IZRADA OPENGL 3D VIDEOIGRE

Prnić, Marin

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Marin Prnić

Z A V R Š N I R A D

Izrada OpenGL 3D videoigre

Split, rujan 2022.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Objektno programiranje

Z A V R Š N I R A D

Kandidat: Marin Prnić

Naslov rada: Izrada OpenGL 3D videoigre

Mentor: Ljiljana Despalatović, v. pred.

Split, rujan 2022.

SADRŽAJ

SAŽETAK

Završni rad bavi se razvojem računalne igre imena “Open Tetro3D”. Radi se o računalnoj igri koja je inspirirana izrazito popularnom igrom “Tetris” kojoj je cilj slaganje blokova različitih oblika, pokušavajući izbjeći njihovo nakupljanje na vrhu te time ostvariti što veći broj bodova. “Open Tetro3D” razvijen je s istom logikom na umu. Pri razvoju igre korišten je programski jezik C++, integrirano razvojno okruženje Visual Studio 2022 te najpoznatije aplikacijsko programsko sučelje za izradu grafike OpenGL. Cilj rada je što kvalitetnija izrada jednostavne ali zabavne 3D računalne igre.

Ključne riječi : računalna igra, Tetris, C++, Visual Studio, OpenGL

SUMMARY

3D OpenGL Game Development

This final paper deals with the development of a computer game called "Open Tetro3D". It is a computer game inspired by the extremely popular game "Tetris". The goal of the game is to arrange differently shaped blocks, trying to avoid their accumulation to the top, and thereby achieve the highest possible number of points. "Open Tetro3D" was developed with the same logic in mind. The game was developed using programming language C++, the integrated development environment Visual Studio 2022 and the most famous application programming interface OpenGL. The aim of this project is to make a simple, yet fun 3D computer game in the most quality way possible.

Keywords: computer game, Tetris, C++, Visual Studio, OpenGL

1. UVOD

Od samih početaka razvoja računala ljudi su pronalazili načine kako bi ih iskoristili u svrhu zabave. Tako su nastale prve računalne igre. Računalna igra predstavlja softver u obliku interaktivne multimedije namijenjen za zabavu, a igra se na računalu. Prve računalne igre s obzirom na tadašnju tehnologiju nisu bile praktične za širu uporabu, a neki od glavnih razloga tomu bili su vrijednost i veličina tadašnjih računala koja su često zauzimala čitave prostorije. Razvojem tehnologije računala su postala znatno manja i pristupačnija široj publici, uslijed čega je došlo i do sve većeg razvoja igara koje su među publikom postajale sve popularnije i unosnije. Jedna od tih igara je Tetris koji je upravo i inspiracija ovog projekta i završnog rada.

Tetris je nastao 1984. godine a osmislio ju je i razvio Alexey Pajitnov. Riječ Tetris dolazi od grčkog prefiksa *tetra*, koji dolazi od grčke riječi *téttares,* što označava broj 4 i engleske riječi *tennis* što je ujedno i Alexeyov najdraži sport. Ideja igre je da igrač rotira i slaže padajuće blokove različitih struktura pune horizontalne redove. Cilj igre je što duže moguće zadržat blokove od nakupljanja do vrha igraćeg polja. S istom takvom logikom na umu je osmišljena i ova igra pod nazivom Open Tetro3D odnosno završni rad.

Rad se sastoji od četiri međusobno povezanih poglavlja.

Drugo poglavlje objašnjava koje su tehnologije i na koji način korištene u izradi ovog rada. Definiraju se i analiziraju pojmovi i značaj korištenog programskog jezika C++, integrirano razvojno okruženje (engl. integrated development environment, IDE) Visual Studio 2022 i aplikacijsko programsko sučelje (engl. application programming interface, API) OpenGl.

Treće poglavlje odnosi se na praktični dio rada, a uključuje razradu koda, njegovu implementaciju i detaljno objašnjenje istog.

Četvrto i završno poglavlje sumira cjelokupni projekt i sadrži zaključna razmatranja proizašla iz rada.

2. KORIŠTENE TEHNOLOGIJE

Prilikom razvoja računalne igre Open Tetro3D korištene su različite tehnologije. Programski jezik koji je korišten je objektno orijentirani jezik C++, integrirano razvojno okruženje je Visual Studio 2022, a za prikazivanje 2D i 3D grafike primijenjeno je aplikacijsko programsko sučelje OpenGL.

2.1. C++

C++ je objektno orijentirani jezik opće namjene. Obuhvaća mogućnosti jezika viših i nižih razina, što ga čini jezikom srednje razine. Razvio ga je danski računalni znanstvenik Bjarn Stroustrup tijekom 80-ih godina prošlog stoljeća.

C++ nastao je kao proširenje na programski jezik C. U početku je samo dopunjavao C jezik svojstvima objektno orijentiranog programiranja pa mu je iz tog razloga originalno ime bilo C with classes (hrv. C s klasama). Bitne promjene koje su nastale razvojem ovog programskog jezika su svojstva *namespace*, *operator overloading* te *error and exception handeling*. Prati koncepte kao što su polimorfizam, nasljeđivanje, enkapsulacija te apstrakcija što olakšava procese razvoja i održavanja.

C++ koristi širok spektar podatkovnih tipova, funkcija i operatora. Kompilacijski je jezik što znači da se implementira pomoću kompajlera (eng. compiler) bez čega se nijedan C++ program ne može biti pokrenut. Kompajler prevodi izvorni u strojni kod te se potom izvodi.

S obzirom na svojstva programskog jezika C++, često se koristi za razvoj računalnih igara zbog čega je i odabran za razvoj računalne igre u sklopu ovog rada.

2.2. Visual Studio 2022

Visual Studio je integrirano razvojno okruženje koje je razvio Microsoft, a koristi se za razvoj različitih vrsta softvera kao što su web stranice, web aplikacije, web usluge, računalni programi i mobilne aplikacije.

Razvoj prve verzije pod kodnim nazivom "Boston" provodio se s idejom izrade određene vrste Microsoft Officea za programere. Godine 1997. lansiran je prvi Visual Studio po nazivom Visual Studio 97. Sadržavao je ažurirane verzije tada već poznatih alata kao što su Visual Basic, Visual C++ i Visual FoxPro te nove alate kao što su Visual InterDev za web razvoj i Visual J++ (Microsoftovo Java okruženje).

Visual Studio sadrži svojstva razvoja (upravljanje, pisanje i popravljanje koda), debugiranja (uočavanje pogrešaka), testiranja (pisanje koda korištenjem alata za testiranje), suradnje (korištenje *version control*) te proširivanja (izbor različitih proširenja za prilagodbu integracijskog razvojnog okruženja).

Visual Studio koristi različite Microsoftove platforme za razvoj softvera. Neke od njih su Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight.

Uključuje uređivač koda koji podržava IntelliSense kao i refaktoriranje koda. Neki od ugrađenih alata su alat za profiliranje koda, dizajner za izradu GUI aplikacija, web dizajner, dizajner klasa i dizajner sheme baze podataka. Prihvaća dodatke koji proširuju funkcionalnost.

Visual Studio podržava 36 različitih programskih jezika, a uz to podržava i ispravljanje pogrešaka za iste jezike. Neki od tih programskih jezika su C, C++, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS.

2.3. OpenGL

OpenGL je višejezični i višeplatformski API, odnosno softverska biblioteka punog naziva *Open Graphic Language*, što znači da je riječ o otvorenom i svima dostupnom grafičkom jeziku. Koristi se za prikazivanje 2D i 3D vektorske grafike. To postiže izravnom komunikacijom s hardverom sustava (posebice grafičkom procesorskom jedinicom) kako bi se postigla hardverska akceleracija. Neovisan je o hardverskom sučelju, a može se implementirati

na različitim vrstama grafičkog hardvera ili u potpunosti korištenjem softvera ukoliko grafički hardver nije dostupan.

