

Arcanoid

Αρχικοποίηση

Βάλτε όλες τις απαραίτητες εντολές για την αρχικοποίηση του παιχνιδιού:

- Imports
- Μεταβλητές

```
WIDTH, HEIGHT = 800, 600
FPS = 60
BG_COLOR = (20, 20, 20)
```

- Game loop
- Exit
- Κλπ

```
1 import pygame
2 import sys
3
4 WIDTH, HEIGHT = 800, 600
5 FPS = 60
6 BG_COLOR = (20, 20, 20)
7
8 pygame.init()
9 screen = pygame.display.set_mode((WIDTH, HEIGHT))
10 pygame.display.set_caption("Arcanoid - Step 1")
11 clock = pygame.time.Clock()
12
13 running = True
14 while running:
15     clock.tick(FPS)
16
17     for event in pygame.event.get():
18         if event.type == pygame.QUIT:
19             running = False
20
21     screen.fill(BG_COLOR)
22     pygame.display.flip()
23
24 pygame.quit()
25 sys.exit()
26
```

Controls

Δείτε και τα προηγούμενα αρχεία και φτιάξε την κίνηση της ρακέτας.

```
running = True
while running:
    clock.tick(FPS)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT]:
        paddle.x -= PADDLE_SPEED
    if keys[pygame.K_RIGHT]:
        paddle.x += PADDLE_SPEED

    if paddle.left < 0:
        paddle.left = 0
    if paddle.right > WIDTH:
        paddle.right = WIDTH
```

Κίνηση μπάλας

Προσθέτε μια μπάλα και στη συνέχεια κάντε την να κινείται. Αν η μπάλα ακουμπήσει στα πλαινά θα πρέπει να αλλάξει πορεία. Το ίδιο θα πρέπει να γίνει και αν ακουμπήσει τη ρακέτα.

```
ball = pygame.Rect(
    WIDTH // 2 - BALL_SIZE // 2,
    HEIGHT // 2,
    BALL_SIZE,
    BALL_SIZE
)
pygame.draw.ellipse(screen, (0, 200, 255), ball)
```

```
collision_ball = ball.inflate(-20, -20)

if collision_ball.colliderect(paddle):
    if ball_vel[1] > 0:
        ball.bottom = paddle.top-1
        ball_vel[1] *= -1
```

Εδώ φτιάχνουμε ένα μικρότερο σχήμα από αυτό της μπάλας (inflate με αρνητικές τιμές) ώστε η μπάλα να μην συγκρούεται με τις άκρες της ρακέτας.

Διορθώσεις:

1. Αυξήστε κατάλληλα το ύψος του παραθύρου
2. Errors

Traceback (most recent call last):

```
File "c:\Users\lampros\Desktop\seed\python2526\LabLessons\arcenoid.py",
line 21, in <module>
    ball_speed_x *= -1
    ^^^^^^^^^^^^^
NameError: name 'ball_speed_x' is not defined
PS C:\Users\lampros\Desktop\seed\python2526>
```

Μεταφέρετε τον κώδικα σύγκρουσης της μπάλας σε συνάρτηση. Αντί να ορίσετε μεταβλητή `ball_speed_x` χρησιμοποιείτε το pygame object `ball` μέσω του `ball.x`. Στο τέλος τη συνάρτησης επιστρέψτε όλο το `ball`.

3. Η ρακέτα μπορεί να φύγει εκτός ορίων
4. ΠΡΟΣΟΧΗ ΟΤΑΝ ΞΑΝΑΟΡΙΖΕΤΕ μεταβλητές...

