Primjena vektorske grafike u videoigri Asteroids

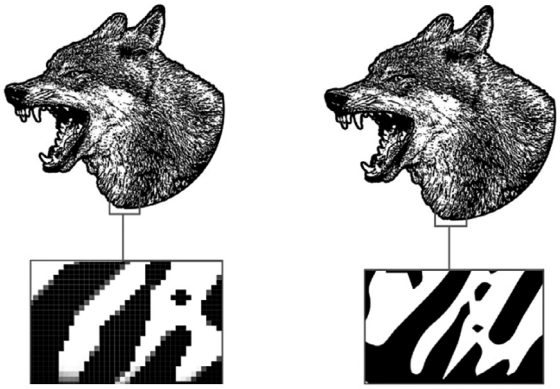
Patrik Ostrunić[1], Tihana Strmečki[2] i Ivana Božić Dragun[3]

U ovom ćemo članku istražiti primjenu vektorske grafike i kako je iskoristiti za stvaranje vizualno privlačnog i zanimljivog iskustva igranja videoigara. Analizirat ćemo primjenu matematičkih vektora u geometrijskom prikazu videoigre *Asteroids*, te prednosti korištenja vektorske grafike nad rasterskom za stvaranje i skaliranje različitih geometrijskih tijela. Za vizualne materijale koristit ćemo besplatni interaktivni program *GeoGebra*, dok ćemo za kod koristiti programski jezik C++.

Za početak navodimo osnovne definicije vezane uz matematički pojam vektora. Neka su točke A i B dvije točke u Euklidskoj ravnini. Vektor u oznaci $\overrightarrow{AB}$ usmjerena je dužina definirana tim dvjema točkama. Početnu točku A zovemo hvatište vektora, a završnu točku B vrh vektora. Skup svih vektora ravnine označavamo s V^2. Vektor je određen s tri osnovna pojma: duljinom, smjerom i orijentacijom. Modul vektora jest duljina dužine $\overrightarrow{AB}$ i označava se s $\left|\overrightarrow{AB}\right|$. Smjer vektora je pravac na kojem leži vektor, koji određuju točke A i B. Orijentacija vektora redoslijed je hvatišta i vrha, što omogućava dvije suprotne orijentacije. Orijentaciju lako odredimo gledajući strelicu koja opisuje vektor vizualno u Euklidskoj ravnini.

U rasterskoj grafici elementi se stvaraju manipulacijom piksela u mreži, pri čemu svaki piksel sadrži karakteristike poput boje, intenziteta i transparentnosti. Za razliku od toga, vektorska grafika vrsta je digitalne grafike koja se sastoji od vektorskih (geometrijskih) oblika i linija. Ona je definirana matematičkim formulama koje određuju oblik, veličinu, boju i poziciju svakog elementa. Upravo je zbog toga najveća prednost vektorske grafike skalabilnost elemenata (sposobnosti grafike da zadrži visoku kvalitetu i jasnoću pri promjeni veličine, bez gubitka oštrine, izobličenja ili pojave pikselizacije) jer se elementi sastoje od vektorskih oblika i linija, a ne od pojedinačnih piksela (Slika 1.).

[1]Patrik Ostrunić, student Tehničkog veleučilišta u Zagrebu
[2]Tihana Strmečki, Tehničko veleučilište u Zagrebu
[3]Ivana Božić Dragun, Tehničko veleučilište u Zagrebu



Slika 1. Razlika između rasterske grafike (lijevo) i vektorske (desno)

Osim skalabilnosti, prednost vektorske grafike jest i zauzimanje male količine memorije jer se pohranjuje mali broj točaka i matematičke relacije između njih koje su izražene kodom. Vektorsku je datoteku lako uređivati jer korisnici mogu brzo i na jednostavan način mijenjati vektorske relacije u svrhu promjene boje elementa ili mijenjanja oblika, što je korisno u iterativnim procesima grafičkog dizajna. Vektorske je datoteke lako prenositi i učitati na različitim uređajima i u različitim programima. Vektorska grafika ima i nedostatke. Jedan od njih jest ograničenost u radu s kompleksnim slikama. Uređivanje fotografija ponekad zahtijeva nijansiranje i miješanje boja koje vektorske datoteke ne mogu omogućiti jednako kvalitetno kao rasterske. Dodatno, vještina stvaranja i potrebno vrijeme zahtjevnije je za vektorske datoteke nego za rasterske. Uz to je i podrška za vektorske grafike na web preglednicima ograničena.