Prva verzija OpenGL-a nastala je 1992. godine, a razvili su je Mark Segal i Kurt Akeley. Osmišljena je kao otvorena alternativa IrisGL-u, API-u tvrtke Silicon Graphics. Za razliku od IrisGL-a, OpenGL nije ovisan o operativnom sustavu niti je vezan uz pojedinačnog proizvođača. Nije ovisan niti o prozorskom sustavu pa ne uključuje funkcije vezane za rad s prozorima, kao ni funkcije za obradu korisničkog unosa. Ne pruža funkcionalnost za opisivanje modela trodimenzionalnih objekata, niti operacije za čitanje slika (JPEG datoteke), već je potrebno konstruirati trodimenzionalne objekte iz malog skupa geometrijskih primitiva (točka, linija, poligon).

OpenGL karakteriziran je stabilnošću, pouzdanošću i prenosivosti te lakoćom primjene. Dopune i izmjene u standardu OpenGL-a razvijaju se održavajući kompatibilnost s ranijim verzijama. Neovisno o operativnom sustavu, aplikacije pisane koristeći OpenGL izgledaju isto te se mogu izvršavati na raznim uređajima uključujući računala, radne stanice i superračunala. Struktura i jednostavno sučelje omogućavaju da se s manje linija koda postigne jednak učinak kao i s ostalim grafičkim bibliotekama.

OpenGL predstavlja stroj stanja (engl. *State machine*), što znači da se uneseno stanje neće mijenjati dok to ne bude eksplicitno promijenjeno. Primjerice, ukoliko definiramo boju nekog objekta, takvi objekti iscrtavat će se u toj boji sve dok je ne promijenimo.

Neke od OpenGL naredbi su primjerice crtanje poligona, dodjeljivanje boja raznim oblicima, primjenu tekstura na poligone (preslikavanje tekstura), povećavanje i smanjivanje, transformiranje poligona, rotiranje objekata itd.

Također se koristi za upravljanje svjetlosnim efektima (izvori svjetlosti, sjenčanje i sjene), a može i stvoriti efekte poput izmaglice i magle koji se mogu primijeniti na određeni objekt ili pak cijelu scenu.

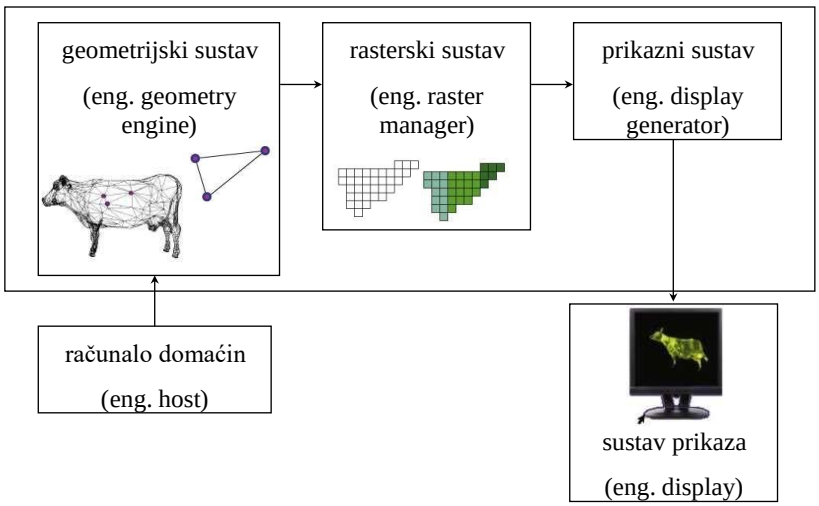
Neke od ugrađenih mogućnosti unutar OpenGL-a uključuju *alpha blending* (transparentnost), *antialiasing* (uglađivanje oštrih rubova), mapiranje teksture, upravljanje pikselima, transformacije u promatranju i modeliranju te već spomenute atmosferske efekte (magla, dim, izmaglica).

2.3.1. Grafički cjevovod

Kako bi se bolje mogla razumjeti stanja grafičkog cjevovoda potrebno je definirati *shader*. Shaderi su računalni programi koji služe kako bi se moglo definirati i "kazati" računalu na koji način da prikaže svaki pojedini piksel. "Komuniciraju" na način da međusobno šalju i primaju podatke. Za OpenGL koristi se GLSL, što je akronim za *openGL Shading Language*. GLSL ima sintaksu sličnu programskom jeziku C koja je vidljiva na slici 1.

Slika 1. Primjer GLSL shadera sa pripadajućim prikazom

Svako crtanje u OpenGL-u odvija se preko grafičkog cjevovoda (engl. graphics pipeline) koji je prikazan na slici 2. Pojam grafičkog cjevovoda odnosi se na niz koraka, odnosno faza koje su potrebne prilikom ostvarivanja prikaza.



Slika 2. Prikaz grafičkog cjevovoda

Dijeli se na tri cjeline: geometrijski sustav, rasterski sustav i prikazni sustav.

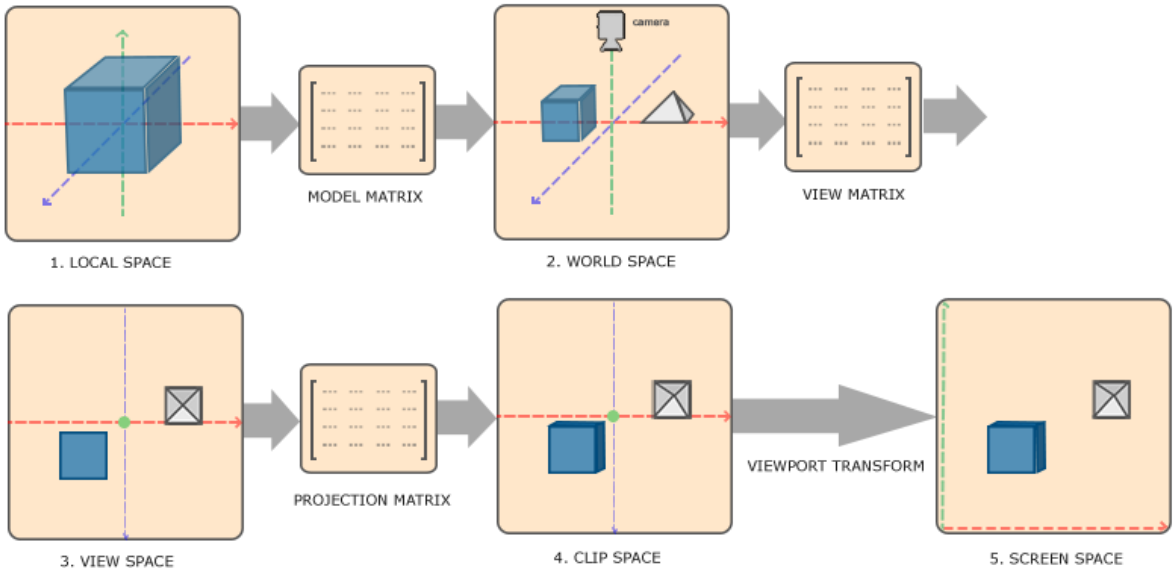
Prilikom pokretanja grafičke aplikacije, geometrijskom sustavu šalju se dijelovi vezani za grafiku, od kojih se pritom izdvajaju vrhovi. Nad njima se vrše transformacije koordinata, kao i računanje osvjetljenja objekta. Za ovu funkciju zadužen je *vertex shader*. *Vertex shader* je mali program koji se izvodi na GPU i procesira vrhove primitive. Nakon što *vertex shader* procesira vrhove, vrhovi koji su inicijalno bili u zadani 3D prostoru bit će transformirani u 2D projekcije.

Rasterski sustav bavi se *rasterizacijom* poligona i linija. Proces *rasterizacije* odnosi se na postupak pretvaranja slike u skup točaka (piksela). Radi procesa *rasterizacije* dolazi do neželjenog učinka (engl. ali-asing) koji se kod prikaza objekata primarno manifestira u obliku oštrih rubova. Ovaj učinak je nemoguće izbjeći u potpunosti, ali s povećanjem rezolucije je sve manje uočljiv. Tijekom *rasterizacije* moguće je pridruživati teksture i boje objektima. Nakon *rasteriziranja* prikaz je potrebno proslijediti *frame bufferu* iz kojeg se slika iščitava kako bi bila ispisana na ekranu.

