Sveučilište u Zagrebu
Fakultet organizacije i informatike

Vektori i pravocrtno gibanje

U Varaždinu, 25.12.2023. Polak Lucija, Niko Rastija, Petra Skoko, grupa G2.1

1. Uvod

Ovaj projekt će obraditi temu vektora i pravocrtnoga gibanja. Na početku su objašnjeni neki osnovni pojmovi koji objašnjavaju tematiku kretanja objekata u simulacijama i u stvarnome životu te se pritom uspoređuju brzina slike i brzina objekta. Kako bismo prikazali simulaciju brzine i kretanje vektora, koristili smo Pygame te programski jezik Python kako bismo napravili nekoliko videoigrica koji pokazuju te kretnje, a to su: kretanje aviona 200 piksela po sekundi, mogućnost mijenjanja brzine aviona, kretanje strelice te odbijanje kuglice od pločice. Na kraju se nalazi naš zaključak o izradi ovoga projekta te uočenim zapažanjima prilikom korištenja nama novog alata za programiranje videoigara. Svi kodovi koje objašnjavamo u ovome radu su napisani te se nalaze u ostalim datotekama koje su priložene uz ovaj projekt.

2. Razrada

2.1. Simuliranje, brzina slike i brzina objekta

Kretanje objekta na računalu se simulira uz pomoć računalne simulacije. Za početak ćemo objasniti što je to računalna simulacija. Računalnu simulaciju možemo definirati kao neku igru u kojoj koristimo matematiku da bismo prikazali što će se dogoditi u stvarnom životu. Računalni program zatim te rezultate prikazuje kao brojeve ili slike koje se kreću [1]. Simulacije se mogu izvršavati pomoću raznih programa poput Pygame, Unity, Unreal Engine itd. U našem slučaju koristit će se Pygame.

Za simulaciju objekta na računalu koristi se objektno orijentirano programiranje gdje stvaramo razne klase koje nam predstavljaju različite karakteristike i ponašanja objekata [2]. Na temelju tih klasa izrađujemo objekte. Svaki objekt ima neke svoje atribute to jest podatke i metode odnosno funkcije koje mu omogućuju da izvršava razne radnje. Tako zapravo definiramo kako će naši objekti izgledati i kako će se ponašati pri čemu nastaje simulacija.

Brzina slike (engl. Frame Rate) se odnosi na mjerljivost koliko se sličica stvara na grafičkoj kartici računala u jednoj sekundi [3]. Povećanje broja sličica u sekundi rezultirat će znatno glađom animacijom [4]. Zbog mogućnosti provjere brzine slike, korisnici računala, pogotovo korisnici računalnih igrica, mogu vidjeti koliko njihovo računalo ima mogućnost korištenja igrica visoke zahtjevnosti grafičke kartice. Brzina slike se mjeri u broju sličica (okvira) u sekundi (engl. Frames Per Second). Brzina slike ovisi o računalnom sklopovlju, tj. očvrsnu (engl. hardware) (poput procesora, radne memorije, grafičke kartice i matične ploče), postavkama rezolucije unutar igrice korisnika te optimiziranosti koda [3]. Optimiziranost koda se odnosi na razvijenost igrice za performanse grafičke kartice.

S druge strane, **brzina objekta** ovisi o brzini i smjeru gibanja. Ovisi i o prijeđenom putu u nekome vremenu. Mjerna jedinica je skalarna veličina koja se izražava u metrima po sekundi, tj. m/s. [5]

Brzina slike i brzina objekta se obje odnose na stvoreni ulaz u određenom vremenu, ali su ulazni podatci drukčiji. Brzina slike za ulaz uzima brzinu računalnih komponenti dok brzina objekta uzima prijeđeni put. Sljedeća slika okvirno uspoređuje kolika mora biti brzina objekta te koja je odgovarajuća brzina slike:

Vrsta objekta	Brzina objekta	Brzina slike
Hod čovjeka	5 km/h	5 fps
Auto unutar naselja	50 km / h	23 fps
Auto na autocesti	130 km/h	60 fps