Vektorska se grafika često koristi za stvaranje logotipa, ilustracija, info grafika, karata i drugih grafičkih elemenata koji zahtijevaju preciznost, skalabilnost i često se koriste u tiskanim medijima. Primjer velikih tvrtki za čije je logotipe korištena vektorska grafika su *Apple* (poznat kao „grickalica od jabuke"), *Nike* („swoosh" ili „kuka"), *Google*, *Coca-Cola* i *Lamborghini*. Navedeni logotipi mogu se skalirati od najmanjih veličina, kao što su npr. ikona aplikacije na mobitelu, do samih logotipa na zgradi sjedišta određene tvrtke (Slika 2.).

Slika 2. Prikaz logotipa na zgradi sjedišta tvrtke Google i na Lamborghini automobilu

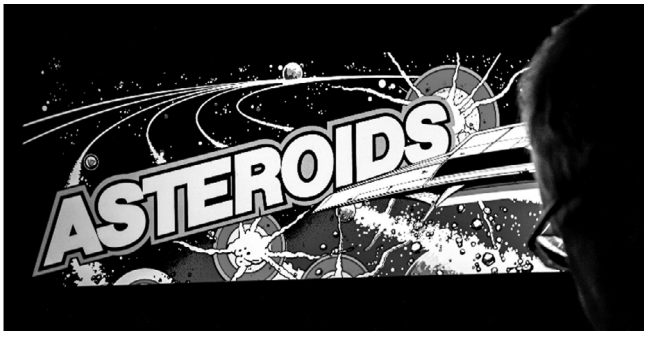
S obzirom na to da obje grafike imaju velikih prednosti, pokazala se nužnom mogućnost konvertiranja jedne u drugu. Softveri poput *Adobe Photoshopa* i *Adobe Illustratora* nude tu opciju. Neki od softvera za stvaranje i uređivanje vektorske grafike su *CorelDRAW, Inkscape* i *Sketch*, dok je najviše korišteni softver *Adobe Illustrator*. Najčešće korištene vrste datoteka vektorske grafike su .ai (*Adobe Illustrator*), .cdr (*CorelDRAW*), .dxf (*Drawing Exchange Format*), .eps (*Encapsulated PostScript*) i .svg (*Scalable Vector Graphics*). Kako bismo iz neobrađene fotografije rasterske grafike dobili verziju fotografije vektorske grafike, ubacujemo je prvo u *Adobe Photoshop*, gdje uz pomoć filtera i uređivanja dobijemo rasteriziranu fotografiju koju potom kopiramo u *Adobe Illustrator*. Dobivenu fotografiju vektoriziramo i imamo konačni rezultat (Slike *3. - 5.*).

Slika 3. Izvorna fotografija

Slika 4. Rasterizirana fotografija nakon uređivanja u Adobe Photoshopu

Slika 5. Vektorizirana fotografija nakon uređivanja u Adobe Illustratoru

Asteroids je jedna od najpoznatijih arkadnih videoigara na temu svemira i višesmjernog pucanja koja koristi vektorsku grafiku. Dizajnirali su je Lyle Rains i Ed Logg, a objavila ju je tvrtka Atari, Inc. u studenom 1979. godine, kada je bilo prodano oko 60 tisuća arkadnih ormarića. Sadržaj igre je da igrač upravlja svemirskim brodom u polju asteroida kojim periodično prelijeću nasumično generirani objekti (nalik na asteroide i leteće tanjure). Cilj je igre skupiti što više bodova pucajući i uništavajući



Slika 6. Videoigra Asteroids

te asteroide i leteće tanjure, izbjegavajući sudare ili protunapade letećih tanjura. Igra postaje teža kako se povećava broj asteroida koji se nasumično generiraju. Igra se temeljila na nedovršenim videoigrama *Spacewar!, Computer Space* i *Space Invaders*.

Igra *Asteroids* prikazana je na vektorskom zaslonu u dvodimenzijskom polju. Napisana je u programskom jeziku Assembly, kao i mnogi klasične arkadne igre toga vremena. Kao što smo napomenuli, mi ćemo koristiti programski jezik C++ koji je danas u češćoj upotrebi. Slijedi programski isječak koda koji se nalazi u zaglavlju igre.