Δημιουργία bricks

Αντιγράψτε και ενσωματώστε την παρακάτω συνάρτηση:

```
def create_bricks(rows=3, brick_h=25, margin=5, top_offset=40):
    bricks = []
    cols = WIDTH // (80 + margin)
    total_margin = margin * (cols + 1)
    brick_w = (WIDTH - total_margin) // cols
    start_x = (WIDTH - (cols * brick_w + total_margin)) // 2 + margin
    for row in range(rows):
        color = ROW_COLORS[row % len(ROW_COLORS)]
        for col in range(cols):
            x = start_x + col * (brick_w + margin)
            y = top_offset + row * (brick_h + margin)
            brick = {
                "rect": pygame.Rect(x, y, brick_w, brick_h),
                "color": color
            }
            bricks.append(brick)
    return bricks
```

Σε αυτή τη συνάρτηση δεν επιστρέφουμε απλά ένα pygame Rect αλλά ένα dictionary. Η διαφορά του dictionary από τους πίνακες είναι ότι μπορεί να περιέχει στοιχεία διαφορετικών ειδών.

Πίνακας (list):

Αποθηκεύει στοιχεία σε σειρά και τα καλούμε με index ([0], [1]). Χρήσιμος όταν όλα τα δεδομένα είναι ίδιας μορφής και μας νοιάζει η σειρά.

Dictionary (dict):

Αποθηκεύει ζεύγη κλειδιού–τιμής και τα καλούμε με όνομα (["rect"], ["color"]). Χρήσιμος όταν θέλουμε διαφορετικά είδη δεδομένων με ξεκάθαρο νόημα.

Για αυτό το λόγο όταν πάμε να τα «τυπώσουμε» στην οθόνη:

```

for brick in bricks:
    pygame.draw.rect(
        screen,
        brick["color"],
        brick["rect"],
        border_radius=4
    )

```

Θα

χρησιμοποιήσουμε:

```

brick["color"], ---> rect
brick["rect"], ---> color

```

```

brick = {
    "rect": pygame.Rect(x, y, brick_w, brick_h),
    "color": color
}

```

Έλεγχος Σύγκρουσης

Φτιάξτε μια συνάρτηση η οποία θα ελέγχει αν η μπάλα χτύπησε (collide) με κάποιο τουβλάκι. Αν ναι αλλάξτε ταχύτητα και αφαιρέστε το τουβλάκι αυτό από τον πίνακα bricks.

```

for brick in bricks[:]:
    if ball.colliderect(brick["rect"]):
        bricks.remove(brick)
        speed[1] *= -1
        return True
return False

```

Προσθέστε έλεγχο/Game Over

```

if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_SPACE and game_over:

```

εδω κάντε ότι χρειάζεται ώστε να **ξανά**αρχίσει το παιχνίδι

```
if not game_over:
    paddle = pd.move_paddle(paddle, PADDLE_SPEED)
    ball = bl.move_ball(ball, paddle)
    if ball.y > HEIGHT:
```

και εδώ κάντε ό,τι χρειάζεται ώστε να χαθεί μία ζωή αν περάσει η μπάλα, και αν τελειώσουν οι ζωές να ενεργοποιηθεί ο πάνω έλεγχος!

```

while running:
    clock.tick(FPS)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE and game_over:
                bricks = br.create_bricks()
                lives = LIVES
                game_over = False
                paddle.x = WIDTH// 2- PADDLE_WIDTH//2
                paddle.y = HEIGHT-4*PADDLE_HEIGHT
                ball.x = WIDTH//2
                ball.y = HEIGHT//2
                speed[0] = BALL_SPEED
                speed[1] = -BALL_SPEED

if not game_over:
    paddle = pd.move_paddle(paddle,PADDLE_SPEED)
    ball = bl.move_ball(ball, paddle)
    if ball.y > HEIGHT:
        lives -= 1
        if lives <= 0:
            game_over = True
            speed[0] = 0
            speed[1] = 0
        else:
            paddle.x = WIDTH// 2- PADDLE_WIDTH//2
            paddle.y = HEIGHT-4*PADDLE_HEIGHT
            ball.x = WIDTH//2
            ball.y = HEIGHT//2
            speed[0] = BALL_SPEED
            speed[1] = -BALL_SPEED

```