Prikazani sustav odnosi se na uređaj za prikazivanje slike. Za ostvarivanje konačnog prikaza potrebno je osigurati podršku za različite standarde koje prihvaća uređaj za prikazivanje (VGA, HDMI, DVI) ili podršku za više sustava prikaza.

2.3.2. Prostor

Očekivani unos za 2D koordinate je x,y, a za 3D koordinate je x,y,z. Prostor koordinatnog sustava za crtanje definiran je granicama od -1.0f do 1.0f . Slovo f označava da se radi o formatu *float* što znači da se radi o decimalnom zapisu. Slika 3 prikazuje različite koordinatne prostore koji se koriste prilikom poziva crtanja vrhova.



Slika 3. Prikaz različitih koordinatnih sustava

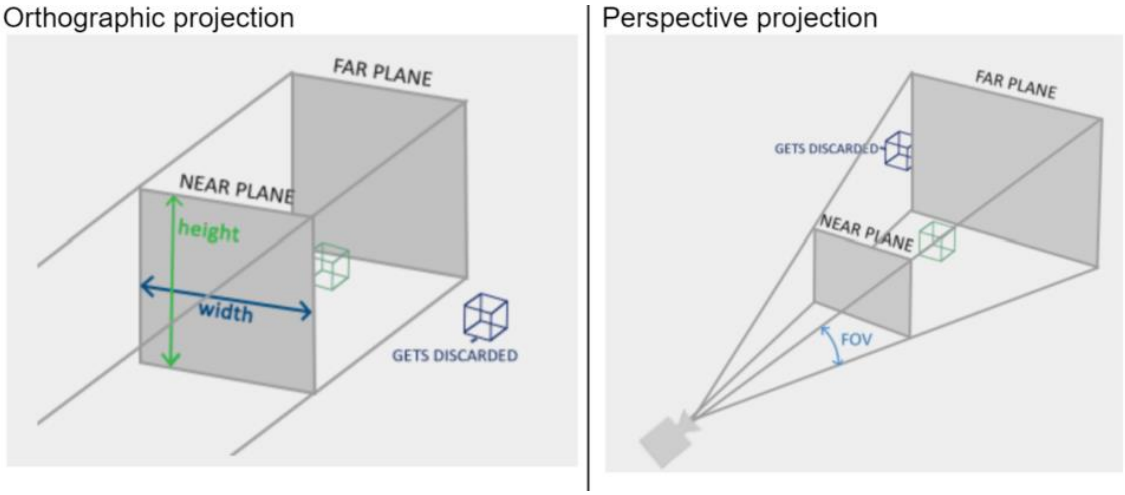
1. Local space – koordinatni prostor u kojemu se nalazi objekt, definira početne točke objekta.
2. World space – koordinatni prostor cjelokupne igre ili aplikacije, definira položaj objekta u prostoru. *Model matrix* je matrica koja rotira, translatira i skalira objekt kako bi bio smješten na svoje pripadajuće mjesto.
3. View space – "Kamera", rezultat je transformiranja "world space" koordinata u koordinate koje korisnik vidi, tj. koje "kamera" vidi. To znači da se "svijet" kreće oko "kamere" umjesto obrnuto.
4. Clip space – Kako bi koordinate bile unutar očekivanih granica (-1.0f do 1.0f), potrebno je normalizirati koordinate svih objekata. To se radi primjenjivanjem *projection matrixa*. Sve koordinate koje su izvan definiranih koordinata *projection matrixa* "režu" se, odnosno neće se prikazati.
5. Screen space – Kako bi se prikaz bio ispravan, koordinate *clip spacea* se pretvaraju u koordinate prikaza.

2.3.3. Projekcije

Metode koje opisuju što vidimo i način na koji to vidimo često se zovu projekcije. Naziv projekcija dolazi od činjenice da objekte unutar 3D prostora “projiciramo” na 2D podlogu. Postoje dvije vrste projekcija, ortografska i perspektivna projekcija, prikazani su na slici 4 .

Ortografska projekcija izravno preslikava koordinate na 2D ravninu, što rezultira nerealnom slikom jer ne uzima u obzir perspektivu. Koristi se za prikaz 2D ravnina.

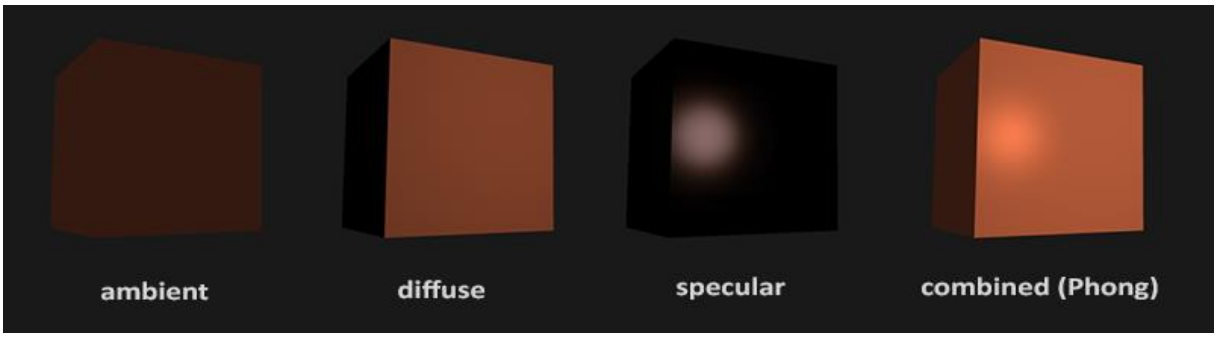
Kao što samo ime nalaže, perspektivna projekcija uzima u obzir faktor perspektive i kao rezultat toga prikazuje realističan prikaz objekata. Što su objekti udaljeniji to će i na ekranu izgledati udaljenije. Efekt perspektive se postiže na način da se projekcija stavlja pod dodatni kut kojim se dočarava dubina i udaljenost. Zbog dodanog efekta udaljenosti i dubine koristi se za prikaze objekata u 3D prostoru.



Slika 4. Prikaz različitih sustava projekcije

2.3.4. Osvjetljenje

Osvjetljenje u OpenGL-u temelji se na aproksimacijama svjetlosti u stvarnom svijetu koristeći pojednostavljene modele koje je lakše obraditi i koji izgledaju slično svjetlosti u stvarnom svijetu. Jedan od tih modela zove se Phongov model rasvjete. Sastoji se od tri komponente: ambijentalne, difuzne i reflektirajuće rasvjete te prikazani su na slici 5.



Slika 5. Prikaz različitih tipova osvjetljenja

Ambijentalno svjetlo je modelirano kao svjetlo koje nema izvor ni smjer te ima jednak efekt na sve objekte u sceni.

Difuzno osvjetljenje određeno je smjerom izvora svjetlosti. Ono stvara efekt nestajanja udaljavanjem od izvora svjetlosti. Strana koja je okrenuta direktno izvoru svjetla snažnije je osvijetljena.

Reflektirajuće svjetlo je direktan odraz svjetla usmjeren prema oku promatrača. Ovisi o koeficijentu sjajnosti. Što je materijal sjajniji refleksija je jača.

Vrste svjetlosti su:

- Direkcijsko svjetlo - svjetlo koje nema svoju poziciju ili izvor. Svo svjetlo dolazi iz paralelnih zraka iz naočigled beskrajne udaljenosti. Primjer direkcijskog svjetla je sunce.