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

class Pozicija {
public:
	int x;
	int y;
};

class Točka {
public:
	int x;
	int y;
};

class SvemirskiBrod {
public:
	Pozicija pozicija;
	double rotacija;
	double brzinaRotacije;
	vector<Točka> koordinateTočaka;
};
```

Klasa **Pozicija** predstavlja položaj svemirskog broda u dvodimenzijskom polju, određenom s dva javna atributa, koordinatama *x* i *y*. Svaka točka svemirskog broda inicijalizira se kao uređeni par cijelih brojeva, pri čemu prvi broj predstavlja koordinatu apscise, a drugi koordinatu ordinate.

Klasa **Točka** predstavlja uređeni par koordinata u istom dvodimenzijskom polju, koji će nam biti potreban za određivanje oblika svemirskog broda.

Klasa **SvemirskiBrod** klasa je koja predstavlja igračev svemirski brod. Ima tri javna atributa:

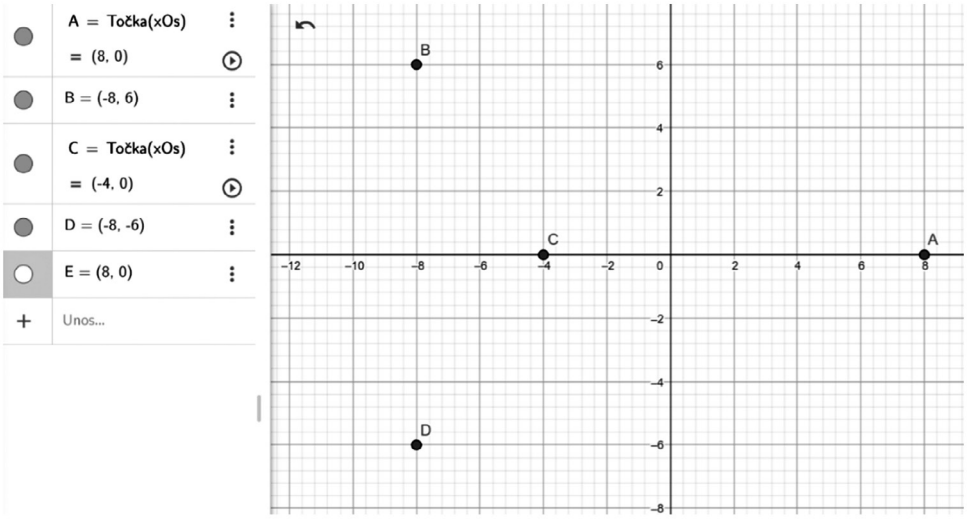
- pozicija – objekt koji određuje točan položaj svemirskog broda u polju na samom početku igre
- rotacija – varijabla tipa double (tip podatka koji se koristi za pohranjivanje brojeva s decimalnim mjestima) koja na početku rotira svemirski brod u željenom smjeru

- brzinaRotacije – varijabla tipa double koja određuje brzinu rotacije svemirskog broda.

U klasi se još nalazi vektor objekata tipa Točka **koordinateTočaka**, koji sadrži sve točke koje formiraju izgled svemirskog broda u polju. U kodu kreiranja svemirskog broda inicijalizacijom objekta SvemirskiBrod koristimo dvodimenzijsko polje veličine 200 × 120 piksela u čiji ga centar smještamo. Koordinatu *x* postavljamo na 100, koordinatu *y* na 60, rotaciju broda inicijaliziramo na vrijednost 5, brzinu rotiranja na 0.07. Potom navodimo dinamički niz vektora određenih s pet uređenih parova cijelih brojeva koji zadaju oblik broda.

