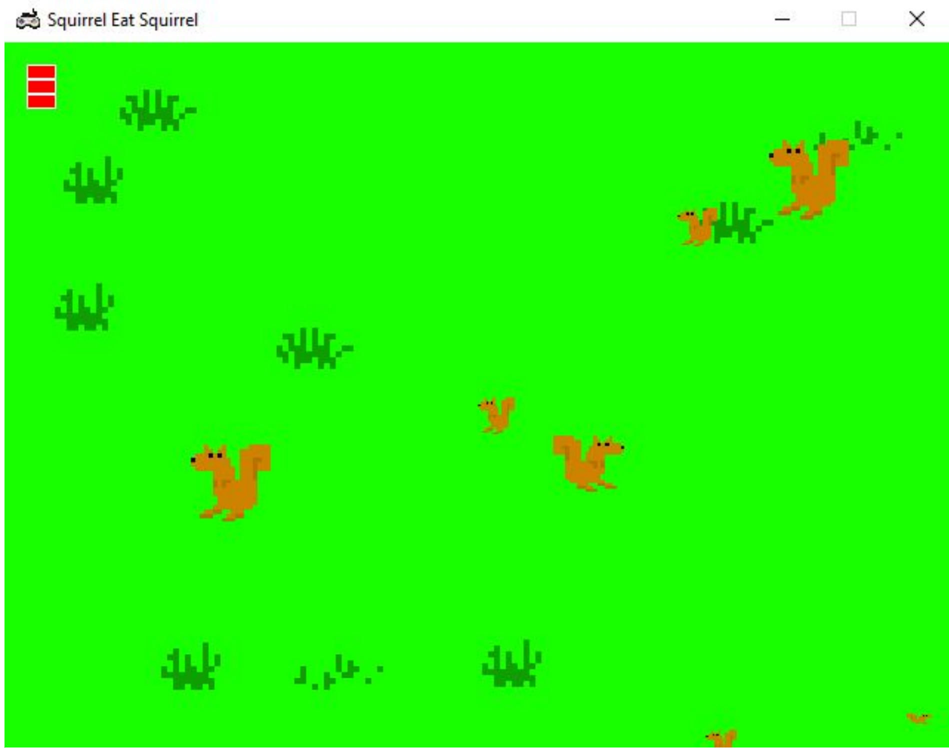
- Icon, title, quit

Objects

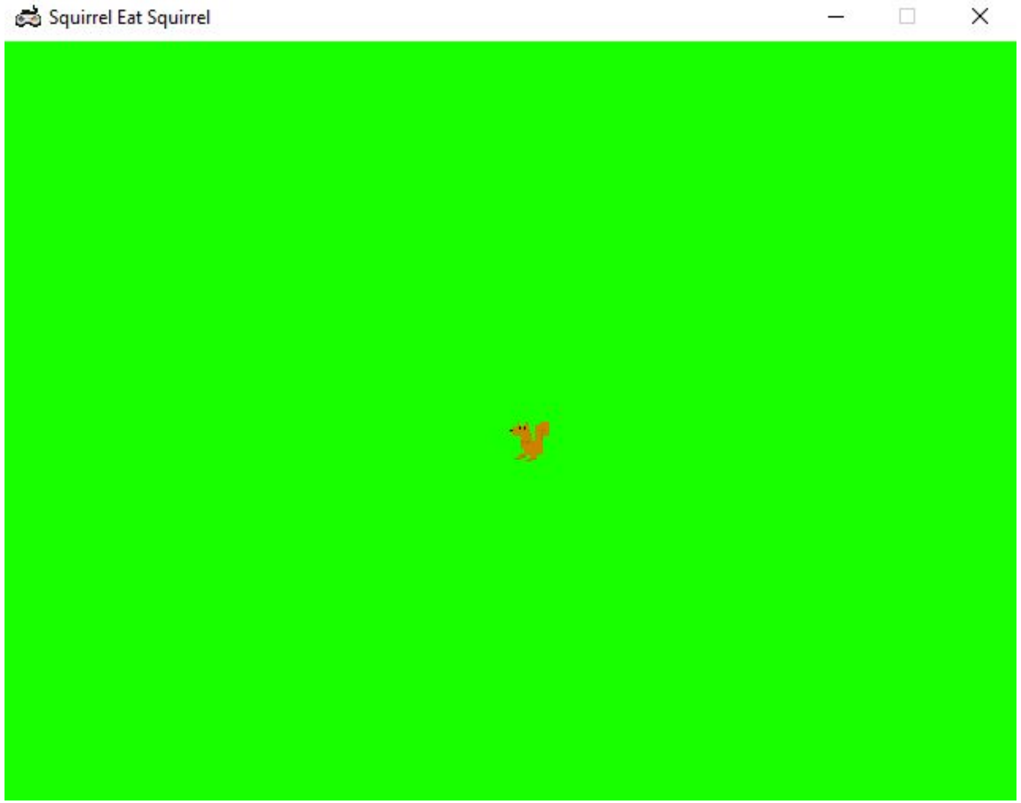
- Player squirrel
- Enemy squirrel
- Grass

Rules

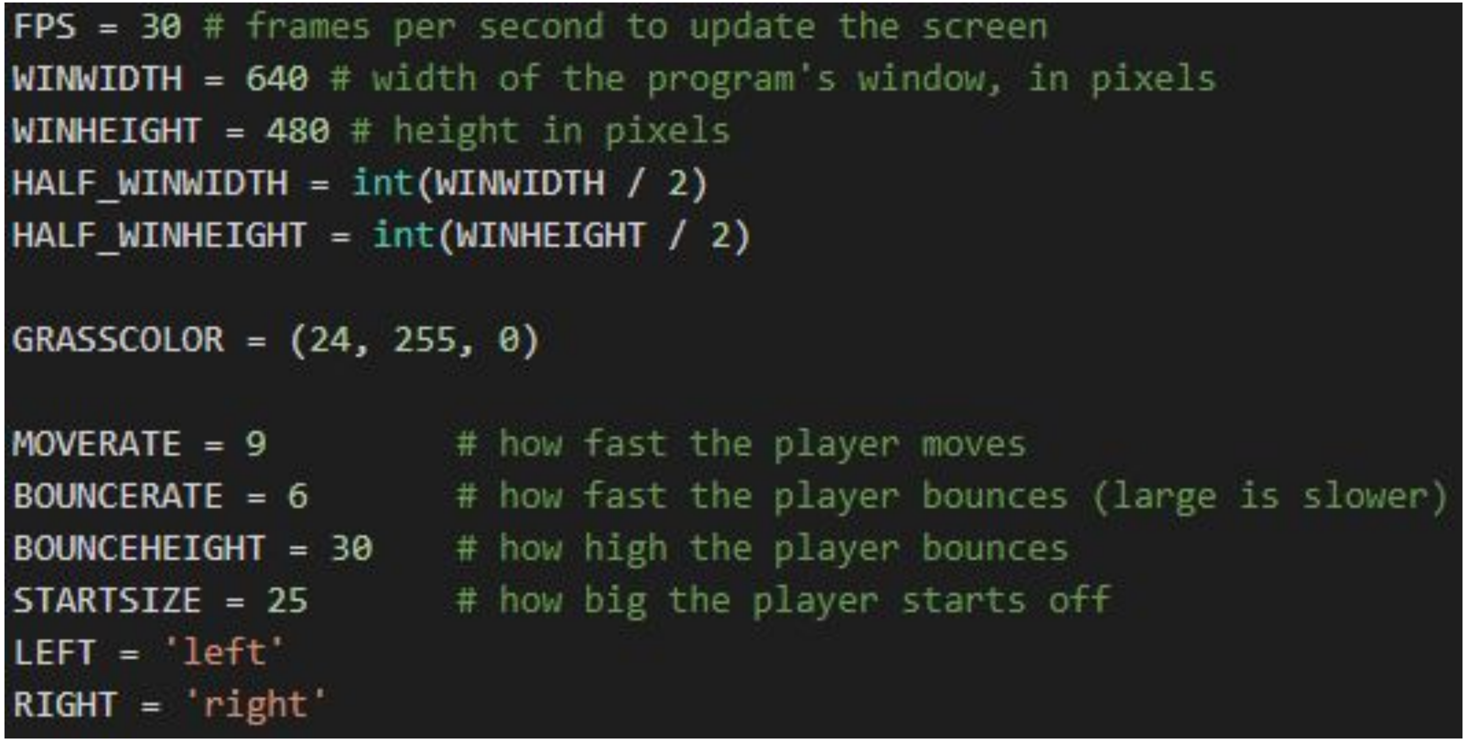
- Eat smaller squirrel to grow
- Hit bigger squirrel to lose health
- Become Omega squirrel to win
- Arrow keys or AWSD keys,
  - Key down to move
  - Key up to stop



