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

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

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

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.

```
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\arcanoid.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 χρησιμοποιείστε 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 λογο όταν πάμε να τα «τυπώσουμε» στην οθόνη:

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

Θα χρησιμοποιησουμε:

```
  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.

```
if ball.colliderect(brick["rect"]):
…
…
```

Αλλάζουμε την μεταβλητή “flag” σε “game_over”. Αυτή τη στιγμή όταν η μπάλα φεύγει από το κάτω όριο, η οθόνη απλά μαυρίζει. Θέλουμε αντί για αυτό, να κάνουν reset οι ρυθμισεις και να χάνουμε μια ζωή. Προσθέστε τον παρακάτω έλεγχο στο παιχνιδι και βάλτε τις κατάλληλες εντολές.

```
if not game_over:
    if ball.y >= HEIGHT:
        [redacted]
        if lives > 0:
            [redacted]
        else:
            game_over = True
```

Όταν χαθούν και οι 3 ζωές (game_over=True) τότε θα μαυρίζει η οθόνη, θα εμφανίζεται κατάλληλο μήνυμα και θα πρέπει αν πατήσουμε space να αρχίζει από την αρχή. Αλλιώς να κάνει ότι έκανε.

```
if game_over:
    screen.fill(BLACK)
    msg_surf = font.render("Game Over - Press Space to play again", True, WHITE)
    msg_rect = msg_surf.get_rect(center=(WIDTH//2, HEIGHT//2))
    [redacted]
else:
    [redacted]
pygame.display.flip()
```

Προσθέστε σε κατάλληλο σημείο τις παρακάτω εντολές

```
heart_size =
        padding =
        start_x =
        start_y =
        for i in range(max(0, lives)):
            hx = start_x + i * (heart_size + padding)
            heart.draw_heart(screen, hx, start_y, heart_size)
```

Επίσης προσθέστε στο αρχείο heart.py τις παρακάτω συναρτήσεις και δοκιμάστε να δείτε τι κάνει η κάθε μια.

```python
def draw_heart_parabolic(surface, x, y, size, color=RED, samples=64):
    cx = x + size / 2
    cy = y + size / 2
    base_scale = size / 34.0
    pts = []
    for i in range(samples):
        t = (i / samples) * 2 * math.pi
        tx = 16 * math.sin(t) ** 3
        ty = 13 * math.cos(t) - 5 * math.cos(2 * t) - 2 * math.cos(3 * t) - math.cos(4 * t)
        px = cx + tx * base_scale
        py = cy - ty * base_scale
        pts.append((int(px), int(py)))
    pygame.draw.polygon(surface, color, pts)
    outline_color = (max(0, color[0] - 60), max(0, color[1] - 60), max(0, color[2] - 60))
    pygame.draw.aalines(surface, outline_color, True, pts)

def draw_heart(surface, x, y, size, color=RED):
    radius = size // 4
    left_center = (x + radius, y + radius)
    right_center = (x + radius*3, y + radius)
    pygame.draw.circle(surface, color, left_center, radius)
    pygame.draw.circle(surface, color, right_center, radius)
    points = [
        (x, y + radius),
        (x + size, y + radius),
        (x + size//2, y + size)
    ]
    pygame.draw.polygon(surface, color, points)
```

Τέλος προσθέστε τις κατάλληλες εντολές ώστε όταν τελειώσουν οι ζωές το παιχνίδι να αρχίζει ξανα όταν πατήσουμε space.

```python
running = True
while running:
    clock.tick(FPS)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN and game_over:
            if event.key == pygame.K_SPACE:
                [redacted]
    paddle = pd.move_paddle(paddle,PADDLE_SPEED)
    ball = bl.move_ball(ball, paddle)
    screen.fill(BG_COLOR)
```

Βρείτε το σημείο ελέγχου σύγκρουσης brick-ball και κάντε την παρακάτω αλλαγή:

```
for brick in bricks:
    pygame.draw.rect(screen, brick["color"], brick["rect"], border_radius=10)
collided = False
for brick in bricks[:]:
    if ball.colliderect(brick["rect"]):
        bricks.remove(brick)
        collided = True
        break
if collided:
    speed[1] *= -1
```

Αυτό θα κάνει μόνο μια σύγκρουση ανά frame,

Διόθρωση με δυναμικο μέγεθος.

```
#dynamic resizing works
if paddle.x > WIDTH - paddle.width:
    paddle.x = WIDTH - paddle.width
```

και διορθωση ορίων

if ball.x > WIDTH or ball.x < 0: -------> if ball.right >= WIDTH or ball.left <= 0:

Powerup!