```
SvemirskiBrod svemirskiBrod = {
       {100, 60},
       5,
       0.07,
       {
              {8, 0},
              {-8, 6},
              {-4, 0},
              {-8, -6},
              {8, 0}
       }
};
```

Kako bismo lakše vizualizirali gornji kod, prikazujemo na Slici 7. točke *A*(8, 0), *B*(-8, 6), *C*(-4, 0), *D*(-8, -6) i *E*(8, 0) u programu *GeoGebra*.



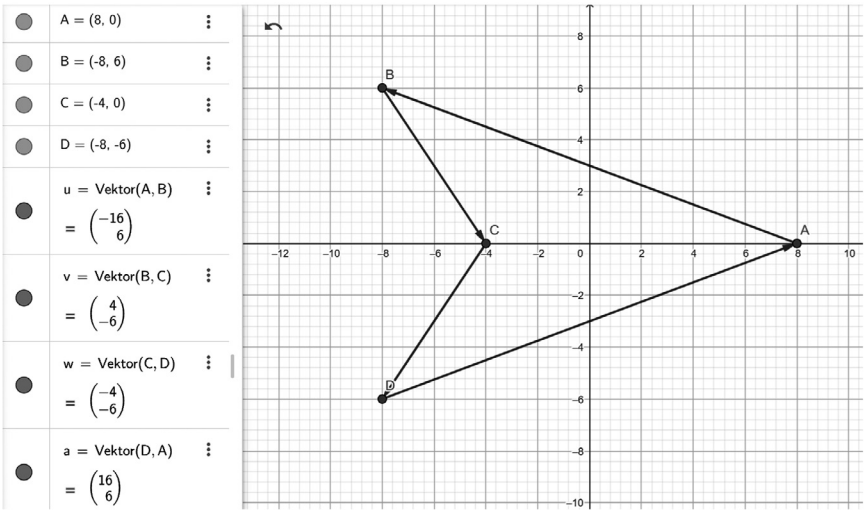
Za finalni oblik svemirskog broda potreban je kod kojim se spaja vrh jednog vektora s hvatištem drugoga. Slijedi programski isječak funkcije **nacrtajVektorskiOblik()** koja će iscrtati spajanje vektora.

```
void nacrtajVektorskiOblik(const SvemirskiBrod& oblik) {
       bool početnaTočka = true;
       Točka zadnjaTočka;
       Točka rotiranaTočka;
       for (const auto& točke : oblik.točke) {
              rotiranaTočka = rotirajTočku(točke, oblik.rotacija);


              if (početnaTočka) {
                     zadnjaTočka = rotiranaTočka;
                     početnaTočka = false;
              }
              else {
                     line(zadnjaTočka.x + oblik.pozicija.x,
     zadnjaTočka.y + oblik.pozicija.y,
     rotiranaTočka.x + oblik.pozicija.x,
     rotiranaTočka.y + oblik.pozicija.y);

                     zadnjaTočka = rotiranaTočka;
              }
       }
}
```

Funkcija nacrtajVektorskiOblik() prima konstantnu referencu na objekt tipa SvemirskiBrod koji sadrži informacije o poziciji, rotaciji i točkama koje definiraju oblik broda. Logička varijabla **početnaTočka** označava je li trenutna točka prva, te u petlji postavljena inicijalno na vrijednost true. **ZadnjaTočka** je objekt tipa Točka koji sadrži koordinate prethodno rotirane točke, dok je **rotiranaTočka** objekt tipa Točka koji sadrži koordinate trenutno rotirane točke. Petlja prolazi kroz sve točke koje su definirane u **oblik.točke** (u našem je primjeru definirano pet točaka), te se za svaku točku poziva funkcija **rotirajTočku()** (koju ćemo kasnije prikazati i pojasniti).

Unutar for-petlje provjerava se je li trenutna točka prva. Ako jest, to znači da kroz petlju prolazimo prvi put, te se prethodna točka postavlja na rotiranaTočka, a varijabla prvaTočka postavlja se na false. Ako trenutna točka nije prva, tada se crta vektor od prethodne rotirane točke (hvatište vektora) do trenutne rotirane točke (vrh vektora). Koordinate vektora određuju se tako da se svaka točka vektorskog oblika broda transformira zbrajanjem svojih koordinata s trenutnom pozicijom broda (**oblik.pozicija**). Time se osigurava da se brod iscrtava na odgovarajućem mjestu u dvodimenzijskom polju, uzimajući u obzir njegovu trenutnu translaciju. Vektor se crta funkcijom **line()**.

Nakon nacrtanog vektora prethodna se točka postavlja na trenutnu rotiranu točku (zadnjaTočka = rotiranaTočka) kako bi se koristila kao prethodna točka u sljedećoj iteraciji petlje. Nakon što je petlja provedena, odmah je jasan izgled svemirskog broda.



Slika 8. Oblik svemirskog broda dobiven funkcijom nacrtajVektorskiOblik

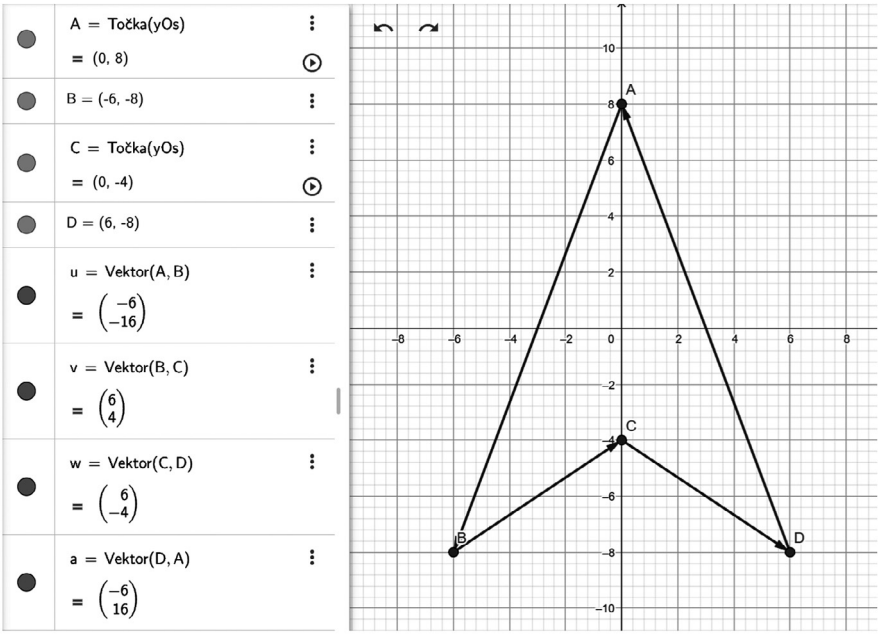
U nastavku slijedi pojašnjenje funkcije **rotirajTočku()** koju koristimo za rotiranje hvatišta i vrha određenog vektora za rotaciju svemirskog broda.