Slika 1: Usporedba brzine objekta i brzine slike, prema literauri[6]

2.2. O alatu Pygame

Pygame je skup Python modula posebno razvijenih za izradu video igara. Sastoji se od raznih računalnih grafičkih i zvučnih biblioteka koje su osobito dizajnirane za korištenje u kombinaciji s programskim jezikom Python. Pygame je svestran i kompatibilan s gotovo svakom platformom i operativnim sustavom. Besplatan je tako da svatko ima mogućnost izraditi svoju video igricu. [7]

Kako bi se započeo rad u PyGameu i omogućilo programiranja igrica, potrebne su osnovne sintakse i naredbe, a navedene su samo neke koje smo mi koristili u programiranju našega projekta poput:

- **pygame.init()** – služi za inicijalizaciju PyGame programa kako bi se napravila baza za daljnji rad.
- **širina_ekrana**, **visina_ekrana** = 800, 600 - služi za postavljanje veličine zaslona koji ćemo koristiti, prvi broj predstavlja širinu dok drugi broj predstavlja visinu našega početnoga ekrana.
- **ekran=pygame.display.set_mode ((širina_ekrana, visina_ekrana))** - stvaramo varijablu „ekran“ u koju pohranjujemo naše vrijednosti za inicjalizaciju ekrana i njegovo stvaranje pomoću prethodno definiranih vrijednosti širine i visine
- **pygame.display.set_caption(”Avion animacija”)** - postavljamo naslov našega novostvorenog zaslona u „Avion animacija“.
- **logo = pygame.image.load('logo_avion.png')** - stvaranje varijable imenom „logo“ koji će sadržavati sliku aviona i time ikonu našega programa.
- **pygame.display.set_icon(logo)** - postavljanje prethodno stvorenog loga u logo aplikacije.
- **while True: . . . pygame.display.update()** - petlja koja nam omogućuje da prozor u kojem se nalazi naša igrica ostane upaljen dok se ne ispuni neki uvjet za gašenje, npr. pritisak na određeni gumb.

2.3. Programska rješenja zadataka

2.3.1 Zadatak 4. Kretanje aviona 200 piksela po sekundi

U ovom zadatku napravili smo animaciju u kojoj imamo dva aviona koja se kreću lijevo i desno po zaslonu. Gornji avion se kreće puno glađe od donjeg jer ima veći FPS nego donji avion što se može vidjeti po zastajkivanju donjega aviona. Zbog sporijeg stvaranja sličica kod donjega aviona, možemo vidjeti da tome objektu brzina slike "kasni" za originalnom brzinom objekta. Kako bi doživljaj programa bio autentičniji, u igrici smo omogućili opciju punoga zaslona (engl. fullscreen). U gornjem lijevom kutu napisana je trenutna brzina koja iznosi 200 piksela po sekundi, a u gornjem desnom kutu prikazana je vrijednost FPS zaslona koja ide do 60. Koristili smo osnovne naredbe koje smo prethodno opisali, ali smo i koristili formule za izračunavanje pomaka aviona kroz vrijeme poput `avion_spori.x += pomakX_os * proteklo_vrijeme * brojac` te smo u petlji koja omogućuje otvoreni prozor svaki put „osvježili" zaslon kako bi naše promjene bile uočljive korisniku pomoću naredbe `pygame.display.flip()`.

Osiguranje da avion ostanu unutar okvira zaslona je omogućeno uvjetima koji provjeravaju kada je avion došao do krajnje točke preko koje ne smije prijeći:

```
    if(avion_brzi.x >= 735):
        avion_brzi.x = 735
    if(avion_brzi.x < 1):
        avion_brzi.x = 1
    if(avion_spori.x >= 735):
        avion_spori.x = 735
    if(avion_spori.x < 1):
        avion_spori.x = 1
```