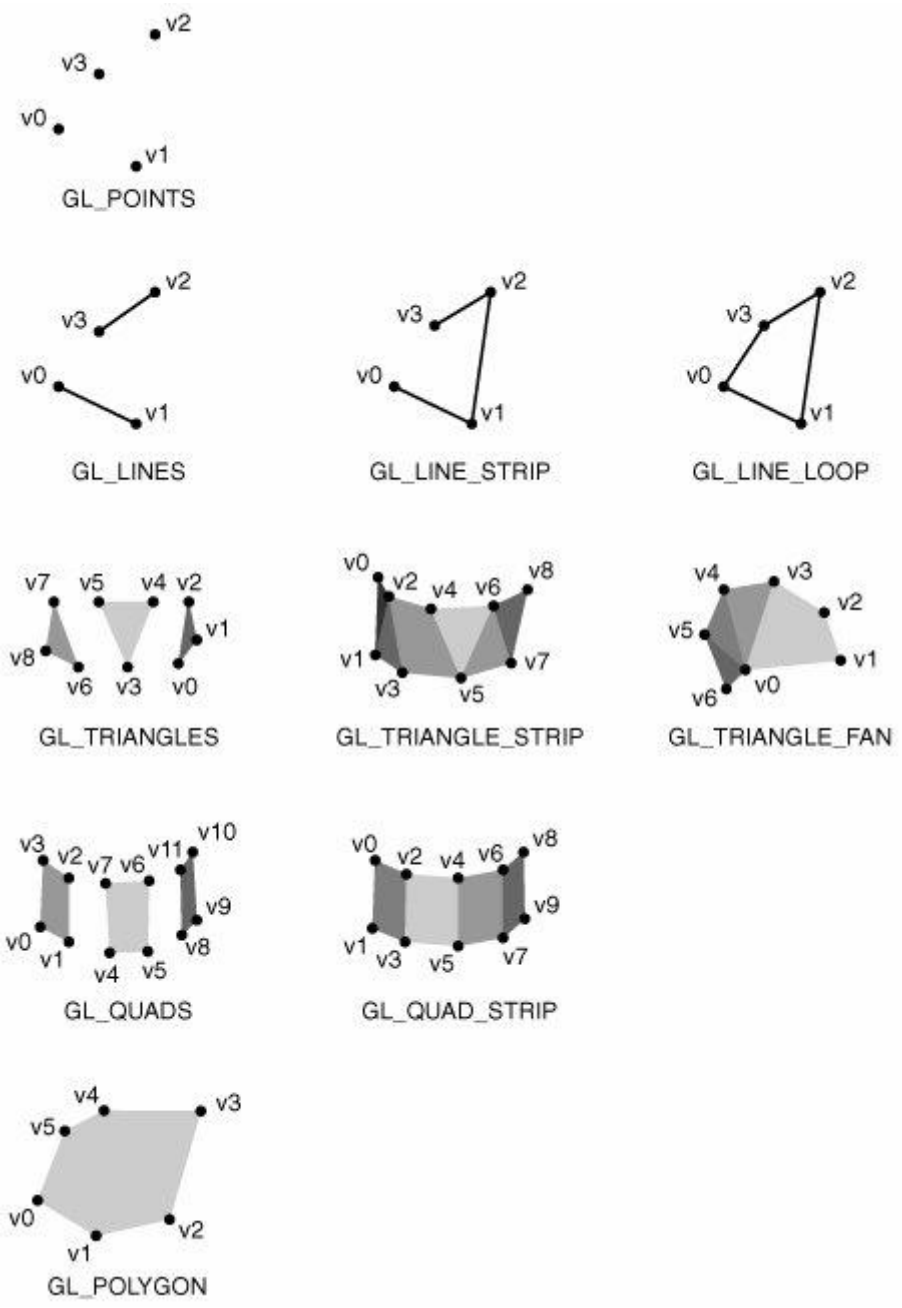
- Točkasti izvor svjetlosti - svjetlo s položajem koje svijetli u svim smjerovima, a njegov najbolji primjer je žarulja.

- Usmjereno svjetlo - slično točkastom svjetlu, no ovo je "odsječeno" kako bi emitiralo u određenom rasponu i pod određenim kutom, kao npr. ručna svjetiljka.

- Prostorno svjetlo - napredniji oblik svjetla koje emitira svjetlost iz nekakvog područja, a jedan primjer je osvjetljeni panel na stropu ili zidu.

2.3.5. Primitivni tipovi (primitivi)

OpenGL podržava nekoliko osnovnih primitivnih tipova, uključujući točke, linije, četverokute i poligone. Svi primitivni tipovi specificirani su pomoću niza vrhova i prikazani su na slici 6.



Slika 6. Prikaz različitih primitivnih tipova

Slika 6. prikazuje osnovne primitivne tipove, a brojevi označavaju redoslijed kojim su navedeni vrhovi.

Bitno je istaknuti kako za primitivni tip GL_LINES samo svaki drugi vrh uzrokuje crtanje segmenta linije. Slično tome, za primitivni tip GL_TRIANGLES, svaki treći vrh uzrokuje crtanje trokuta. Primitivni tip GL_TRIANGLE_STRIP i GL_TRIANGLE_FAN proizvodi novi trokut za svaki dodatni vrh.

3. IMPLEMENTACIJA IGRE

3.1. Inicijalizacija projekta koristeći Visual Studio 2022

Kao što je već prethodno navedeno, za izradu igre Open Tetro 3D korišten je programski jezik C++, integrirano razvojno okruženje Visual Studio 2022 i aplikacijsko programsko sučelje OpenGL s alatom za kreiranje konteksta i prozora freeglut.

Potrebno je napraviti novi projekt, što se radi na način da se na alatnoj traci klikne na File > New > Project > Templates > Visual C++ > Empty Project. Nakon toga potrebno je dodijeliti ima projekta i odredišnu mapa.

3.2. Inicijalizacija freegluta

Kako bi se mogao koristiti freeglut prvo ga je potrebno inicijalizirati.

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);
glutInitWindowPosition(100, 100);
glutInitWindowSize(VPWIDTH + VPWIDTH/2 + 2 * margins, VPHEIGHT + 2 * margins);
glutCreateWindow("Open Tetro 3D");
```

Ispis 1: Prikaz inicijalizacije freegluta

Funkcija `glutInit` prikazana je u ispisu 1 te se koristi kako bi se inicijalizirala GLUT knjižnica. Funkcija `glutInitDisplayMode` postavlja inicijalne parametre prikaza. U ovom slučaju to su `GLUT_RGBA` koji definira korišteni *colorspace*, `GLUT_DEPTH` koji služi kako bi se mogle koristiti mogućnosti prikaza dubine, dok `GLUT_DOUBLE` definira korištenje dva *buffera*. Za određivanje veličine prikazanog prozora koristi se funkcija `glutInitWindowSize`, dok se za kreiranje prozora i dodjeljivanje imena istom prozoru koristi `glutCreateWindow`.

3.3. Ispis teksta

Kako bi se mogao ispisati tekst na ekranu, potrebno je osmisliti i implementirati način da se isto ostvari.

```
void BitmapText(char* str, int wcx, int wcy)
{
	glRasterPos2i(wcx, wcy);
	for (int i = 0; str[i] != '\0'; i++) {
		glutBitmapCharacter(GLUT_BITMAP_9_BY_15, str[i]);
	}
}
```

Ispis 2: Prikaz funkcije za ispis teksta

Ispis 2 prikazuje funkciju `BitmapText`, koja kao parametar prima tekst koji je potrebno ispisati te koordinate na kojima će se taj tekst nalaziti. Za pozicioniranje u prostoru koristi se `glRasterPos2i`, dok se za ispisivanje samog teksta koristi funkcija `glutBitmapCharacter`.

3.4. Brzina igre

Alat freeglut posjeduje *callback* funkciju `glutTimerFunc` koja se iznova aktivira nakon određenog vremenskog perioda.