```
Točka rotirajTočku(const Točka& točka, double rotacija) {
	double rotiranoX = (točka.x * cos(rotacija)) - (točka.y *
sin(rotacija));
	double rotiranoY = (točka.y * cos(rotacija)) + (točka.x *
sin(rotacija));

	return { static_cast<int>(rotiranoX), static_
cast<int>(rotiranoY) };
}
```

Funkcija rotirajTočku() služi za rotaciju točke oko ishodišta (0, 0) i izvršava se primjenjujući formulu za rotaciju točke u dvodimenzijskom prostoru. Funkcija prima konstantnu referencu na objekt tipa Točka, koji sadrži koordinate x i y točke koju želimo rotirati (on je vrh prethodnog i hvatište sljedećeg vektora), te vrijednost rotacije θ u radijanima. Unutar funkcije računaju se nove koordinate x' i y' rotiranog hvatišta prethodnog i vrha sljedećeg vektora prema formulama:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = y\cos\theta + x\sin\theta \,.$$

Nakon izračuna vraćaju se nove koordinate kao objekt tipa Točka koja predstavlja rezultat rotacije po zadanim parametrima. Postupak se ponavlja za svaku točku koja zadaje oblik broda, što rezultira iscrtavanjem svih vektora u novom položaju. Na Slici 9. dan je primjer rotiranog svemirskog broda za kut $\frac{\pi}{2}$.



Slika 9. Rotirani svemirski brod za kut $\frac{\pi}{2}$

Nakon primjera izrade i rotacije oblika svemirskog broda u vektorskoj grafici, slijedi primjer izrade asteroida koji svemirski brod mora uništavati. Klasa **Brzina** predstavlja brzinu objekta i sadrži dvije varijable tipa float – brzinu i smjer. Klasa **Asteroid** identična je klasi SvemirskiBrod, uz tri dodane varijable tipa float – akceleraciju, deceleraciju i razmjer asteroida, uz dodatni objekt klase Brzina.