Kašnjenje aviona sporije brzine slike smo omogućili koristeći selekciju u petlji glavnoga programa. Selekcija omogućuje da nakon što brži (gornji) avion prođe 5 okvira, sporiji ga uspije dostići te nastaviti svoj put. Ovime smo omogućili upravo simuliranje brzine slika dva različita objekta u prostoru te kako se brzina slika prenosi u uređajima različite kvalitete. Kod koji to omogućuje glasi:

```
    if brojac == 5:
        avion_spori.x = avion_brzi.x
        avion_spori.x += pomakX_os * proteklo_vrijeme * brojac
        avion_spori.y += pomakY_os * proteklo_vrijeme * brojac
        brojac = 0
```

2.3.2 Zadatak 5. Mogućnost mijenjanja brzine aviona

Ovaj zadatak je sličan prethodnom zadatku, međutim, razlika je u broju aviona. Naime, u ovome zadatku imamo jedan avion koji se kreće lijevo i desno. U ovome je programu moguće podešavati brzinu upotrebom gornje i donje strelice na tipkovnici. Brzina se može mijenjati u rasponu od 100 do 2000 piksela po sekundi te je prikazana u gornjem lijevom kutu zaslona. U gornjem desnom kutu zaslona prikazan je FPS koji također ide do 60 kao i u prethodnom zadatku. Ključni dio koda koji bismo željeli istaknuti je sljedeći:

```
    keys = pygame.key.get_pressed()

    if keys[pygame.K_UP]:
        if brzina_aviona < 2000 and (time.time() -
vrijeme_strelica_gore) > 0.1:
            brzina_aviona += brzina_promjena
            vrijeme_strelica_gore = time.time()

    elif keys[pygame.K_DOWN]:
        if brzina_aviona > 100 and (time.time() -
vrijeme_strelica_dolje) > 0.1:
            brzina_aviona -= brzina_promjena
            vrijeme_strelica_dolje = time.time()
```

Iznad navedeni dio koda nam omogućuje mijenjanje brzine aviona koji se kreće po zaslonu. Prva selekcija se odnosi na povećanje brzine pritiskom na strelicu prema gore na tipkovnici, a druga selekcija na strelicu prema dolje te smanjenje brzine. Kako bi brzina pomaka našega aviona bila usklađena uz njegovo kretanje u prostoru, potrebno je uskladiti vrijeme te se ono ažurira.

2.3.3 Zadatak 6. Kretanje strelice

U ovom zadatku koristimo naše znanje o vektorima i pravocrtnom gibanju koje smo stekli na kolegiju Matematičke metode za informatičare, tj. MMI. Za rješavanje ovog zadatka koristili smo trigonometriju. Za početak ćemo ukratko opisati kako smo postavili određena svojstva strelice.

Prvo je bilo potrebno definirati početne pozicije strelice. To smo učinili tako što smo postavili da nam varijable `x` i `y` predstavljaju početne koordinate strelice te smo zatim koristili argument `center` kako bi se one inicijalno pojavile na sredini zaslona. Nakon toga smo odredili za koliko će se strelica micati po x i y osi varijablama `pomakXos` i `pomakYos`, a te se vrijednosti zatim dodaju trenutnim koordinatama strelice kako bi se izračunao njezin novi položaj.

S obzirom na to da se naša strelica može kretati posvuda bilo je potrebno stvoriti uvjete koje ju ograničavaju da ne izlazi iz zadanog okvira igrice. Osim toga, bilo je potrebno omogućiti strelici da se rotira, a to smo učinili pomoću određene funkcije tako što smo uzeli vrijednost rotacije koja je određena varijablom `rotation` i oko rotirane strelice, nakon rotacije, generirali smo pravokutni oblik (`rotated_rect`) koristeći njezine trenutne koordinate.