```
void timer(int id)
{
	if (game.killed) {

		game.paused = true;
		game.timer = 1000;
		game.clearMainGrid();
		game.clearNextPieceGrid();
	}
	else if (!game.paused) {
		game.update();
		switch (game.level) {
		case 1:
			mciSendString(L"play lvl1 repeat", NULL, 0, NULL);
			game.timer = 1000;
			break;
```

```
            case 2:
                  game.timer = 900;
                  break;
            case 3:
                  game.timer = 800;
                  mciSendString(L"stop  lvl1", NULL, 0, NULL);
                  mciSendString(L"seek lvl1 to start", NULL, 0, NULL);
                  mciSendString(L"play lvl2 repeat", NULL, 0, NULL);
                  break;
            case 4:
                  game.timer = 700;
                  break;
            case 5:
                  game.timer = 600;
                  mciSendString(L"stop  lvl2", NULL, 0, NULL);
                  mciSendString(L"seek lvl2 to start", NULL, 0, NULL);
                  mciSendString(L"play lvl3 repeat", NULL, 0, NULL);
                  break;
            case 6:
                  game.timer = 500;
                  break;
            case 7:
                  game.timer = 400;
                  mciSendString(L"stop  lvl3", NULL, 0, NULL);
                  mciSendString(L"seek lvl3 to start", NULL, 0, NULL);
                  mciSendString(L"play lvl4 repeat", NULL, 0, NULL);
                  break;
            case 8:
                  game.timer = 300;
                  break;
            case 9:
                  mciSendString(L"stop  lvl4", NULL, 0, NULL);
                  mciSendString(L"seek lvl4 to start", NULL, 0, NULL);
                  mciSendString(L"play lvl5 repeat", NULL, 0, NULL);
                  game.timer = 200;
                  break;
            case 10:
                  game.timer = 100;
                  break;

            }
            glutTimerFunc(game.timer, timer, 0);
      }
      glutPostRedisplay();
}
```

Ispis 3: Prikaz funkcije za brzinu igre

Funkcija `timer` provjerava u kojem je stanju igra te je prikazana u ispisu 3. Dok je igra aktivna, provjerava trenutni nivo igre te sukladno tome prilagođava brzinu igre i glazbu koja se koristi,

dok se u slučaju završetka igre, odnosno "gubitka", brzina igre vraća na početnu brzinu, a igrače polje se raščisti. Funkcija `mciSendString` koristi se za reprodukciju glazbe dok se funkcija `game.update` koristi kako bi se ažuriralo stanje igre, što će biti objašnjeno u kasnijem poglavlju. Kako bi nakon svake promjene ista bila vidljiva, koristi se funkcija `glutPostRedisplay` kako bi se ponovno ispisao trenutni prozor.

3.5. Unos korisnika

Kako bi bilo moguće koristiti tipkovnicu prilikom upravljanja igre, implementirana je funkcija `keyboard`, prikazana je u ispisu 4.

```
void keyboard(unsigned char key, int x, int y)
{
	if (game.paused && game.killed) {
		if (key == 13) {
			game.killed = false;
			game.restart();
			if (game.level < 3)
				mciSendString(L"play lvl1 repeat", NULL, 0, NULL);
			else if (game.level < 5)
				mciSendString(L"play lvl2 repeat", NULL, 0, NULL);
			else if (game.level < 7)
				mciSendString(L"play lvl3 repeat", NULL, 0, NULL);
			else if (game.level < 9)
				mciSendString(L"play lvl4 repeat", NULL, 0, NULL);
			else if (game.level > 9)
				mciSendString(L"play lvl5 repeat", NULL, 0, NULL);

			glutTimerFunc(game.timer, timer, 0);
		}
	}
	else {
		if (key == 27) {
```

```
                if (game.level < 3)
                        mciSendString(L"play lvl1 repeat", NULL, 0, NULL);
                else if (game.level < 5)
                        mciSendString(L"play lvl2 repeat", NULL, 0, NULL);
                else if (game.level < 7)
                        mciSendString(L"play lvl3 repeat", NULL, 0, NULL);
                else if (game.level < 9)
                        mciSendString(L"play lvl4 repeat", NULL, 0, NULL);
                else if (game.level > 9)
                        mciSendString(L"play lvl5 repeat", NULL, 0, NULL);
                game.paused = !game.paused;
                glutTimerFunc(game.timer, timer, 0);
        }
        if (key == 'r') {
                mciSendString(L"stop  lvl1", NULL, 0, NULL);
                mciSendString(L"stop  lvl2", NULL, 0, NULL);
                mciSendString(L"stop  lvl3", NULL, 0, NULL);
                mciSendString(L"stop  lvl4", NULL, 0, NULL);
                mciSendString(L"stop  lvl5", NULL, 0, NULL);

                mciSendString(L"seek lvl1 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl2 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl3 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl4 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl5 to start", NULL, 0, NULL);
                game.restart();
        }
    }
    glutPostRedisplay();
}
```

Ispis 4: Prikaz funkcije za unos tipkovnice

Funkcija provjerava stanje igre. Ovisno o tome, različite tipke izvršavaju različite funkcije. Nakon što je korisnik "izgubio" igru, tipkom "enter" (key == 13) započinje se nova igra, dok se tipka "escape" (key == 27) koristi kako bi korisnik mogao pauzirati i nastaviti igru. Dok je igra pauzirana korisnik može i započeti igru od početka pritiskom na tipku "r" (key == r).

Tipke kako što su "gore", "dolje", "lijevo" i "desno" spadaju pod tipke posebnog unosa pa je za to implementirana funkcija `special`, prikazana u ispisu 5.

```
void special(int key, int x, int y)
{
	if (!game.paused && !game.killed) {
		if (key == GLUT_KEY_LEFT) {
			game.move(-1);
		}
		else if (key == GLUT_KEY_RIGHT) {
			game.move(1);
		}
		else if (key == GLUT_KEY_UP) {
			game.rotateShape(1);
		}
		else if (key == GLUT_KEY_DOWN) {
			game.update();
		}
	}
	glutPostRedisplay();
}
```

Ispis 5: Prikaz funkcije za posebne unose

Pritiskom na tipke "lijevo" ili "desno", aktivni oblik će se pomaknuti u željenom smjeru na način da se funkciji `move` proslijedi parametar kojim definiramo smjer kretanja po x osi. Tipka "gore" koristi se kako bi se rotirao aktivni oblik, dok tipka "dolje" povećava brzinu padanja aktivnog oblika na način da ažurira stanje igre prilikom čega se aktivni komad spušta dok ne dođe do uklapanja.

3.6. Prikaz na ekran

Kako bi ispis na ekranu bio pravilan, alat freeglut koristi funkciju `glutDisplayFunc` koja za parametar prima funkciju prema kojoj će izvršavati promjene prikaza na ekranu. Sljedeći segmenti koda dio su funkcije `display` koja se prosljeđuje funkciji `glutDisplayFunc` kao parametar.

3.6.1. Prikaz igračeg polja

Igraće polje je jednostavno i sastoji se od prozirnih kocki. Implementirano je na način koji je opisan u nastavku i prikazan u ispisu 6.

```
for (int i = 0; i < COLS; i++) {
      for (int j = 0; j < ROWS; j++) {
            glPushMatrix();
            glColor4f(1.0, 1.0, 1.0, 0.5);
            glTranslatef(.5 + i, .5 + j, 1);
            glutWireCube(1);
            glPopMatrix();
      }
}
```

Ispis 6: Prikaz koda za ispis igraćeg polja

Prolazeći po redovima i stupcima igraćeg polja, svakoj kocki se dodjeljuje boja koristeći `glColor4f`, koji prima 4 parametra pomoću kojih se dodjeljuje pripadajuća boja. Format boje je *RGBA*, što je akronim za *red blue green* dok zadnji parametar označava *alpha* odnosno prozirnost. Funkcija `glTranslatef` koristi se kako bi svaki blok bio postavljen na svoje mjesto. Radi to na način da se unesu x,y,z i koristeći te koordinate izvrši se pomak. Funkcija `glutWireCube` se koristi kako bi se nacrtala prazna kocka. Kako i samo ime nalaže, kocka će biti sastavljena od "žice". Funkcija kao parametar prima željenu veličinu kocke.

3.6.2. Prikaz aktivnih oblika

Aktivni oblik sastoji se od četiri kvadrata koji su inspirirani *tetrominoima* iz popularne igre "Tetris". Implementiran je na način prikazan u ispisu 7.

```
for (int r = 0; r < ROWS; r++) {
```

```
      for (int c = 0; c < COLS; c++) {
            Square& square = game.mainGrid[r][c];
            if (square.isFilled) {
                  glPushMatrix();
                  glColor3f(square.red, square.green,square.blue);
                  glTranslatef(c +.5, r+.5, 1.0f);
                  glutSolidCube(.8);
                  glPopMatrix();
            }
      }
}
```

Ispis 7: Prikaz koda za ispis aktivnog oblika

Provjeravaju se svi redovi i stupci. Ukoliko je blok potrebno "ispuniti", koristi se funkcija `glutSolidCube` koja za parametar prima željenu veličinu kocke. Kao što ime nalaže, kocka će biti puna, što znači da će svi bridovi biti u dodijeljenoj boji. Isto tako, dodjeljuje mu se boja ovisno o obliku.

3.6.3. Prikaz sljedećih aktivnih oblika

Ova funkcija slična je prikazu aktivnog oblika, uz razliku što se ne koristi 3D objekt, već 2D objekt. Prikazana je ispisom 8.