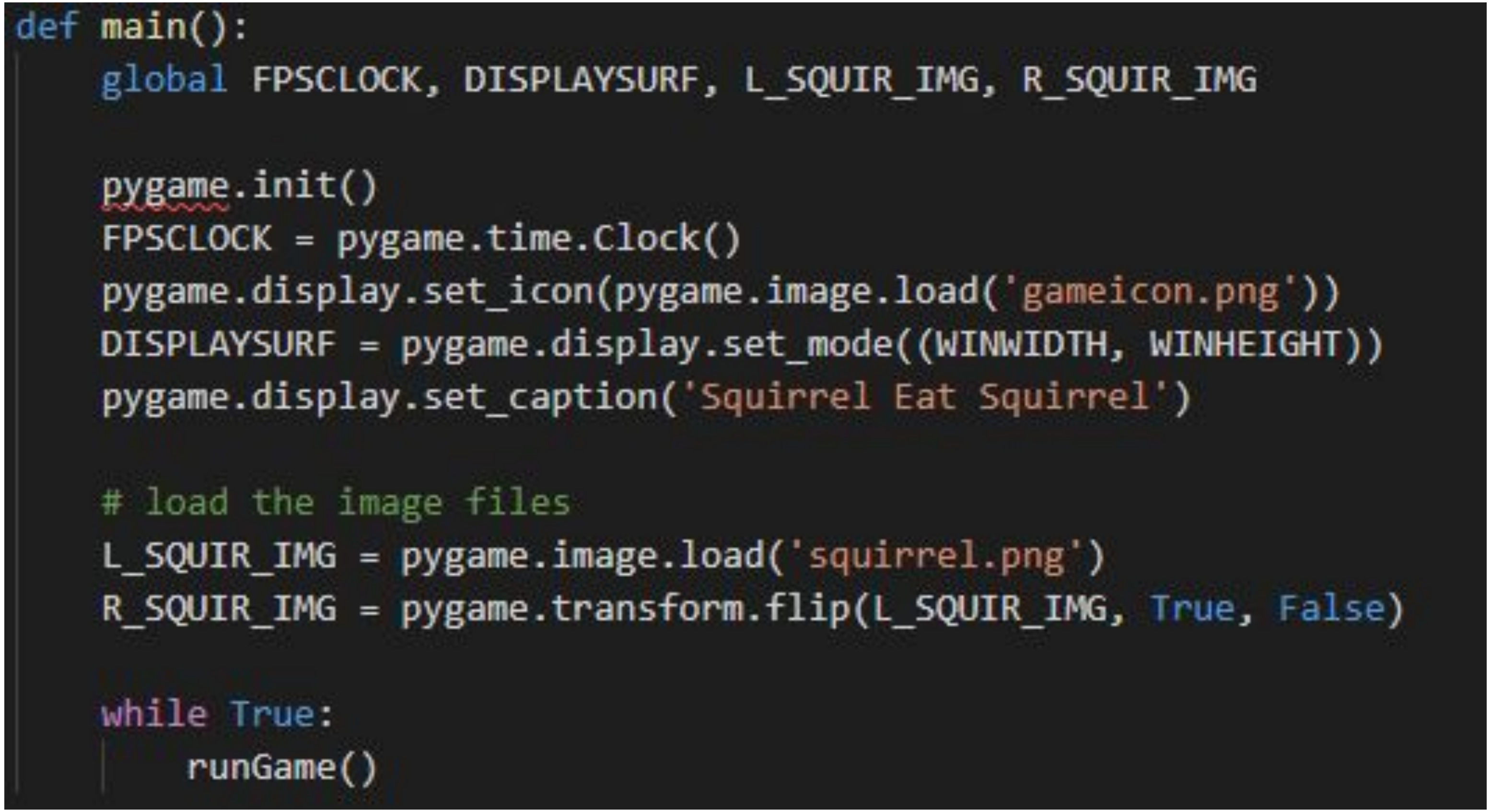
Project 1:



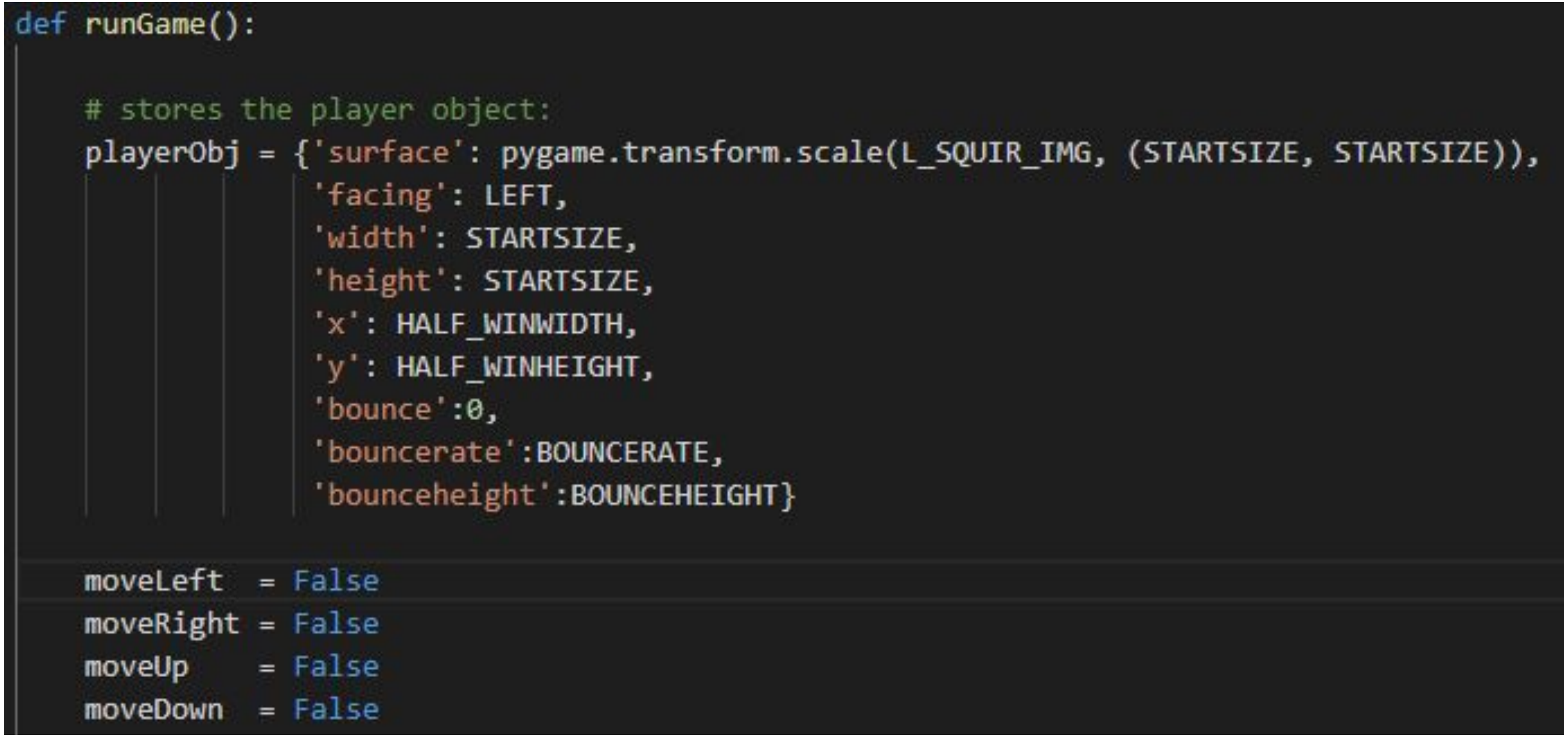
Code



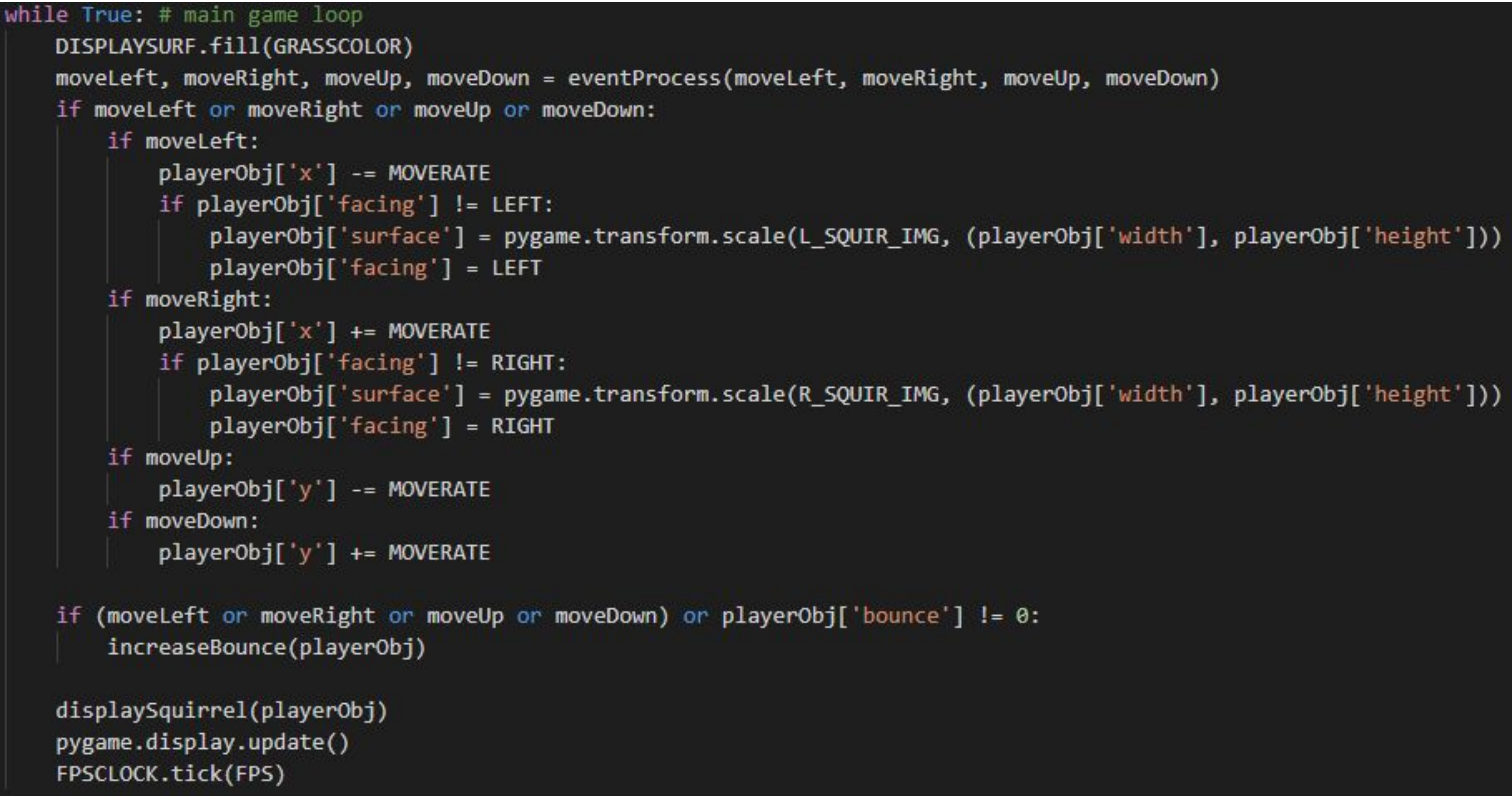
```python
FPS = 30 # frames per second to update the screen
WINWIDTH = 640 # width of the program's window, in pixels
WINHEIGHT = 480 # height in pixels
HALF_WINWIDTH = int(WINWIDTH / 2)
HALF_WINHEIGHT = int(WINHEIGHT / 2)

GRASSCOLOR = (24, 255, 0)

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
LEFT = 'left'
RIGHT = 'right'
```