U programu koristimo trigonometriju za određivanje pomaka strelice uzimajući u obzir njezinu rotaciju i smjer kretanja. Ovo je dio koda gdje to računamo:

```
if moving_forward:
    pomakXos = math.sin((((rotation + 90) / 180) * math.pi) % (2 *
   math.pi)) * brzina
    pomakYos = math.cos((((rotation + 90) / 180) * math.pi) % (2 *
   math.pi)) * brzina
elif moving_backward:
    pomakXos = math.sin((((rotation + 90) / 180) * math.pi) % (2 *
   math.pi)) * (-brzina)
    pomakYos = math.cos((((rotation + 90) / 180) * math.pi) % (2 *
   math.pi)) * (-brzina)
```

Trigonometrijske funkcije poput `math.sin` i `math.cos` koriste se za izračunavanje komponenti pomaka po x i y osi. Matematičke formule koje smo koristili omogućuju strelici da se kreće u određenom smjeru ovisno o kutu rotacije. Koristimo trigonometriju kako bismo pretvorili rotacije strelice u horizontalne i vertikalne komponente pomaka koje ćemo onda koristiti za pomicanje strelice. U našem kodu `rotation` je 0 te predstavlja trenutni kut rotacije strelice. Prvo, izrazom $\frac{\texttt{(rotation + 90)}}{180}$ pretvaramo kut iz stupnjeva u omjer radijana pa je zatim dobiveni rezultat potrebno pomnožiti s π kako bismo pretvorili omjer radijana u radijane. Potom je potrebno reducirati kut na interval $[0, 2\pi\rangle$ što smo učinili na način da smo uzeli dobiveni kut u radijanima i izvršili operaciju modulo ($\%(2\pi)$) nad njim. Na kraju cijelu jednadžbu množimo s brzinom kako bismo postigli željeni pomak. Formule koje smo koristili za horizontalni pomak (`pomakXos`) i vertikalni pomak (`pomakYos`) su skoro identične, a jedinu razliku čini upotreba funkcije sinus (`sin`) za horizontalni pomak dok smo za vertikalni pomak koristili funkciju kosinus (`cos`).

2.3.4 Zadatak 7. Odbijanje kuglice od pločice

Posljednji zadatak u ovome radu je igra s lopticom i pločicom koja se pomiče uz pomoć strelica na tipkovnici. Kada je igra započeta, omogućen je pregled dosad ostvarenih bodova i novododana funkcionalnost - odabir opcije igre, tj. je li uključen teži način igre. Ovo omogućuje da

korisnik dobije priliku igrati malo kompliciraniju igru gdje je brzina loptice brže nego prema originalnim postavkama. Ovaj pregled se nalazi u gornjem lijevom kutu, u desnome se nalazi broj preostalih života, a u sredini se nalazi gumb za odabir punog zaslona igrice. Nakon što broj preostalih života dođe na ”0”, korisnik dobiva mogućnost odabrati želi li ponovno pokrenuti igru, odabir težine igre, broj ostvarenih bodova te opcija za izlazak iz igrice. Ukoliko je uključena opcija teže igre, u gornjem lijevom kutu će pisati ”TEZA IGRA: True”, a ukoliko je isključena, pisat će tekst ”TEZA IGRA: False”.

Pomicanje pločice u smjeru pritisnutih strelica je omogućeno unutar provjere pokrenutosti igre. Naime, kuglica se kreće u nasumičnome smjeru gledajući x-os brzinom od 5 piksela po sekundi (osim kada je uključena opcija teže igre). Kod u programu:

```
    if (kraj_igre == False):
        kuglica_x += kuglica_smjer_x * kuglica_brzina
        kuglica_y += kuglica_smjer_y * kuglica_brzina

        if (micanje_desno and not micanje_lijevo):
            plocica_pomak_Xos = plocica_brzina
        elif (micanje_lijevo and not micanje_desno):
            plocica_pomak_Xos = -plocica_brzina
        elif (not micanje_lijevo and not micanje_desno):
            plocica_pomak_Xos = 0

        plocica_x += plocica_pomak_Xos * prosjecno_vrijeme
```

Odbijanje loptice od zidova je zajamčeno naredbom koja će smjer kuglice pomnožiti s `-1` po x-osi kada se dođe do lijevoga ili desnoga ruba,a provjera odbijanja loptice u pločicu se provjerava usporedbom x i y koordinata loptice i pločice. Ako se loptica nalazi unutar granica pločice, doći će do odbijanja te se smjer kuglice pomnoži s `-1` kako bi loptica po y-osi krenula u suprotnome smjeru. Ovo se nalazi u kodu u naredbama:

```
    # Odbijanje od zidova
        if kuglica_x - kuglica_radius <= 0 or kuglica_x +
    kuglica_radius >= sirina_ekrana:
            kuglica_smjer_x *= -1

            # Odbijanje od plotjice
        if (
            plocica_x <= kuglica_x <= plocica_x + plocica_sirina and
            plocica_y <= kuglica_y + kuglica_radius <= plocica_y +
    plocica_visina
        ):
```

```
            kuglica_smjer_y *= -1
            bodovi += 1
```

