

Za radoznale: Petlja igre, vreme i kretanje

Ovo nije obavezno i neće biti na ispitu. Za one koji žele da vide kako se ista stvar radi naprednije. Svi ispisi su dobijeni pokretanjem u pygame-ce.

1. Stalni korak: fizika koja ne zavisi od dt

Na času smo u svakom frejmu predmet pomerili za brzina $\times$ dt. To je dovoljno za većinu malih igara. Kad igra ima skokove, gravitaciju i sudare, različit dt ponekad daje različite rezultate: na sporom računaru lik skoči malo više ili prođe kroz tanak zid. Zato mnoge igre fiziku računaju u stalnim koracima, na primer 1/60 sekunde, a crtaju koliko stignu.

```
KORAK = 1 / 60          # fizika uvek ide po 1/60 sekunde
akumulator = 0
y = 0
brzina = 0
gravitacija = 500       # piksela u sekundi na kvadrat

for dt in [0.05, 0.016, 0.1, 0.016]: # frejmovi različitog trajanja
    akumulator += dt
    koraka = 0
    while akumulator >= KORAK:
        brzina += gravitacija * KORAK
        y += brzina * KORAK
        akumulator -= KORAK
        koraka += 1
    print(f"frejm od {dt} s, broj koraka fizike: {koraka}, y = {y:.1f}")
```

Ispis:

```
frejm od 0.05 s, broj koraka fizike: 3, y = 0.8
frejm od 0.016 s, broj koraka fizike: 0, y = 0.8
frejm od 0.1 s, broj koraka fizike: 6, y = 6.2
frejm od 0.016 s, broj koraka fizike: 1, y = 7.6
```

Akumulator skuplja vreme. Dok u njemu ima bar jedan ceo korak, računamo jedan korak fizike. Dug frejm od 0,1 s daje šest koraka, a kratak frejm ponekad nijedan; ostatak vremena čeka sledeći frejm. Ovo je ideja iz teksta Glena Fidera „Fix Your Timestep!“.

Zašto ovo nismo radili na času: za igre koje pravimo u ovom predmetu brzina $\times$ dt je dovoljna, a stalni korak uvodi još jednu petlju i promenljivu koja čuva ostatak vremena.

Fajl: `primeri/za_radoznale/07_r1_stalni_korak.py`

2. FRect: pravougaonik sa realnim koordinatama (samo pygame-ce)

Rect pamti samo cele brojeve. Kada predmet pomeramo za 1,6 piksela po frejmu i čuvamo položaj u Rect, decimale se gube i predmet ide sporije nego što treba, ili stoji. Zato smo na času položaj čuvali u Vector2. U pygame-ce postoji i FRect, koji pamti realne brojeve.

```
import pygame as pg

r = pg.Rect(0, 0, 40, 40)
f = pg.FRect(0, 0, 40, 40)
for frejm in range(10):
    r.x += 0.6          # Rect odseca decimale: 0 + 0.6 je opet 0
    f.x += 0.6
print("Rect:", r.x)
print("FRect:", round(f.x, 1))
```

Ispis:

```
Rect: 0
FRect: 6.0
```

Posle deset frejmova Rect je i dalje na nuli, a FRect je stigao do 6. FRect radi sa svim funkcijama za sudare, kao i obični Rect.

Zašto ovo nismo radili na času: FRect postoji samo u pygame-ce, a naši primeri treba da rade i u starom pygame-u; Vector2 rešava isti problem i postoji u obe biblioteke.

Fajl: `primeri/za_radoznale/07_r2_frect.py`

3. Smer i rotacija vektora

Vector2 ima još mnogo metoda. `normalize()` pravi strelicu dužine 1 u istom smeru, pa je posle lako pomnožiti brojem i dobiti tačnu brzinu. `rotate(ugao)` okreće strelicu za zadati broj stepeni. Tako se pravi, na primer, metak koji leti ka mišu ili sneg koji vetar nosi ukoso.

```
import pygame as pg

igrac = pg.Vector2(100, 100)
mis = pg.Vector2(400, 500)
smer = (mis - igrac).normalize()
brzina = smer * 300      # 300 piksela u sekundi, ka mišu
print(smer, brzina.length())

vetar = pg.Vector2(0, 80).rotate(-30)  # sneg pada ukoso
print(round(vetar.x, 1), round(vetar.y, 1))
```

Ispis:

```
[0.6, 0.8] 300.0
40.0 69.3
```

Razlika dva položaja je strelica od igrača ka mišu. Kad je normalizujemo, dužina je 1, pa je posle množenja brojem 300 brzina tačno 300 piksela u sekundi, u kom god smeru da je miš.

Zašto ovo nismo radili na času: za ovo je potrebno razumevanje uglova i smeru, a u temi 07 smo se bavili samo time da kretanje ne zavisi od računara.

Fajl: `primeri/za_radoznale/07_r3_smer_i_rotacija.py`