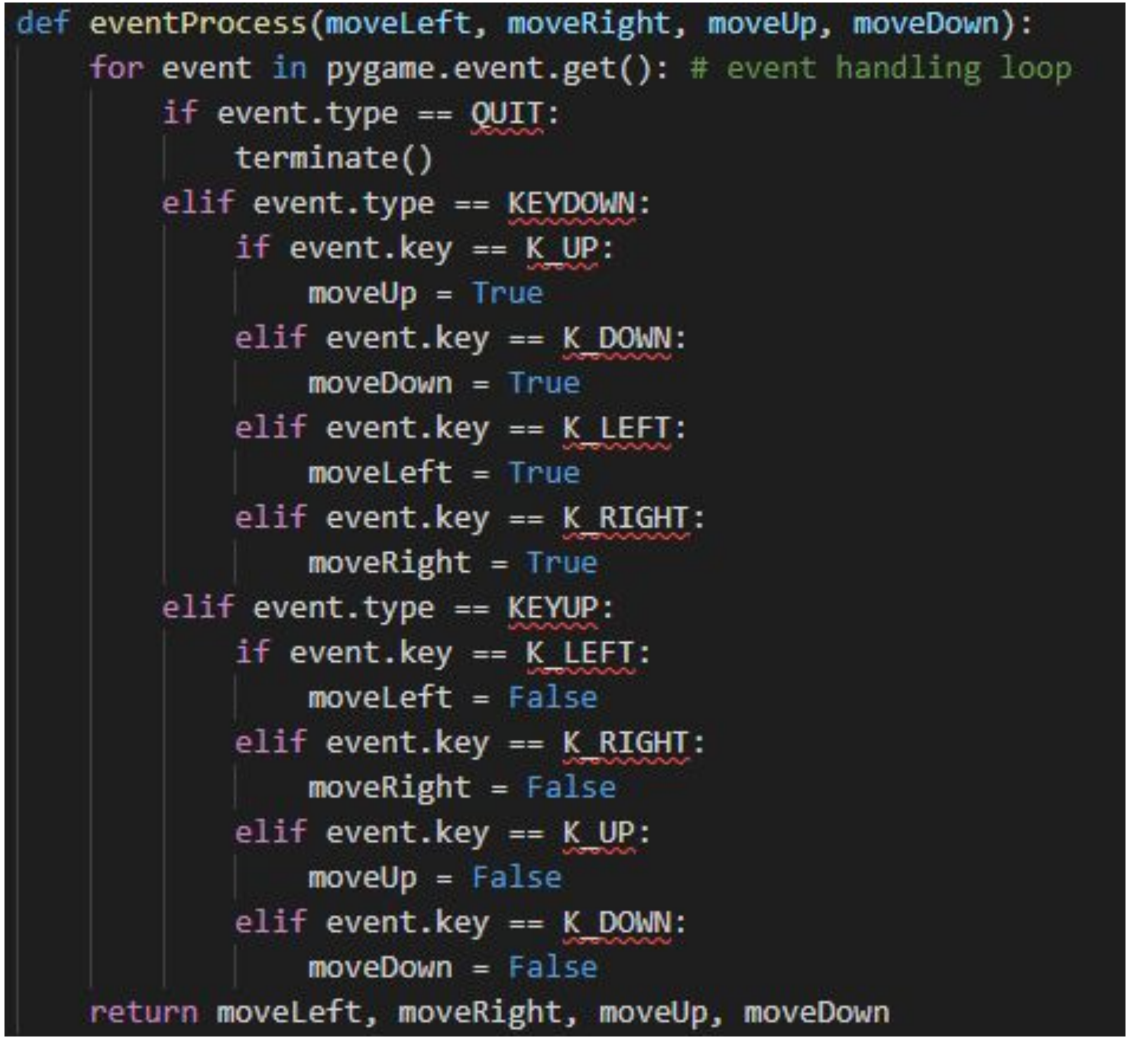


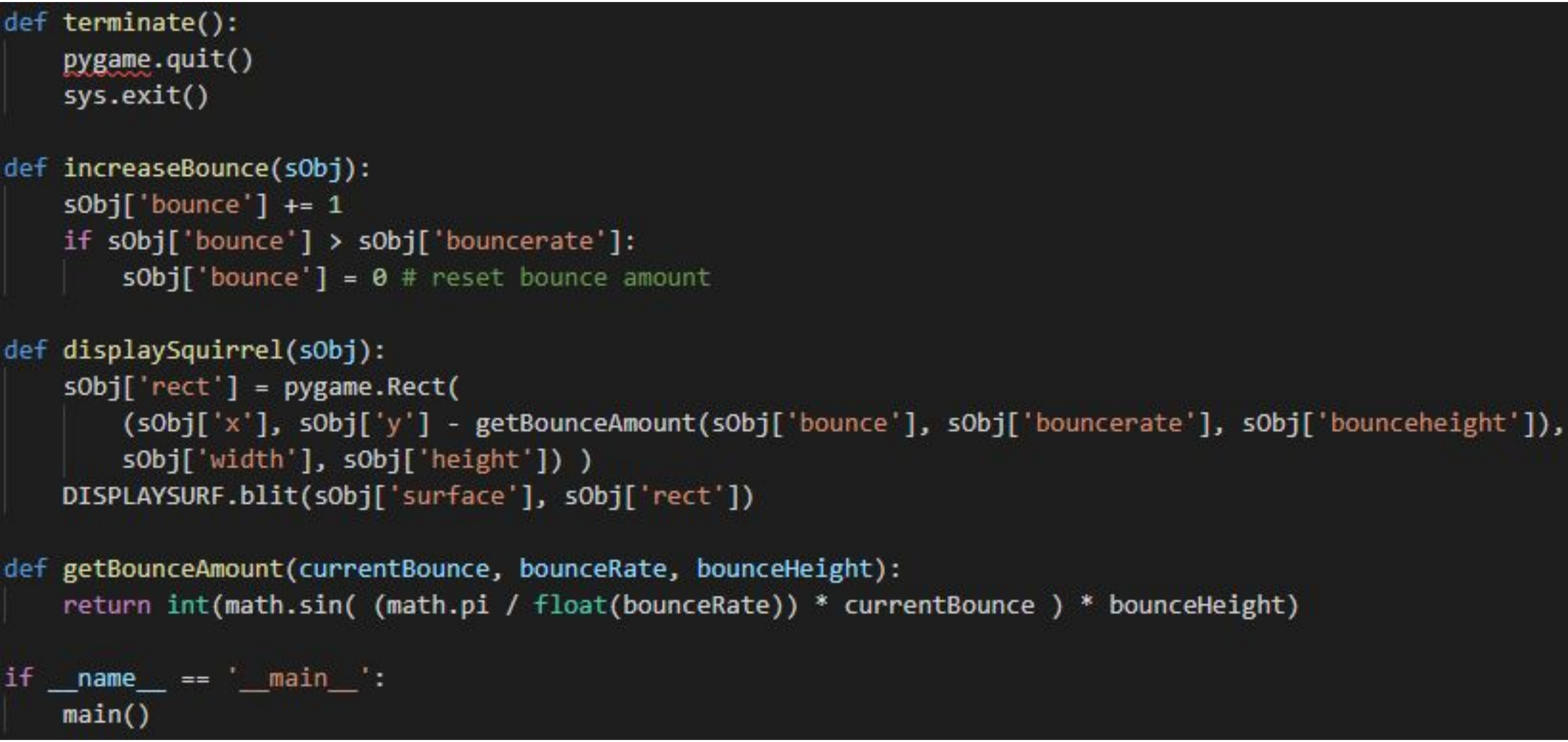
```
def main():
    global FPSCLOCK, DISPLAYSURF, L_SQUIR_IMG, R_SQUIR_IMG

    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')

    # load the image files
    L_SQUIR_IMG = pygame.image.load('squirrel.png')
    R_SQUIR_IMG = pygame.transform.flip(L_SQUIR_IMG, True, False)

    while True:
        runGame()
```