```
for (int r = 1; r < 5; r++) {
            for (int c = 0; c < 2; c++) {
                  Square& square = game.nextPieceGrid[r][c];
                  if (square.isFilled) {
                        glColor3f(square.red, square.green, square.blue);
                        glRectd(c + .1, r + .1, c + .9, r + .9);
                  }
            }
```

```
    }
```

Ispis 8: Prikaz koda za ispis sljedećeg aktivnog bloka

Prikazani kod identičan je kao i onaj prethodni, s razlikom što je sljedeći aktivni blok 2D objekt pa se koristi funkcija `glRectd` kako bi se ispisao na ekranu. Funkcija `glRectd` prima četiri parametra u formatu x1,y1,x2,y2 te ih koristi kako bi se nacrtao kvadrat. Koordinate x1 i y1 definiraju donju lijevu točku dok koordinate x2 i y2 definiraju gornju desnu točku kvadrata.

3.6.4. Prikaz stanja igre u pauzi i pri završetku igre

U trenutku pauziranja igre na ekranu se prikaže tekst "GAME PAUSED", dok se u trenutku završetka igre na ekranu prikazuje tekst "GAME OVER" te je ponuđena opcija za ponovni početak igre. Implementacija je prikazana u ispisu 9.

```
if (game.paused && !game.killed) {
                  glColor3f(1, 1, 1);
                  sprintf_s(msg, N, "GAME PAUSED");
                  BitmapText(msg, 140, VPHEIGHT / 2);
                  mciSendString(L"pause lvl1", NULL, 0, NULL);
                  mciSendString(L"pause lvl2", NULL, 0, NULL);
                  mciSendString(L"pause lvl3", NULL, 0, NULL);
                  mciSendString(L"pause lvl4", NULL, 0, NULL);
                  mciSendString(L"pause lvl5", NULL, 0, NULL);


            }
            if (game.paused && game.killed) {
                  glColor3f(1, 1, 1);
                  sprintf_s(msg, N, "GAME OVER");
                  BitmapText(msg, 155, VPHEIGHT / 2 + 50);
                  sprintf_s(msg, N, "YOUR SCORE: %d", game.score);
                  BitmapText(msg, 140, VPHEIGHT / 2);
                  sprintf_s(msg, N, "Press [ENTER] to restart ...");
                  BitmapText(msg, 75, VPHEIGHT / 2 - 100);
```

```
                mciSendString(L"stop  lvl1", NULL, 0, NULL);
                mciSendString(L"stop  lvl2", NULL, 0, NULL);
                mciSendString(L"stop  lvl3", NULL, 0, NULL);
                mciSendString(L"stop  lvl4", NULL, 0, NULL);
                mciSendString(L"stop  lvl5", NULL, 0, NULL);

                mciSendString(L"seek lvl1 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl2 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl3 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl4 to start", NULL, 0, NULL);
                mciSendString(L"seek lvl5 to start", NULL, 0, NULL);
            }
```

Ispis 9: Prikaz koda za ispis za ispis teksta prilikom pauze ili završetka igre

Vrši se provjera u kojem je stanju igra. Koristeći se funkcijom `sprintf_s`, varijabla `msg` koristi se kao *buffer* u koji se sprema ispis teksta. Taj isti *buffer* prosljeđuje se već prethodno analiziranoj funkciji `BitmapText` kako bi se ispisao tekst. U slučaju da je igra završena zaustavlja glazbu.

3.6.5. Reproduciranje glazbe

```
mciSendString(L"open \"lvl1.wav\" type mpegvideo alias lvl1", NULL, 0,
NULL);
mciSendString(L"play lvl1 repeat", NULL, 0, NULL);
mciSendString(L"stop lvl1", NULL, 0, NULL);
mciSendString(L"seek lvl1 to start", NULL, 0, NULL);
```

Ispis 10 : Prikaz funkcija za reproduciranje i manipuliranje glazbe

Funkcija `mciSendString`, prikazana u ispisu 10, koristi se kako bi se reproducirala glazba. Slanjem parametra "open" definira se da je u pitanju otvaranje datoteke definiranog tipa kojoj se pridodjeljuje ime parametrom "alias". Parametar "play" koristi se kako bi se reproducirala glazba pod datim imenom. Slično parametru "play", parametar "stop" koristi se kako bi se zaustavila reprodukcija glazbe. Kako bi se reproducirana glazba vratila na određeni dio koristi se parametar "seek".

3.7. Logika igre

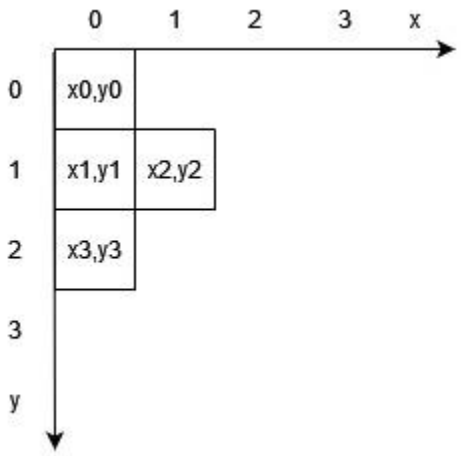
U nastavku slijedi prikaz aktivnih oblika, prikazan je u ispisu 11.

```
const int gamePieces[numPieces][numRotations][numSpaces] =
{
	{
		{0, 0, 1, 0, 0, 1, 1, 1},
		{0, 0, 1, 0, 0, 1, 1, 1},
		{0, 0, 1, 0, 0, 1, 1, 1},
		{0, 0, 1, 0, 0, 1, 1, 1},
	},
	{
		{0, 0, 0, 1, 0, 2, 0, 3},
		{0, 0, 1, 0, 2, 0, 3, 0},
		{0, 0, 0, 1, 0, 2, 0, 3},
		{0, 0, 1, 0, 2, 0, 3, 0},
	},
	{
		{0, 0, 0, 1, 1, 1, 0, 2},
		{1, 0, 0, 1, 1, 1, 2, 1},
		{0, 1, 1, 0, 1, 1, 1, 2},
		{0, 0, 1, 0, 2, 0, 1, 1}
	},
	{	{0, 0, 1, 0, 0, 1, 0, 2},
```

```
            {0, 0, 0, 1, 1, 1, 2, 1},
            {1, 0, 1, 1, 0, 2, 1, 2},
            {0, 0, 1, 0, 2, 0, 2, 1}
      },
      {     {0, 0, 1, 0, 1, 1, 1, 2},
            {0, 0, 1, 0, 2, 0, 0, 1},
            {0, 0, 0, 1, 0, 2, 1, 2},
            {2, 0, 0, 1, 1, 1, 2, 1}
      },
      {     {0, 0, 0, 1, 1, 1, 1, 2},
            {1, 0, 2, 0, 0, 1, 1, 1},
            {0, 0, 0, 1, 1, 1, 1, 2},
            {1, 0, 2, 0, 0, 1, 1, 1}
      },
      {     {1, 0, 0, 1, 1, 1, 0, 2},
            {0, 0, 1, 0, 1, 1, 2, 1},
            {1, 0, 0, 1, 1, 1, 0, 2},
            {0, 0, 1, 0, 1, 1, 2, 1}
      }
```

Ispis 11 : Prikaz koda aktivnih oblika

Svaki oblik definiran je na način da je prikazan matricom svojih koordinata zajedno sa svim svojim rotacijama. Svi oblici su definirani u formatu x0,yo,x1,y1,x2,y2,x2,y3 koji se koriste prilikom ispisa oblika. Primjerice, oblik "T" definiran je koordinatama `{0,0,0,1,1,1,0,2}` i njegova struktura bit će prikazana u nastavku na slici 7.



Slika 7. Prikaz oblika "T" u koordinatnom sustavu

Prateći već prethodno definirani format iscrtan je pripadajući oblik.

3.7.1. Provjera redova

Kako bi igra mogla funkcionirati nesmetano, svaki ispunjeni red trebalo bi izbrisati, a da bismo mogli izbrisati red prvo moramo provjeriti je li red ispunjen. Provjera redova implementirana je na način prikazan u ispisu 12.