```
class Brzina {
public:
	float brzina;
	float smjer;
};

class Asteroid {
public:
	Pozicija pozicija;
	Brzina brzina;
	float akceleracija;
	float deceleracija;
	float rotacija;
	float brzinaRotacije;
	float razmjer;
	vector<Pozicija> točke;
};
```

Funkcija naziva **stvoriAsteroid()** stvara i vraća instancu objekta tipa Asteroid. Prvo se inicijaliziraju sve varijable potrebne za asteroid s početnim vrijednostima.

Zatim se stvara prva točka asteroida **početnaTočka** koja predstavlja vrh prethodnog i hvatište sljedećeg vektora temeljem kojih ćemo dobiti oblik asteroida. Stvara se na udaljenosti s *x* koordinatom vrijednosti ASTEROID_RADIJUS i koordinatom vrijednosti 0, te se dodaje u vector<Pozicija> točke. Nakon toga slijedi petlja koja generira preostale točke asteroida. Petlja se izvršava ASTEROID_BROJ_TOČAKA - 1 puta. Za svaku iteraciju petlje generiraju se nasumični radijus i kut. Potom se stvara objekt vektor tipa Brzina postavljanjem vrijednosti brzine i smjera na temelju generiranih vrijednosti, te objekt komponente tipa Pozicija koji predstavlja koordinate za pojedinu točku asteroida. Kut vector.smjer označit ćemo s α. Koriste se sljedeće formule za računanje koordinata:

$$x = vector.brzina \cdot \cos\alpha$$
$$y = vector.brzina \cdot \sin\alpha \ .$$

Zbog nasumičnosti generiranja radijusa vector.brzina i kuta asteroid je nepravilne proizvoljne forme. Generirane komponente dodaju se u vector<Pozicija> točke. Nakon provedene petlje dodaje se posljednja točka asteroida koja je ista kao i prva točka. Na kraju se vraća stvoreni objekt asteroida. Donji kod opisuje proces stvaranja asteroida s određenim brojem točaka koje možemo predočiti hvatištem ili vrhom određenog vektora, koristeći matematičke funkcije za generiranje radijusa, kuta i izračunavanje koordinata pojedinih točaka asteroida.

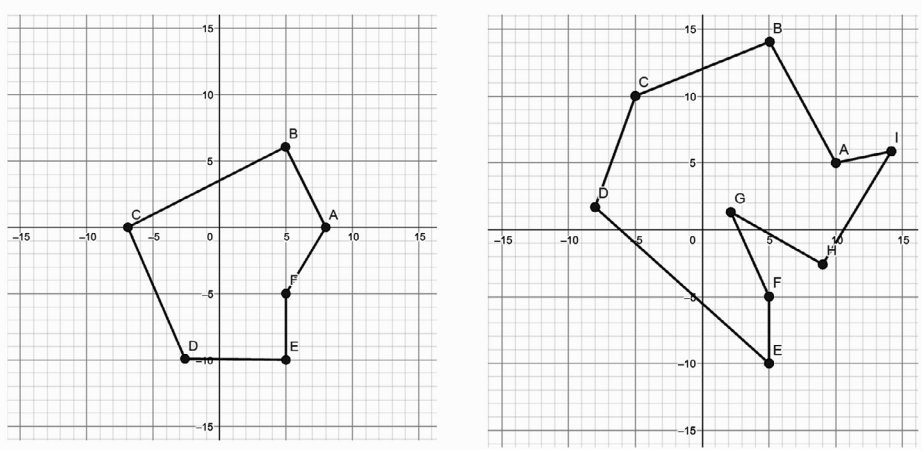
```
Asteroid stvoriAsteroid() {
       Asteroid asteroid;
       asteroid.pozicija.x = 0;
       asteroid.pozicija.y = 0;
       asteroid.brzina.brzina = 0;
       asteroid.brzina.smjer = 0;
       asteroid.akceleracija = 0.05f;
       asteroid.deceleracija = 0.01f;
       asteroid.rotacija = 0;
       asteroid.razmjer = razmjer;
       asteroid.brzinaRotacije = 0.07f;

vector<float> nasumičniKutevi(ASTEROID_BROJ_TOČAKA);
float sumaKuteva = 0.0f;

for (int i = 0; i < ASTEROID_BROJ_TOČAKA; i++) {
nasumičniKutevi[i] = randomFloat(0.5f, 2.0f);
sumaKuteva += nasumičniKutevi[i];
}

for (int i = 0; i < ASTEROID_BROJ_TOČAKA; i++) {
nasumičniKutevi[i] = (nasumičniKutevi[i] / sumaKuteva) * 2 * _Pi;
}

vector<float> kumulativniKutevi(ASTEROID_BROJ_TOČAKA + 1); kumula-
tivniKutevi[0] = 0;
for (int i = 1; i <= ASTEROID_BROJ_TOČAKA; i++) {
```