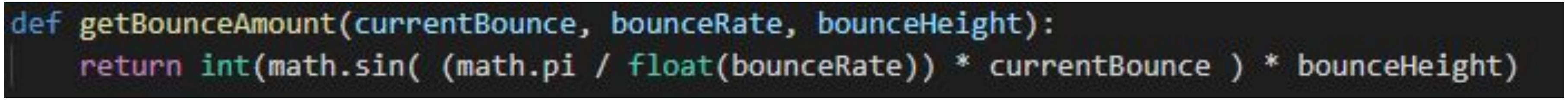
```python
def runGame():

    # stores the player object:
    playerObj = {'surface': pygame.transform.scale(L_SQUIR_IMG, (STARTSIZE, STARTSIZE)),
                 'facing': LEFT,
                 'width': STARTSIZE,
                 'height': STARTSIZE,
                 'x': HALF_WINWIDTH,
                 'y': HALF_WINHEIGHT,
                 'bounce':0,
                 'bouncerate':BOUNCERATE,
                 'bounceheight':BOUNCEHEIGHT}

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


```python
while True: # main game loop
    DISPLAYSURF.fill(GRASSCOLOR)
    moveLeft, moveRight, moveUp, moveDown = eventProcess(moveLeft, moveRight, moveUp, moveDown)
    if moveLeft or moveRight or moveUp or moveDown:
        if moveLeft:
            playerObj['x'] -= MOVERATE
            if playerObj['facing'] != LEFT:
                playerObj['surface'] = pygame.transform.scale(L_SQUIR_IMG, (playerObj['width'], playerObj['height']))
                playerObj['facing'] = LEFT
        if moveRight:
            playerObj['x'] += MOVERATE
            if playerObj['facing'] != RIGHT:
                playerObj['surface'] = pygame.transform.scale(R_SQUIR_IMG, (playerObj['width'], playerObj['height']))
                playerObj['facing'] = RIGHT
        if moveUp:
            playerObj['y'] -= MOVERATE
        if moveDown:
            playerObj['y'] += MOVERATE

    if (moveLeft or moveRight or moveUp or moveDown) or playerObj['bounce'] != 0:
        increaseBounce(playerObj)

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

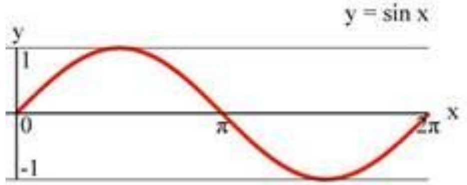

```python
def eventProcess(moveLeft, moveRight, moveUp, moveDown):
    for event in pygame.event.get(): # event handling loop
        if event.type == QUIT:
            terminate()
        elif event.type == KEYDOWN:
            if event.key == K_UP:
                moveUp = True
            elif event.key == K_DOWN:
                moveDown = True
            elif event.key == K_LEFT:
                moveLeft = True
            elif event.key == K_RIGHT:
                moveRight = True
        elif event.type == KEYUP:
            if event.key == K_LEFT:
                moveLeft = False
            elif event.key == K_RIGHT:
                moveRight = False
            elif event.key == K_UP:
                moveUp = False
            elif event.key == K_DOWN:
                moveDown = False
    return moveLeft, moveRight, moveUp, moveDown
```

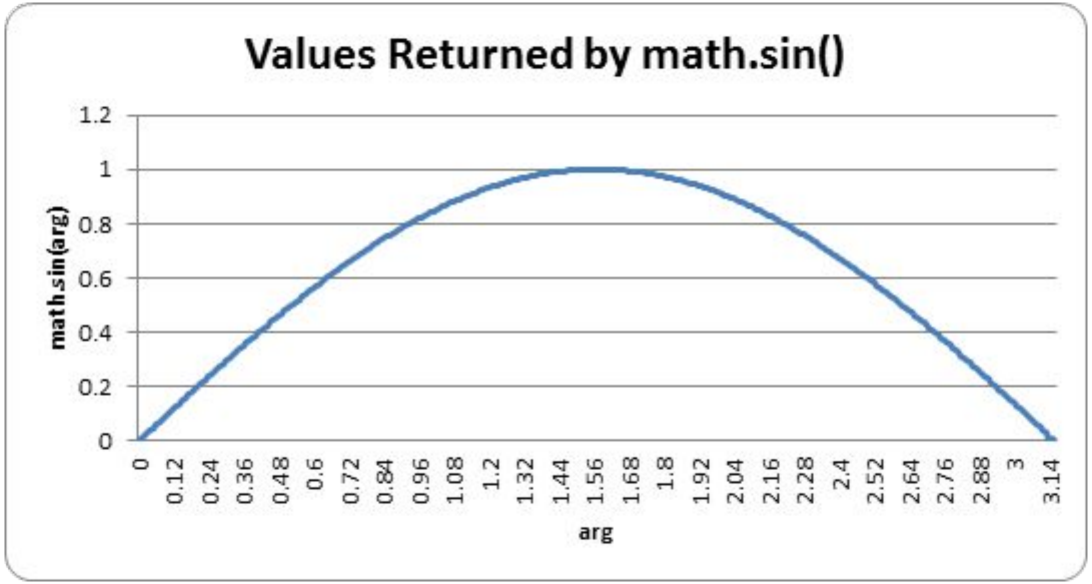


```python
def terminate():
    pygame.quit()
    sys.exit()

def increaseBounce(sObj):
    sObj['bounce'] += 1
    if sObj['bounce'] > sObj['bouncerate']:
        sObj['bounce'] = 0 # reset bounce amount

def displaySquirrel(sObj):
    sObj['rect'] = pygame.Rect(
        (sObj['x'], sObj['y'] - getBounceAmount(sObj['bounce'], sObj['bouncerate'], sObj['bounceheight']),
        sObj['width'], sObj['height']) )
    DISPLAYSURF.blit(sObj['surface'], sObj['rect'])

def getBounceAmount(currentBounce, bounceRate, bounceHeight):
    return int(math.sin( (math.pi / float(bounceRate)) * currentBounce ) * bounceHeight)

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

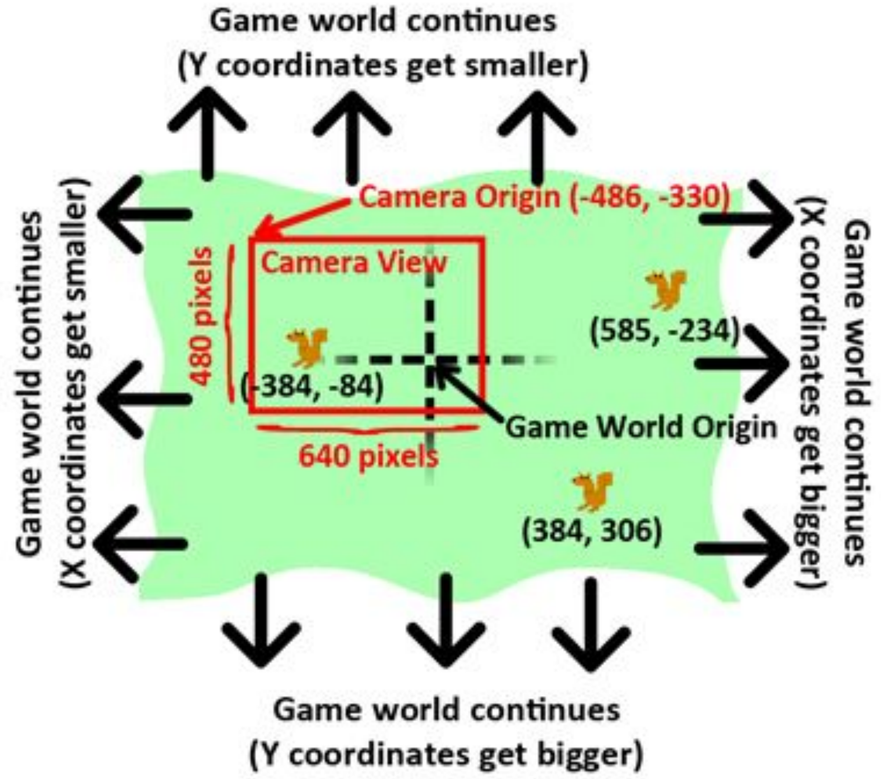