```
void Game::checkLine() {
      int curr_level = level;
      int fullRows = 0;
      for (int r=0; r<ROWS; r++) {
            bool fullRow = false;
                  for (int c=0; c<COLS; c++) {
                        Square &square = mainGrid[r][c];
                        if (square.isFilled){
                              fullRow = true;
                        }
                        else {
                              fullRow = false;
```

```
                        break;
                    }
                }
        if(fullRow) {
                for ( int c =0; c < COLS; c++){
                    mainGrid[r][c].toBeDeleted = true;
                }
                deleteLines = true;
                linesCleared++;
                fullRows++;
        }
        else
        {
                mciSendString(L"seek tup to start", NULL, 0, NULL);
                mciSendString(L"play tup", NULL, 0, NULL);
        }
}
switch (fullRows)
{
        case 1:
                score += 100;
                break;
        case 2:
                score += 200;
                break;
        case 3:
                score += 400;
                break;
        case 4:
                score += 500;
                break;
}
```

```
      if (score >= curr_level * 1000 && score <10000)
      {
            mciSendString(L"seek new_level to start", NULL, 0, NULL);
            mciSendString(L"play new_level", NULL, 0, NULL);
            level++;
      }
}
```

Ispis 12 : Prikaz funkcije za provjeru redova

U ovom slučaju provjeravaju se svi blokovi u svakom redu zasebno. Ukoliko je red ispunjen, označava se za brisanje i povećava se broj izbrisanih redova. Varijabla `fullRows` označava broj redova koji će biti izbrisani dok varijabla `deleteLines` označava postoji li potreba za brisanjem linija. U slučaju da postoje redovi koje je potrebno brisati, provjerava se koliko ih je i sukladno tome korisniku se pridjeljuju bodovi. U slučaju da je broj bodova veći nego što treba biti za trenutnu razinu, razina se povećava i proizvodi se pripadajući zvuk koji signalizira prelazak na iduću razinu. Provjerava se promatrajući umnožak trenutne razine i broja tisuću. Ukoliko je broj veći, prelazi se na iduću razinu, a ujedno se provjerava je li broj manji od deset tisuća kako bi se ograničio broj razina na deset (svaka razina predstavlja tisuću bodova).

3.7.2. Brisanje redova

Objašnjeno je kako prepoznati kada je red ispunjen. U nastavku će biti prikazan način brisanja ispunjenih redova.

```
void Game::clearLine() {
      mciSendString(L"seek line_clear to start", NULL, 0, NULL);
      mciSendString(L"play line_clear", NULL, 0, NULL);
      for (int r = ROWS-1; r > 0; r--){
            if (mainGrid[r][0].toBeDeleted){
                  for (int r2 = r; r2>0; r2--){
```

```
                        for (int c = 0; c < COLS; c++){
                              mainGrid[r2][c].isFilled=mainGrid[r2-
1][c].isFilled;
                              mainGrid[r2][c].isActive=mainGrid[r2-
1][c].isActive;
                              mainGrid[r2][c].toBeDeleted=mainGrid[r2-
1][c].toBeDeleted;
                              mainGrid[r2][c].red=mainGrid[r2-1][c].red;
                              mainGrid[r2][c].green=mainGrid[r2-1][c].green;
                              mainGrid[r2][c].blue=mainGrid[r2-1][c].blue;
                        }
                  }
                  r++;
            }
      }
      deleteLines = false;
}
```

Ispis 13 : Prikaz funkcije za brisanje redova

Funkcija prikazana u ispisu 13 prolazi po svim redovima počevši od vrha te provjerava koji su redovi označeni za brisanje. Red koji je označen za brisanje biva "pregažen" na način da se redovi iznad njega spuštaju za jedan red. Na taj način efektivno se briše red. Prilikom brisanja reda reproducira se zvuk koji signalizira da je došlo do brisanja reda.

3.7.3. Provjera uklapanja

U slučaju kad ne bi bilo ikakve provjere uklapanja, svaki komad samo bi padao u nedogled te igra ne bi imala smisla. Upravo iz tog razloga provjera uklapanja implementirana je na način koji je demonstriran u nastavku i prikazana u ispisu 14.

```
bool Game::moveCollision(int dir) {
      int x, y;
      const int* trans = activePiece.rotations();
```

```
	for (int i = 0; i < 8; i += 2) {
		x = activePiece.x + trans[i];
		y = activePiece.y + trans[i + 1];
		if (dir == 0)
			y += 1;
		else
			x += dir;
		if (x >= COLS || y >= ROWS || x < 0 || (mainGrid[y][x].isFilled
&& !mainGrid[y][x].isActive))
			return true;
	}
	return false;
}
```

Ispis 14 : Prikaz funkcije za provjeru uklapanja prilikom pokreta

Provjerava se može li se trenutni oblik "smjestiti" prilikom pokreta. Varijabla `trans` se koristi kako bi se uračunala rotacija trenutnog oblika. U suprotnom provjera bi bila neispravna. Funkcija u slučaju preklapanja vraća `true`.

```
bool Game::rotationCollision() {
	int x, y;
	const int* trans = activePieceCopy.rotations();
	for (int i = 0; i < 8; i += 2) {
		x = activePieceCopy.x + trans[i];
		y = activePieceCopy.y + trans[i + 1];

		if (x >= COLS || y >= ROWS || x < 0 || (mainGrid[y][x].isFilled
&& !mainGrid[y][x].isActive))
			return true;
	}
	return false;
}
```

Ispis 15 : Prikaz funkcije za provjeru uklapanja prilikom rotiranja

Prikazani kod u ispisu 15 je identičan kao i onaj prošli, osim što se ne uzima u obzir smjer kretanja oblika.

3.7.4. Rotiranje oblika

Kako bi igra bila zanimljiva omogućeno je rotiranje oblika da se razbije monotonost. Implementirano je na način prikazan u ispisu 16.

```
void Game::rotateShape(int dir) {
	activePieceCopy = Piece(random());
	activePieceCopy.x = activePiece.x;
	activePieceCopy.y = activePiece.y;
	activePieceCopy.rotation = activePiece.rotation;
	activePieceCopy.type = activePiece.type;
	activePieceCopy.rotatePiece(dir);

	if(canRotate(activePieceCopy)) {
		fixActivePiece();
		activePiece.rotatePiece(dir);
		updateActivePiece();
	}
}
```

Ispis 16 : Prikaz funkcije za rotiranje oblika

Radi se kopija trenutnog oblika na način da se generira nasumični oblik. Potom mu se dodjeljuju koordinate, rotacija i tip trenutnog oblika. Kopija se pritom rotira, nakon čega se provjerava je li moguća ta rotacija. U slučaju da je moguća, rotira se i originalni oblik.

```
bool Game::canRotate(Piece activeP) {
	if(rotationCollision()) {
		return false;
	}
	else
	{
		mciSendString(L"seek tup to start", NULL, 0, NULL);
		mciSendString(L"play tup", NULL, 0, NULL);
		return true;
	}
}
```

Ispis 17 : Prikaz funkcije za provjeru rotiranja oblika

Pomoćna funkcija `canRotate`, prikazana u ispisu 17, vraća `true` ili `false` ovisno o tome može li se oblik rotirati. Za provjeru rotacije oslanja se na prethodno definiranu funkciju `rotationCollision`. Prilikom uspješne rotacije reproducira se zvuk koji signalizira rotaciju.

3.7.5. Ažuriranje oblika

S obzirom da je igra vrlo dinamična, potrebno je s vremena na vrijeme ažurirati oblike kako bi postojao uvid u stvarno stanje igre.