Novododana funkcionalnost teže igre mijenja jednu bitnu stavku, a to je, kao što je prethodno navedeno, brzina kuglice. Brzina kuglice postane duplo veća nego što je inače kada je ova opcija uključena te za to služi jednostavna selekcija u programu:

```
    if (teza_igra == True):
        kuglica_brzina = 10
    else:
        kuglica_brzina = 5
```

3. Zaključak

Ukoliko pričamo o brzini slike kod videoigara, za bolju kvalitetu nam je potrebna što veća brzina slike, a manjom brzinom igra izgleda manje fluidno te su uočljiva zastajkivanja naših kretajućih objekata. Zbog toga je mnogim igračima videoigara važnije imati što bolje sklopovlje radi što bolje kvalitete igralačkog iskustva.

Stjecanje nove vještine stvaranja videoigre i korištenja jezika Python kako bi se to omogućilo ukazalo nam je na nove načine korištenja vektorskog prostora i gibanja u prostoru kako bismo prethodno naučenu teoriju mogli primijeniti u praksi i u području koje se svakim danom sve više koristi i postaje raznovrsnije.

Kako bismo potvrdili istinitost novih saznanja o brzini slike i objekta, postavili smo upit literaturi [8]: ”Usporedi brzinu slike i brzinu objekta kada pričamo o videoigrama”. Dobili smo odgovor koji je dodatno potvrdio naše teze ”U idealnoj situaciji, želite visok FPS i pravilno kalibriranu brzinu objekta kako biste postigli optimalno iskustvo igranja. Međutim, ponekad postoje kompromisi jer za postizanje visokih FPS-ova može biti potrebna moćnija hardverska oprema, a postizanje ravnoteže između brzine objekta i FPS-a može zahtijevati pažljivo prilagođavanje igre ili postavki.”

S obzirom na to da nam je ovo prvi put korištenja platforme poput Pygame, ChatGPT nam je, također, pomogao u shvaćanju sintakse jezika Python koji se sintaksno razlikuje od jezika C++, pomogao nam je u snalaženju u novoj atmosferi programiranja videoigara te je proširio naše opće znanje o programiranju videoigrica i njihovim karakteristikama.

Literatura

[1] Uređivači enciklopedije Brittanica, *Computer simulation*, https://www.britannica.com/technology/computer-simulation, Brittanica, 27.09.2023.

[2] facundo3141592, *How do computer programs simulate real world objects?*, https://brainly.com/question/34617417, Brainly, 07.10.2023.

[3] Michael Klappenbach, *Understanding and Optimizing Video Game Frame Rates*, https://www.lifewire.com/optimizing-video-game-frame-rates-811784#:~:text=The%20areas%20that%20can%20impact,and%20developed%20for%20graphics%20performance, Lifewire, 05.09.2022.

[4] Reducible, Building Collision Simulations: An Introduction to Computer Graphics, https://www.youtube.com/watch?v=eED4bSkYCB8, YouTube, 19.01.2021.

[5] Nora Hashem, *Brzina Objekta Ovisi O*, https://hr.sadaalomma.com/Brzina-objekta-ovisi-o/, Eho nacije, 19.01.2023.

[6] Martin Daněk, https://www.researchgate.net/figure/Required-frame-rate-versus-object-motion-speed_tbl1_261377994, ReserchGate, 01.2012.

[7] Pygame, *About - wiki*, https://www.pygame.org/wiki/about (24.12.2023.)

[8] ChatGPT, https://chat.openai.com/ (25.12.2023.)

[9] Alat LaTeX, 25.12.2023.