```python
def getBounceAmount(currentBounce, bounceRate, bounceHeight):
    return int(math.sin( (math.pi / float(bounceRate)) * currentBounce ) * bounceHeight)
```



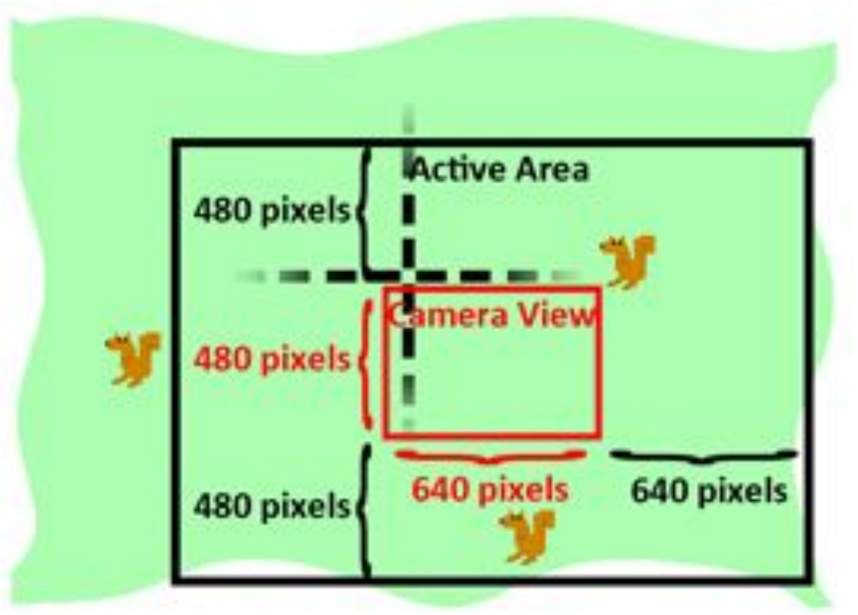


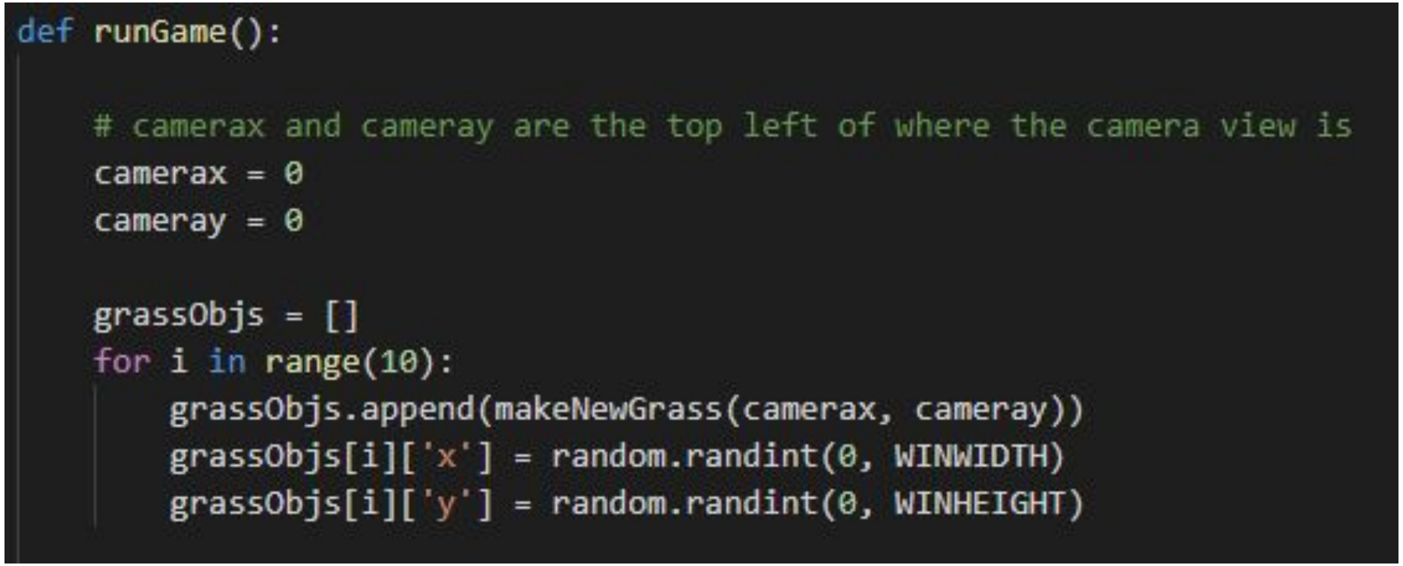
Project 2: Game World and Camera View



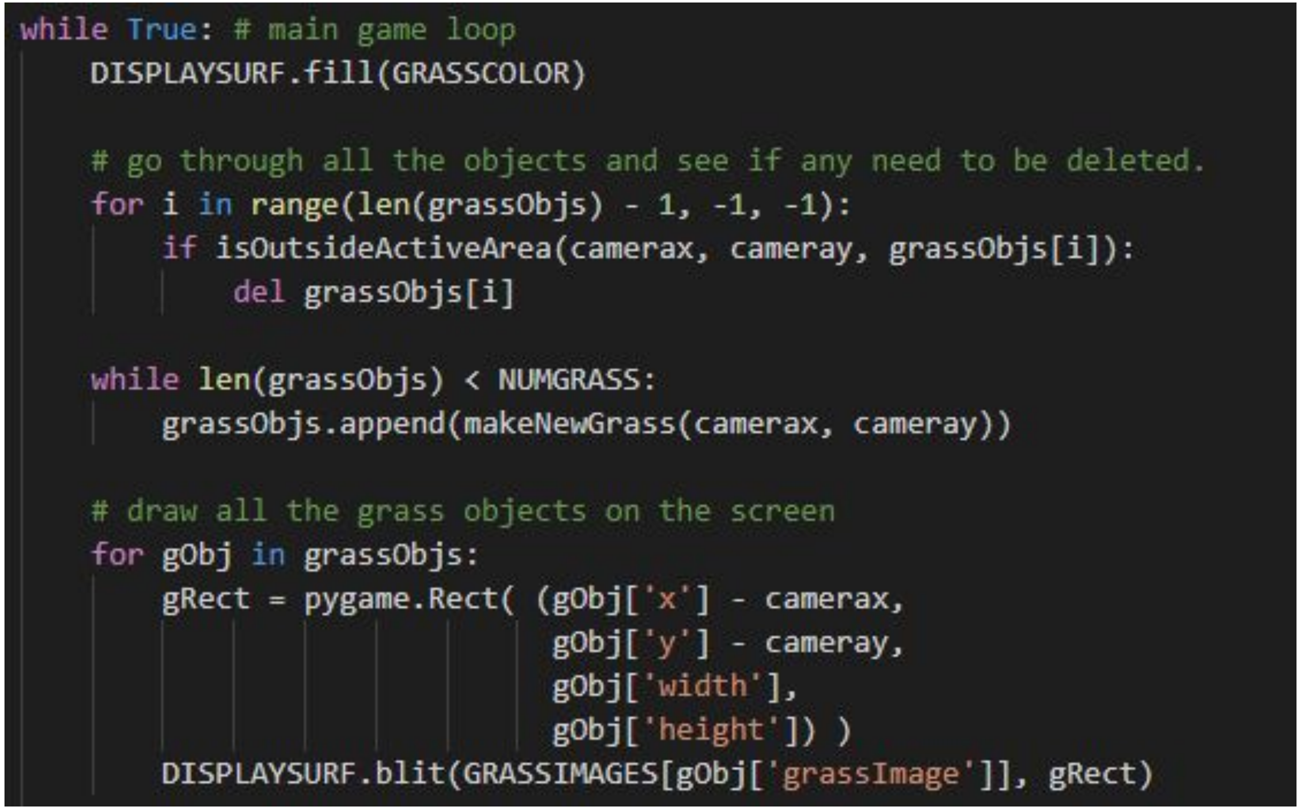
Active Area

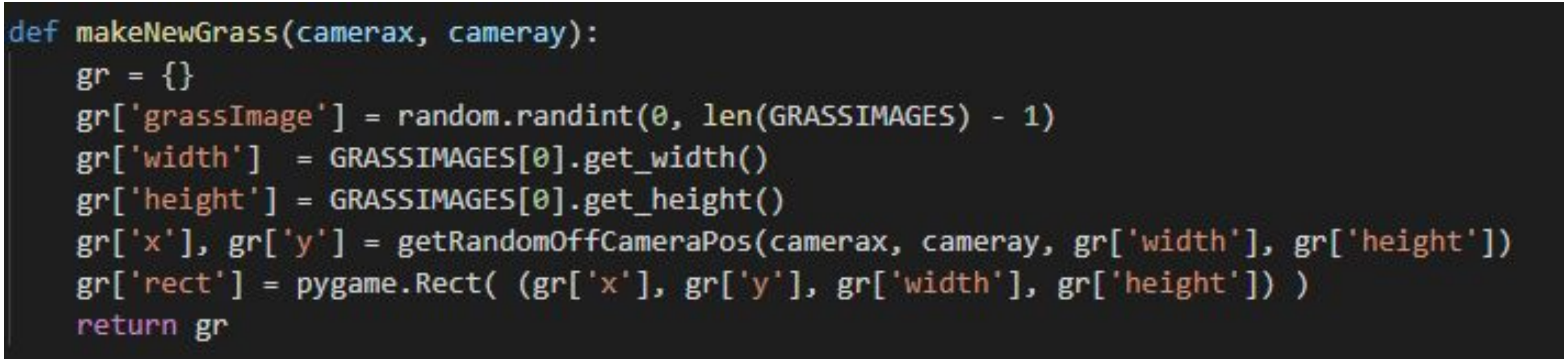
http://invpy.com/mariocamera



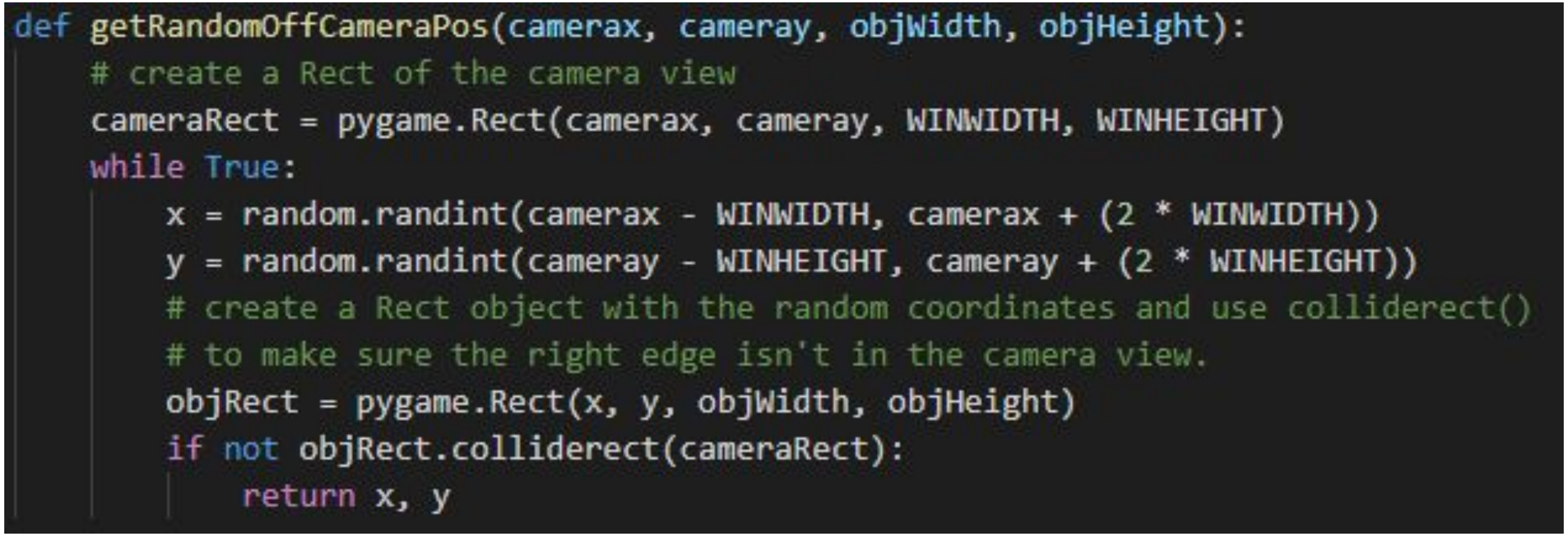