Δύο λίστες. Το powerups είναι η λίστα που κρατάει αυτά που πέφτουν από πάνω και είναι της μορφης

```
powerups.append({
                    "x": cx,
                    "y": cy,
                    "type": random.choice(["grow", "shrink"])
```

Αρα {x,y,grow or shrink)

```
powerups = []
# active powerups stack: list of {"mult": float, "end": timestamp}
active_powerups = []
```

και το active_powerups το οποίο είναι αυτά που έχουμε πιάσει και είναι ενεργά.

```
    {"mult": αριθμός, "end": χρόνος_λήξης}
```

Όταν σπάσει ένα τούβλο και με κάποια πιθανότητα:

```
            if random.random() < POWERUP_CHANCE:
                cx, cy = brick["rect"].center
                powerups.append({
                    "x": cx,
                    "y": cy,
                    "type": random.choice(["grow", "shrink"])
                })
```

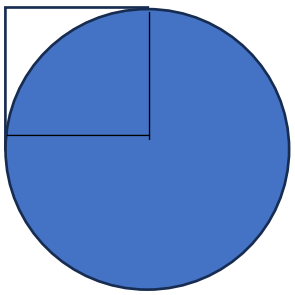
1. Παράγεται ένας τυχαίος αριθμός από 0 έως 1.
2. Αν είναι μικρότερος από `POWERUP_CHANCE` → δημιουργείται powerup και μπαίνει στη λίστα powerups.
3. Αποθηκεύουμε:
    - θέση (x, y)
    - τύπο (grow ή shrink με random.choice

Σε κάθε επανάληψη θα πρέπει να σχεδιάσουμε το κάθε powerup και να το κάνουμε να κινείται προς τα κάτω. Άρα τρέχουμε τη λίστα με τα powerups (αν υπάρχουν) και αλλάζουμε διαρκώς τη θέση y.

```
for pu in powerups[:]:
        pu["y"] += POWERUP_SPEED
```

Στη συνέχεια το σχεδιάζουμε κατά τα γνωστά:

```
pu_rect = pygame.Rect(
      int(pu["x"]) - POWERUP_RADIUS, #x
      int(pu["y"]) - POWERUP_RADIUS, #y
      POWERUP_RADIUS*2, POWERUP_RADIUS*2) #ακτινα*2 - διάμετρος
pygame.draw.circle(screen, (200, 160, 120), pu_rect.center,
POWERUP_RADIUS)
```



Αν το ακουμπήσουμε με το paddle:

κρατάμε τον χρόνο τώρα

```
    now = pygame.time.get_ticks()
```

παίρνουμε το είδους του multiplier και το αποθηκευουμε

```
if pu["type"] == "grow":
                mult = POWERUP_MULT_GROW
            else:
                mult = POWERUP_MULT_SHRINK
```

Γεμίζουμε τον πίνακα με τα **ενεργά** multiplier

```
active_powerups.append({"mult": mult, "end": now + POWERUP_DURATION})
```

```
total = 1.0
for ap in active_powerups:
     total *= ap["mult"]
     paddle.width = max(20, min(WIDTH, int(base_paddle_width * total)))
powerups.remove(pu)
```

Επειδή μπορούμε ταυτόχρονα να έχουμε παραπάνω από ένα powerups, θα κρατήσουμε ένα σύνολο αυτών. Το total αποθηκευει τον συνολικο συντελεστή απ όλα τα ενεργά powerups. Αφού αλλάξουμε το μέγεθος, σβήνουμε το powerup από την λίστα ώστε να εξαφανιστεί. Το ίδιο κάνουμε και αν μας ξεφύγει…

```
elif pu["y"] - POWERUP_RADIUS > HEIGHT:
            powerups.remove(pu)
```

Στη συνέχεια θα αθροίσουμε το σύνολο των συντελεστώ και για όση ώρα είναι ενεργά θα φτιάχνουμε το νεό πλάτος του paddle:

```
 if active_powerups:
        now = pygame.time.get_ticks()
        active_powerups = [ap for ap in active_powerups if ap["end"] >now]
        total = 1.0
        for ap in active_powerups:
            total *= ap["mult"]
        new_width = max(20, min(WIDTH, int(base_paddle_width * total)))
        if new_width != paddle.width:
            paddle.width = new_width
```

Αλλάξτε την powerups:

```
pu_type = random.choices(["grow", "shrink"], weights=[GROW_WEIGHT,
SHRINK_WEIGHT], k=1)[0]
                powerups.append({
                    "x": cx,
                    "y": cy,
                    "type": pu_type
```

Τα WEIGHTS είναι συντελεστές που έχουν αθροισμα 1 ώστε να αλλαξει η πιθανοτητα εμφάνισης κάθε powerup.

Επίσης αλλάζουμε και το active_powerups:

```
    now = pygame.time.get_ticks()
            if pu["type"] == "grow":
                mult = POWERUP_MULT_GROW
                active_powerups.append({"mult": mult, "end": now + POWERUP_DURATION})
            else:
                mult = SHRINK_MULT_OVERRIDE
                active_powerups[:] = [{"mult": mult, "end": now + POWERUP_DURATION}]
            powerups.remove(pu)
```

Στην πρώτη περίπτωση θα αθροίζει τα grow pu ενώ αν τύχει shrink θα τα σβήσει όλα και θα βάλει μόνο ένα.