```
kumulativniKutevi[i] = kumulativniKutevi[i-1] +
    nasumičniKutevi[i-1];
}

       Pozicija početnaTočka;
       početnaTočka.x = ASTEROID_RADIJUS;
       početnaTočka.y = 0;
       asteroid.točke.push_back(početnaTočka);

       float radijus = 0;
       Brzina vector;

       for (int točka = 1; točka < ASTEROID_BROJ_TOČAKA; ++točka)
{
radijus = randomFloat(ASTEROID_RAD_PLUS - ASTEROID_RAD_MINUS, AS-
TEROID_RAD_PLUS + ASTEROID_RAD_PLUS);
              float kut = kumulativniKutevi[točka];

              vector.brzina = radijus;
              vector.smjer = kut;

              Pozicija komponente;
              komponente.x = vector.brzina * cos(vector.smjer);
              komponente.y = vector.brzina * sin(vector.smjer);
              asteroid.točke.push_back(komponente);
       }

       Pozicija zadnjaTočka = početnaTočka;
       asteroid.točke.push_back(zadnjaTočka);

       return asteroid;
}
```

Funkcija **nacrtajAsteroid()** sadrži for-petlju koja prolazi kroz dinamički niz koji smo stvorili u programu vector<Asteroid> asteroids u koji se sprema svaki asteroid koji smo generirali. Za svaki asteroid poziva se funkcija nacrtajVektorskiOblik(), koja crta vektore spajanjem vrha jednog vektora s hvatištem drugog vektora.

```
void nacrtajAsteroid() {
       for (const auto& asteroid : asteroids) {
              nacrtajVektorskiOblik(asteroid);
       }
}
```

Kao što znamo, konstanta ASTEROID_BROJ_TOČAKA označava od koliko će se točaka asteroid sastojati. Slijedi primjer na Slici 10. za ASTEROID_BROJ_TOČAKA = 7 i ASTEROID_BROJ_TOČAKA = 9.

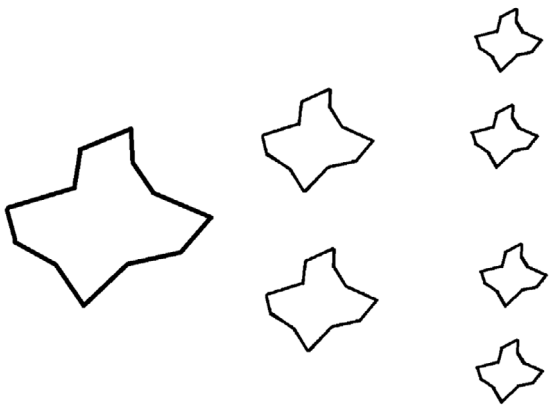


Slika 10. Primjer asteroida sa 7 i s 9 točaka

U igri, kada pogodimo asteroide laserskom zrakom, oni se moraju podijeliti na dva dvostruko manja asteroida jednakog oblika. Stoga nam preostaje prikazati skaliranje asteroida. Funkcija **explodeAsteroid()** kao ulazne parametre prima indeks asteroida koji se nalazi u vector<Asteroid> asteroids i referencu na sami asteroid koji je eksplodirao. Prije nego što obrišemo eksplodirani asteroid, spremamo njegovu trenutnu veličinu u privremenu varijablu **originalSkala** koja služi kao kopija izvorne vrijednosti skale**.** Originalna skala asteroida predstavlja njegovu trenutačnu veličinu prije eksplozije odnosno faktor skaliranja koji određuje dimenzije asteroida u igri. Ova vrijednost omogućuje dinamičko smanjivanje asteroida pri eksploziji, čime se održava konzistentan vizualni i mehanički prikaz podjele asteroida. Ako je originalna skala manja od 4, generiramo dva nova asteroida koji su dvostruko manji od izvornog. Ovaj prag osigurava da se asteroidi ne dijele beskonačno, već do određene minimalne veličine. Recimo da je vrijednost **originalSkala** jednaka 1. Petlja će se izvršiti dva puta, što znači da iz jednog asteroida stvara dva dvostruko manja (koristeći funkciju stvoriAsteroid() koji će se skalirati novom skalom **novaSkala**), te ćemo iz ta dva dobiti još dva dvostruko manja asteroida. Skalabilnost je na ovom primjeru vektorske grafike bitna jer nam pruža fleksibilnost u stvaranju asteroida različitih veličina bez potrebe za pisanjem posebnog koda za svaku veličinu. Osim skaliranja veličine, skalabilnost se odnosi na prilagodbu drugih svojstava asteroida poput brzine rotacije i smjera novih asteroida.

Brzinu kretanja, brzinu rotacije i smjer novih asteroida postavljamo na nasumičnu vrijednost. Pozicija novih asteroida postavlja se na poziciju eksplodiranih asteroida, te se novi asteroidi dodaju u vector<Asteroid> asteroids koristeći funkciju **push_back()**.