```python
def runGame():

    # camerax and cameray are the top left of where the camera view is
    camerax = 0
    cameray = 0

    grassObjs = []
    for i in range(10):
        grassObjs.append(makeNewGrass(camerax, cameray))
        grassObjs[i]['x'] = random.randint(0, WINWIDTH)
        grassObjs[i]['y'] = random.randint(0, WINHEIGHT)
```

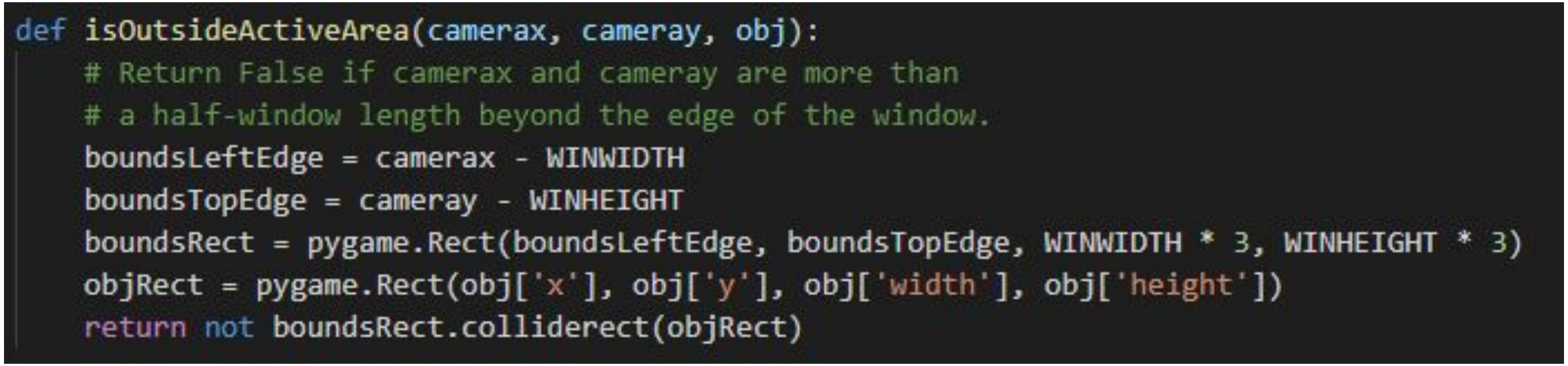

```python
while True: # main game loop
    DISPLAYSURF.fill(GRASSCOLOR)

    # 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]

    while len(grassObjs) < NUMGRASS:
        grassObjs.append(makeNewGrass(camerax, cameray))

    # 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)
```



```python
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
```



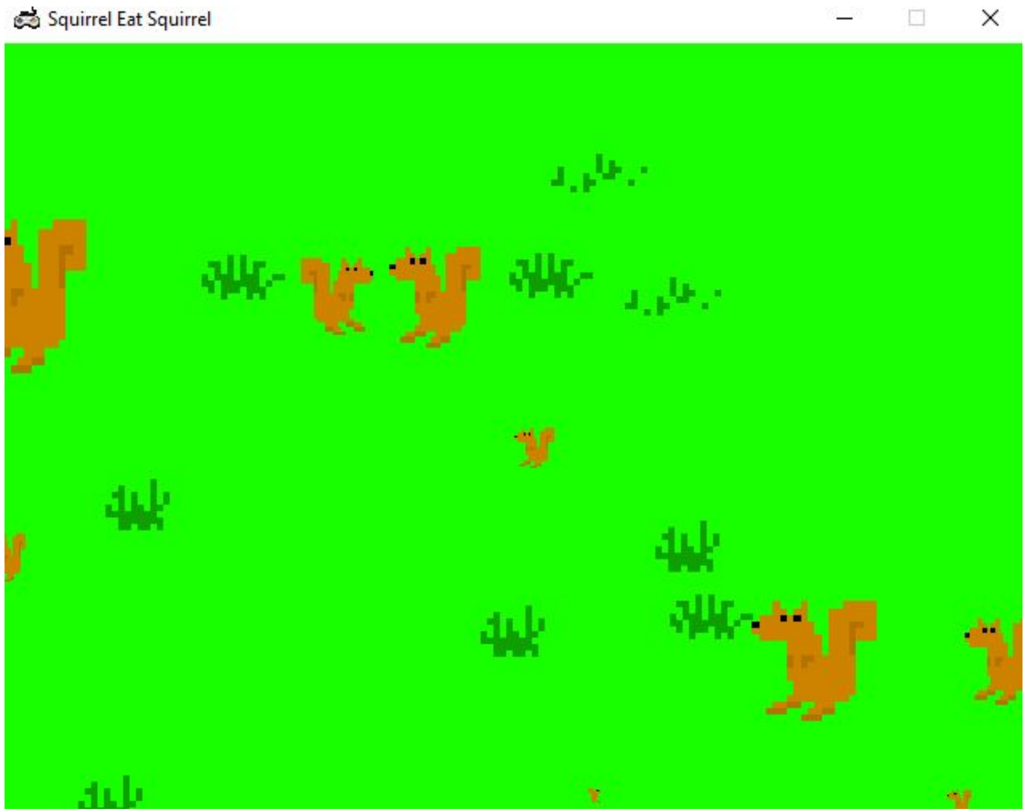
```python
def getRandomOffCameraPos(camerax, cameray, objWidth, objHeight):
    # create a Rect of the camera view
    cameraRect = pygame.Rect(camerax, cameray, WINWIDTH, WINHEIGHT)
    while True:
        x = random.randint(camerax - WINWIDTH, camerax + (2 * WINWIDTH))
        y = random.randint(cameray - 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
```

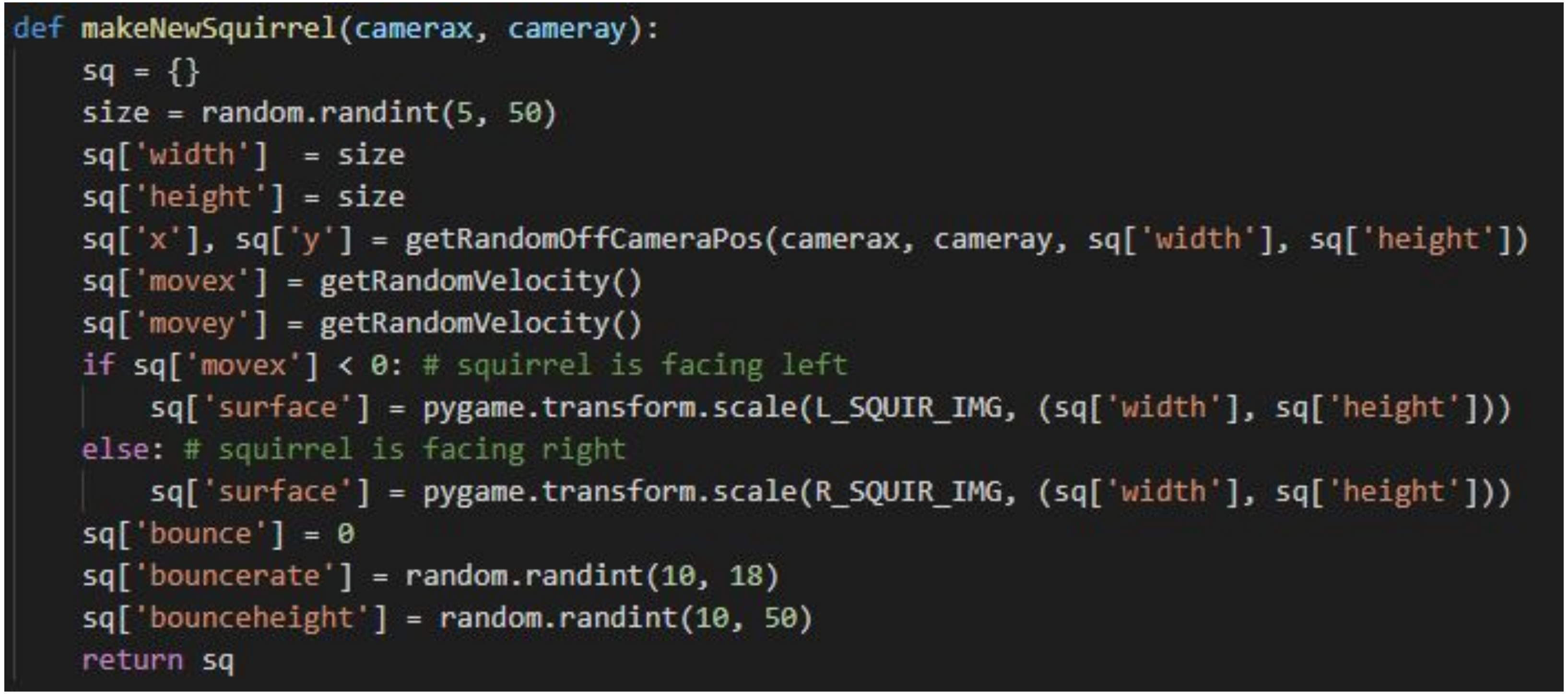


