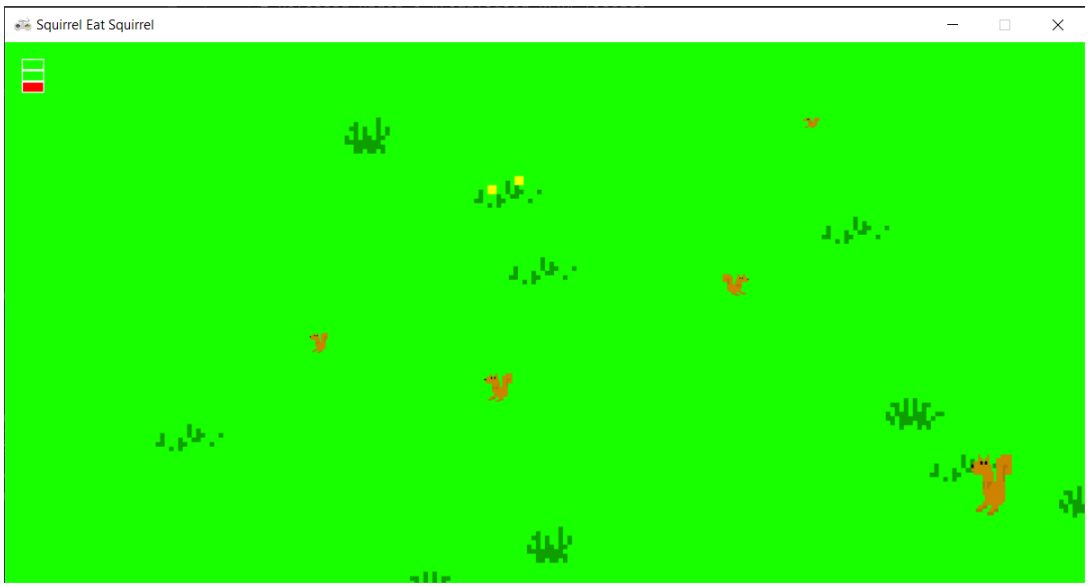
Лабараториска вежба Squirrel

1. Increase the camera area of the game to 960x480 pixels (1 point).

```
# Change for requirement 1:
# WINWIDTH = 640 # width of the program's window, in pixels
WINWIDTH = 960  # width of the program's window, in pixels
WINHEIGHT = 480  # height in pixels
```

За да го исполниме ова барање само ги ажурираме константите кои ги означуваат димензиите на екранот



2. Define two new constants for the CAMERASLACK constants for the horizontal and vertical direction appropriately (1 point).

```
# Change for requirement 2:
# CAMERASLACK = 90     # how far from the center the squirrel moves
before moving the camera
CAMERASLACK_X = 120  # how far from the center the squirrel moves
before moving the camera
CAMERASLACK_Y = 60  # how far from the center the squirrel moves
before moving the camera
```

Наместо да имаме една константа CAMERASLACK ќе дефинираме по една се секоја насока x и y. Дополнително поставуваме CAMERASLACK_X да е поголема од CAMERASLACK_Y за да одоговара со односот помеѓу ширината и висината на екранот.

Потоа во main loop во делот каде што се користи константа CAMERASLACK ја заменуваме со CAMERASLACK_X или CAMERASLACK_Y соодветно.

```
# Change for requirement 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

if (camerax + HALF_WINWIDTH) - playerCenterx > CAMERASLACK_X:
    camerax = playerCenterx + CAMERASLACK_X - HALF_WINWIDTH
elif playerCenterx - (camerax + HALF_WINWIDTH) > CAMERASLACK_X:
    camerax = playerCenterx - CAMERASLACK_X - HALF_WINWIDTH
if (cameray + HALF_WINHEIGHT) - playerCentery > CAMERASLACK_Y:
    cameray = playerCentery + CAMERASLACK_Y - HALF_WINHEIGHT
elif playerCentery - (cameray + HALF_WINHEIGHT) > CAMERASLACK_Y:
    cameray = playerCentery - CAMERASLACK_Y - HALF_WINHEIGHT
```

3. Introduce the possibility an enemy squirrels to bounce downwards (the squirrel could bounce only in one direction only upwards or only downwards). The direction is determined when creating an enemy squirrel. (3 points)

```
# Change for requirement 3:
# def getBounceAmount(currentBounce, bounceRate, bounceHeight):
def getBounceAmount(currentBounce, bounceRate, bounceHeight,
bounceDir='up'):
    # 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
    if bounceDir == 'up':
        return int(math.sin((math.pi / float(bounceRate)) *
currentBounce) * bounceHeight)
    return -int(math.sin((math.pi / float(bounceRate)) *
currentBounce) * bounceHeight)
```

Функцијата getBounceAmount ја менуваме така што ќе прима уште еден аргумент кој ќе има предодредена вредност 'up' и ќе ја означува насоката на отскокнување (bounce).

Овој аргумент ќе го искористиме така што доколку насоката е up се враќа вредност добиена со истата функција како и пред промената, а доколку вредноста на аргументот е ‘down’ тогаш пак се враќа вредност добиена со функцијата пред промената но помножена со -1.

```
def makeNewSquirrel(camerax, cameray):
...
    # Change for requirement 3:
    sq['bouncedir'] = random.choice(["up", "down"])
    return sq
```

Ја менуваме функцијата makeNewSquirrel така што речникот (dictionary) што ја представува верверичката ќе има уште еден атрибут кој ќе ја означува насоката на отскокнување (bounce) и по случаен избор ќе прима вредност up или down.

4. Change the logic of the game. If the squirrel is hit by the larger animal it becomes smaller using the same logic as for the getting bigger at the current game. The game is over when the squirrel grows to the WINSIZE or LOSTSIZE (5).

```
# Change for requirement 4:
LOSESIZE = 10
# MAXHEALTH = 3  # how much health the player starts with
```

Додаваме константа LOSESIZE која ќе ја представува минималната дозволена големина на верверичката контролирана од играчот. Ако големината на верверичката добие вредност помала од LOSESIZE тогаш играчот изгубил.
Дополнително ја бришеме (коментираме) константата MAXHEALTH затоа што нема да ни биде потребна.

```
# Change for requirement 4:
# draw the health meter
# drawHealthMeter(playerObj['health'])
```

```
# Change for requirement 4:
# 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)
```

Ја отсрануваме (коментираме) наредбата со која исцртуваме уште колку животи ни преостанале затоа што не е потребна.

```
elif not invulnerableMode:
    # player is smaller and takes damage
    invulnerableMode = True
    invulnerableStartTime = time.time()

    # Change for requirement 4:
    # playerObj['health'] -= 1
    # if playerObj['health'] == 0:
    #     gameOverMode = True  # turn on "game over mode"
    #     gameOverStartTime = time.time()
    playerObj['size'] -= int((sqObj['width'] * sqObj['height']) **
0.2) + 1
    if playerObj['size'] < LOSESIZE:
        gameOverMode = True  # turn on "game over mode"
        gameOverStartTime = time.time()
    else:
        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']))
```

Во main loop во делот каде се извршува логиката доколку играчот се судрил со поголема верверичка правиме измени така што големината на играчот ја намалуваме користејќи ја истата фомула која се користи при зголемување на играчот и дополнително проверуваме дали големината на играчот е помала од минималната дозволена големина LOSESIZE доколку е тогаш играчот изгубил и се уклучува "game over mode", а во спротивен случај се ажурира големината на верверичката.