```
void Game::updateActivePiece() {
	const int* trans = activePiece.rotations();
	for(int i = 0; i < 8; i += 2){
		Square &square = mainGrid[activePiece.y + trans[i +
1]][activePiece.x + trans[i]];
		square.isFilled = true;
		square.isActive = true;
		square.red = activePiece.redVal;
		square.green = activePiece.blueVal;
		square.blue = activePiece.greenVal;
	}
}
```

Ispis 18 : Prikaz funkcije za ažuriranje aktvnog oblika

```
void Game::updateNextPieceGrid() {
      const int* transNext = nextPiece.rotations();
      for (int i = 0; i < 8; i += 2) {
            Square& squareNext = nextPieceGrid[nextPiece.y + transNext[i +
1]][nextPiece.x + transNext[i]];
            squareNext.isFilled = true;
            squareNext.isActive = true;
            squareNext.red = nextPiece.redVal;
            squareNext.green = nextPiece.blueVal;
            squareNext.blue = nextPiece.greenVal;
      }
}
```

Ispis 19 : Prikaz funkcije za ažuriranje sljedećeg oblika

```
void Game::fixActivePiece() {

      const int* trans = activePiece.rotations();

      for(int i = 0; i < 8; i += 2){

            Square &square = mainGrid[activePiece.y + trans[i +
1]][activePiece.x + trans[i]];

            square.isFilled = false;

            square.isActive = false;

      }

}
```

Ispis 20 : Prikaz funkcije za ažuriranje padajućeg oblika

Funkcije `updateActivePiece` i `updateNextPieceGrid`, koje su prikazane u ispisima 18 i 19, razlikuju se samo u polju kojeg provjeravaju, a izuzev toga implementacija je izuzeto slična. Postavljaju se kao aktivni i ispunjeni oblici i dodjeljuje im se prikladna boja. Funkcija `fixActivePiece`, prikazana u ispisu 20, koristi se kako bi padajući oblik bio ažuran dok se igra izvodi.

3.7.6. Čišćenje polja

Ispis 21 prikazuje implementaciju čišćenja polja u igri.

```
void Game::clearMainGrid()
{
	for (int r=0; r<ROWS; r++) {
		for (int c=0; c<COLS; c++) {
			mainGrid[r][c].isFilled = false;
			mainGrid[r][c].isActive = false;
		}
	}
}
```

Ispis 21 : Prikaz funkcije za čišćenje polja igre

U ovom dijelu prolazi se po redovima i stupcima polja igre koji se postavljaju kao neaktivni i neispunjeni.

```
void Game::clearNextPieceGrid()
{
	for (int r = 0; r < 5; r++) {
		for (int c = 0; c < 5; c++) {
			nextPieceGrid[r][c].isFilled = false;
			nextPieceGrid[r][c].isActive = false;
		}
	}
}
```

Ispis 22 : Prikaz funkcije za čišćenje polja sljedećeg oblika

Ispis 22 identičan je kao u prethodnom primjeru, uz razliku da se radi o polju sljedećeg oblika.

3.7.7. Pomicanje oblika

Bez pomicanja oblika, igra ne bi mogla funkcionirati. Pomicanje je implementirano na način prikazan u ispisu 23.

```
void Game::move(int dir)
{
	if(moveCollision(dir))
		return;
	fixActivePiece();
	activePiece.x += dir;
	updateActivePiece();
}
```

Ispis 23 : Prikaz funkcije za pomicanje oblika

Funkcija `move` provjerava sudara li se trenutni oblik s rubovima polja. Ukoliko je tako, funkcija ignorira komandu za pokretom i ažurira ga. Varijabla `dir` označava pomak po x osi.

3.7.8. Ponovno pokretanje

Kada igra završi, potrebna je mogućnost njezinog ponovnog pokretanja. To je implementirano na način koji je prikazan u ispisu 24.

```
void Game::restart()
{
	clearMainGrid();
	clearNextPieceGrid();
	score = 0;
	level = 1;
	linesCleared = 0;
	shapesCount = 1;
	killed = false;
	paused = false;
	deleteLines = false;
	timer = 1000;

	activePiece = Piece(random());
	activePiece.x = COLS/2;
	activePiece.y = 0;
```

```
	updateActivePiece();


	nextPiece = Piece(random());
	nextPiece.x = COLS/2;
	nextPiece.y = 0;
	updateNextPieceGrid();

}
```

Ispis 24 : Prikaz funkcije za ponovno pokretanje igre

Funkcija `restart` čisti sva polja, generira idući aktivni oblik, kao i oblik u nizu nakon njega te postavlja sve parametre na početno stanje. Novi aktivni oblik se postavlja po sredini vrha polja igre.

3.7.9. Ažuriranje stanja igre

Kao što je i slučaj s ažuriranjem oblika, igru je potrebno ažurirati redovito kako bi korisnik imao uvid u njezino realno stanje.

```
void Game::update() {
	if (moveCollision(0)) {
		if (activePiece.y < 3) {
			killed = true;
			mciSendString(L"seek game_over to start", NULL, 0, NULL);
			mciSendString(L"play game_over", NULL, 0, NULL);
		}
		else {
			updateActiveAfterCollision();
			checkLine();
			if (deleteLines)
				clearLine();
				genNextPiece();


				clearNextPieceGrid();
```

```
                        updateNextPieceGrid();
                        updateActivePiece();
                }
        }
        else {
                fixActivePiece();
                activePiece.y++;
                updateActivePiece();
        }
}
```

Ispis 25 : Prikaz funkcije za ažuriranje stanja igre

Funkcija prikazana u ispisu 25 provjerava je li je došlo do sudara, odnosno uklapanja oblika. U slučaju da je tako, provjerava se nalazi li se aktivni oblik unutar polja igre. Ukoliko se aktivni oblik ne nalazi unutar polja igre, igra se završava, no u suprotnom igra se nastavlja.

4. ZAKLJUČAK

Kroz ovaj rad prikazan je i opisan proces izrade i implementacije jednostavne igre popularnog tipa koja je inspirirana poznatom igrom “Tetris”. U procesu razvoja ove računalne igre korištena su različita programska rješenja te tehnologije počevši od programskog jezika u kojem je rad napisan, do korištenog integriranog razvojnog okruženja te primijenjenog aplikacijskog programskog sučelja.

Programski jezik C++ omogućio je brzu realizaciju i implementaciju različitih funkcija koje su potrebne za nesmetan rad igre. Izbor ovog programskog jezika u ovom projektu pokazao se optimalnim radi svoje izuzetno kvalitetne integracije s aplikacijskim programskim sučeljem OpenGL, dobre podržanosti od strane Visual Studia te svoje brzine, efektivnosti i brojnih alata koje nudi. Važno je istaknuti kako se Visual Studio pokazao kao izuzetno praktičan alat za pisanje i izvođenje C++ koda.

OpenGL bio je od velike važnosti tijekom procesa razvoja ove računalne igre. Svojom brzinom omogućio je nesmetano razvijanje grafike za ovaj projekt te se pokazao izuzetno funkcionalnim kada je u pitanju razvoj ovakve računalne igre. Budući da je OpenGL poznato aplikacijsko programsko sučelje koje svojim korisnicima pruža ekstenzivnu dokumentaciju vezanu uz korištenje i primjenu postojećih funkcija, predstavljao je pravi odabir za ovaj projekt.

Rezultat ovog rada je jednostavna, ali dinamična i zabavna, 3D računalna igra koja unatoč svojoj jednostavnosti ima moć pružanja izazova i zadržavanja pažnje korisnika.

LITERATURA

1. Overvoorde, A. (2019.) Modern OpenGL Guide
2. Cvetković, D., Markovicć, D., Dulanović, D. (2006.) OpenGL praktikum, Računarski fakultet Beograd
3. Čupić, M., Mihajlović, Ž. (2021.) Interaktivna računalna grafika kroz primjere u OpenGL-u, Fakultet elektrotehnike i računarstva Zagreb
4. Cplusplus, https://cplusplus.com/doc/ (6.9.2022.)
5. Microsoft, https://visualstudio.microsoft.com/ (1.9.2022.)
6. OpenGL, https://www.opengl.org/Documentation/Documentation.html (5.9.2022.)
7. LearnOpenGL, https://learnopengl.com/ (6.9.2022.)