```python
def isOutsideActiveArea(camerax, cameray, obj):
    # Return False if camerax and cameray are more than
    # a half-window length beyond the edge of the window.
    boundsLeftEdge = camerax - WINWIDTH
    boundsTopEdge = cameray - WINHEIGHT
    boundsRect = pygame.Rect(boundsLeftEdge, boundsTopEdge,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

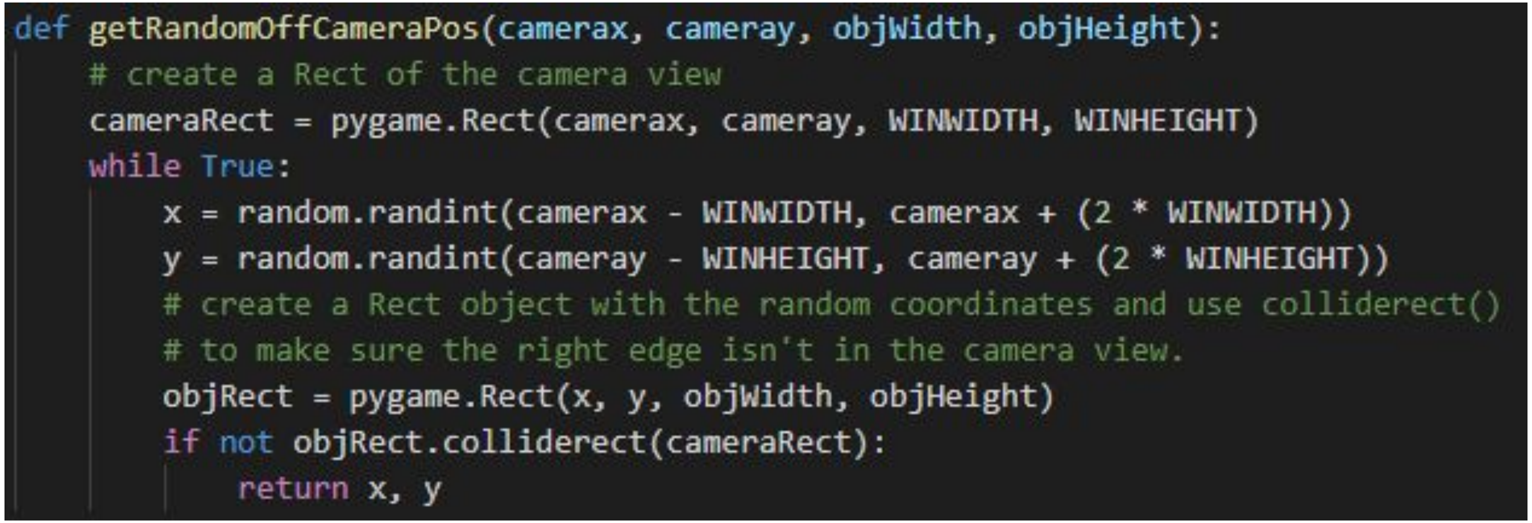
Project 3: Squirrel fight

- Generate enemy squirrels
- Enemy squirrel movement
- Collision of player squirrel and enemy squirrel

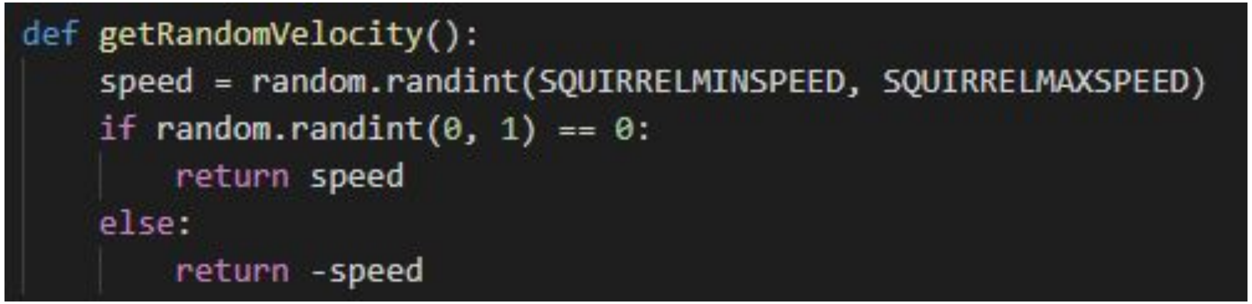




```python
def makeNewSquirrel(camerax, cameray):
    sq = {}
    size = random.randint(5, 50)
    sq['width']  = size
    sq['height'] = size
    sq['x'], sq['y'] = getRandomOffCameraPos(camerax, cameray, sq['width'], sq['height'])
    sq['movex'] = getRandomVelocity()
    sq['movey'] = getRandomVelocity()
    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
```

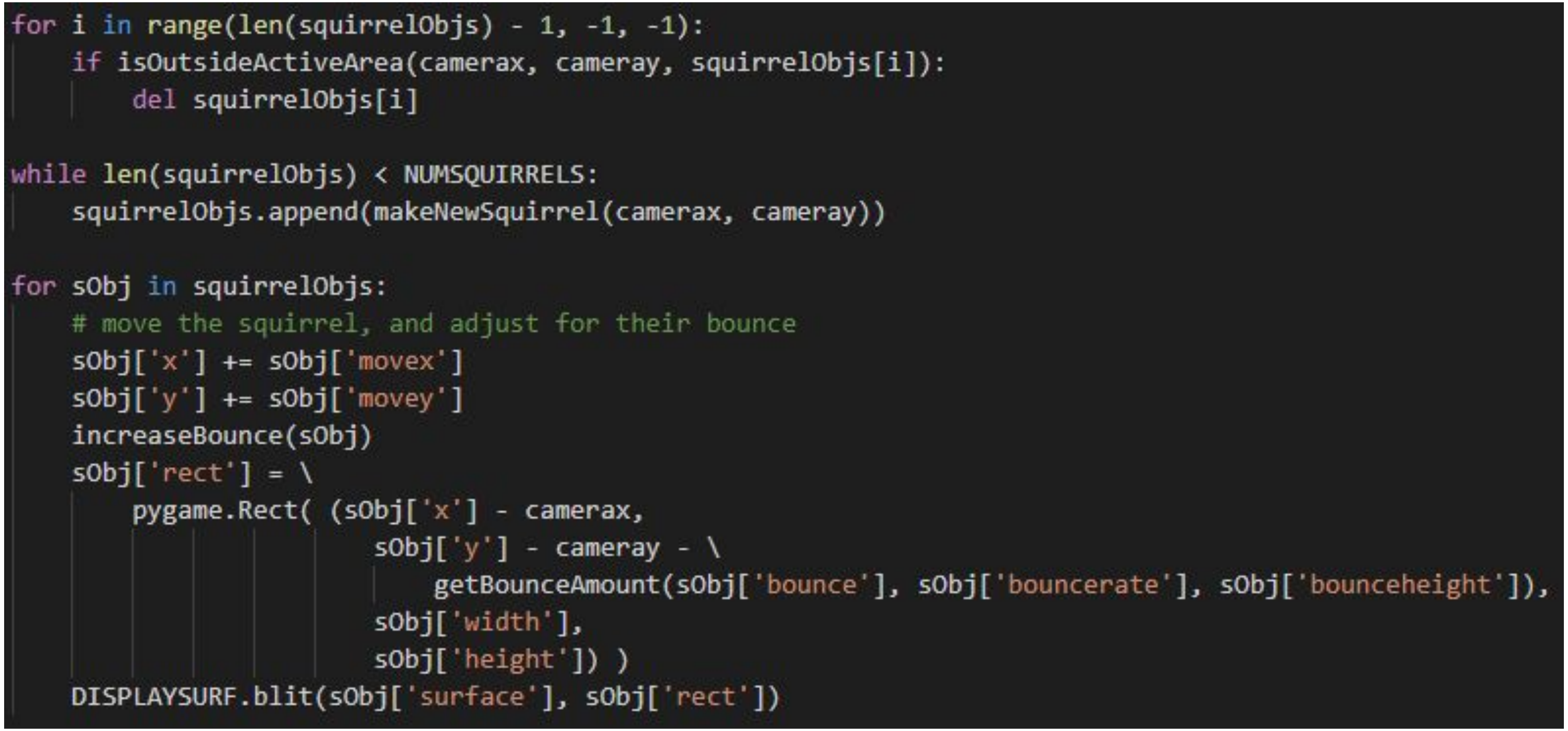


```python
def getRandomOffCameraPos(camerax, cameray, objWidth, objHeight):
    # create a Rect of the camera view
    cameraRect = pygame.Rect(camerax, cameray, WINWIDTH, WINHEIGHT)
    while True:
        x = random.randint(camerax - WINWIDTH, camerax + (2 * WINWIDTH))
        y = random.randint(cameray - 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
```