```
void explodeAsteroid(int index, Asteroid& asteroid) {
	Pozicija pozicija = asteroid.pozicija;
	float originalSkala = asteroid.razmjer;

	asteroids.erase(asteroids.begin() + index);

	if (originalSkala < 4) {
		float novaSkala = originalSkala * 2;

		for (int brojač = 0; brojač < 2; ++brojač) {
			Asteroid noviAsteroid = stvoriAsteroid(novaS-
kala);
			noviAsteroid.brzina.brzina = (static_cast<-
float>(rand()) / RAND_MAX) * (ASTEROID_MAX_BRZINA - ASTEROID_MIN_
BRZINA)+ ASTEROID_MIN_BRZINA;

			noviAsteroid.brzina.smjer = static_cast<-
float>(rand()) / RAND_MAX * (2 * _Pi);

			noviAsteroid.brzinaRotacije = (static_cast<-
float>(rand()) / RAND_MAX) * (2 * ASTEROID_MAX_ROT) - ASTEROID_MAX_
ROT;

noviAsteroid.pozicija = pozicija;

			asteroids.push_back(noviAsteroid);
		}
	}
}
```



Slika 11. Skaliranje asteroida

Na Slici 11. možemo vidjeti proces skaliranja asteroida nakon što eksplodiraju. Najveći asteroid ima početnu vrijednost **originalSkala = 1** (na slici lijevo). Nakon što ga se pogodi, ulazi u petlju i dijeli se na dva nova asteroida čija je nova skala **originalSkala = 2** (središnji asteroidi).

Nakon sljedećeg pogotka ovi se asteroidi ponovno dijele na još dva manja asteroida (na slici desno), pri čemu svaki dobiva vrijednost varijable **originalSkala = 4**. Budući da je prag za daljnje dijeljenje postavljen na **originalSkala < 4**, najmanji asteroidi više ne ulaze u petlju te, kada budu pogođeni, samo nestaju bez daljnjeg skaliranja (uništeni su), čime se osigurava da se asteroidi ne dijele beskonačno.

Zaključak

U ovom smo članku dali primjere kako su se iskoristile prednosti vektorske grafike u videoigri Asteroids za omogućavanje atraktivnog vizualnog iskustva. Upotreba matematičkih vektora u geometrijskom prikazu igre važna je za stvaranje različitih geometrijskih oblika i za njihovo jednostavno skaliranje. Kreativna primjena vektorske grafike u videoigrama široka je i naglašava njezinu važnost za stvaranje estetski privlačnih vizualnih elemenata.

Literatura:

1. Strmečki, T.: *Matematika 1*, Tehničko veleučilište u Zagrebu, Zagreb, 2021.
2. Gerlinger Marion, Vectornator, https://www.vectornator.io/blog/how-to-vectorize-an-image/, 2022.
3. Strmečki, T., Mičković, V., Rak, R., Pajković, L.: Primjena matrica u praksi 1 – Digitalna obrada fotografija, Poučak, 23., 92.; 68-79, 2022.
4. Vektorska grafika, Wikipedia, https://en.wikipedia.org/wiki/Vector_graphics
5. Lutkevich, Ben, TechTarget, https://www.techtarget.com/whatis/definition/vector-graphics, 2021.
6. Asteroids (video igra), Wikipedia, https://en.wikipedia.org/wiki/Asteroids_(video_game)
7. Mozolevska, Victoria; Kevuru Games, https://kevurugames.com/blog/vector-vs-raster-graphics-the-complete-guide-to-pros-cons-and-applications/, 2022.
8. Asteroids, Bytes n Bits, https://bytesnbits.co.uk/asteroids/, 2020.
9. Rouse, Margaret, TechoPedia, https://www.techopedia.com/definition/25853/vector-graphics-rendering, 2017.