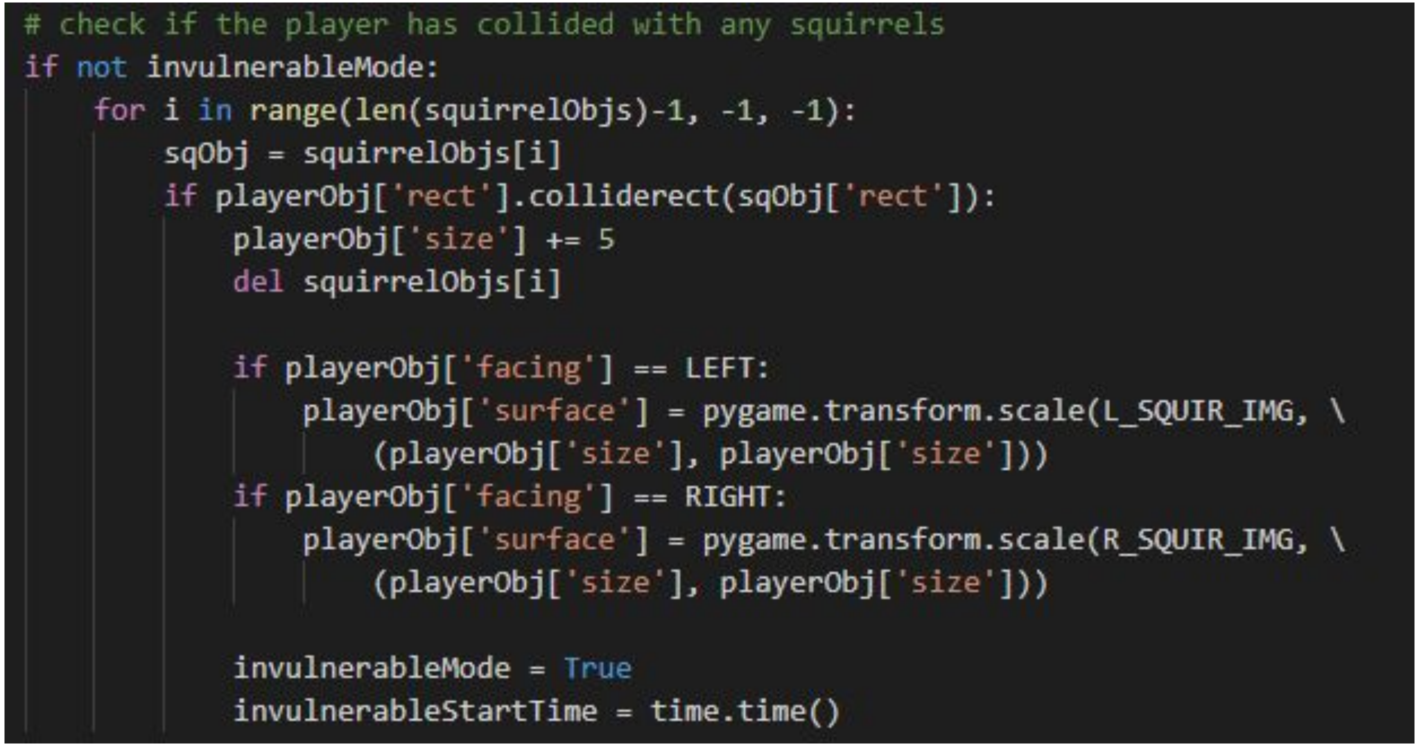
```python
def getRandomVelocity():
    speed = random.randint(SQUIRRELMINSPEED, SQUIRRELMAXSPEED)
    if random.randint(0, 1) == 0:
        return speed
    else:
        return -speed
```

In the main loop:

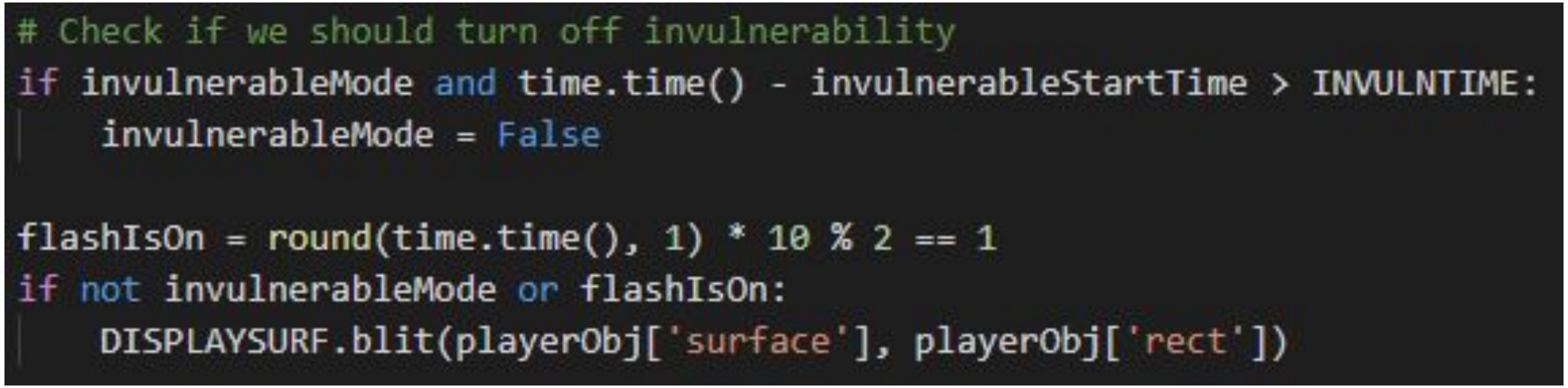


```python
for i in range(len(squirrelObjs) - 1, -1, -1):
    if isOutsideActiveArea(camerax, cameray, squirrelObjs[i]):
        del squirrelObjs[i]

while len(squirrelObjs) < NUMSQUIRRELS:
    squirrelObjs.append(makeNewSquirrel(camerax, cameray))

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

Collision of player squirrel and enemy squirrels



```python
# check if the player has collided with any squirrels
if not invulnerableMode:
    for i in range(len(squirrelObjs)-1, -1, -1):
        sqObj = squirrelObjs[i]
        if playerObj['rect'].colliderect(sqObj['rect']):
            playerObj['size'] += 5
            del squirrelObjs[i]

            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']))

            invulnerableMode = True
            invulnerableStartTime = time.time()
```



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

flashIsOn = round(time.time(), 1) * 10 % 2 == 1
if not invulnerableMode or flashIsOn:
    DISPLAYSURF.blit(playerObj['surface'], playerObj['rect'])
```

Bonus

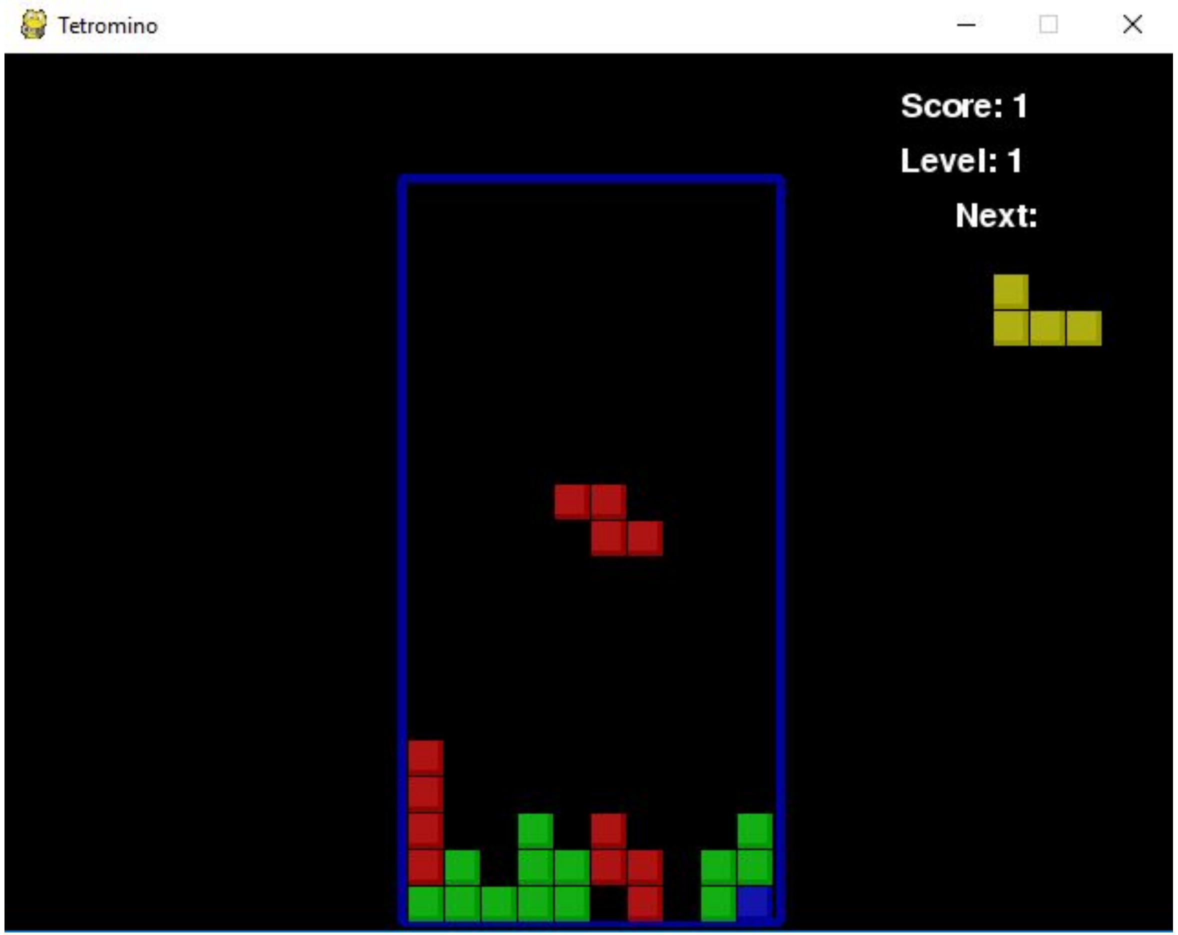
Tetromino

Tetromino

Press a key to play.

Instruction:

- Left Key: Left
- Right Key: Right
- Up Key: Rotation
- Down Key: Down
- Space: Drop


Tetromino
Score: 1
Level: 1
Next: