Univerzitet u Beogradu – Elektrotehnički Fakultet
Katedra za računarsku tehniku i informatiku

Poređenje alata za razvoj 3D video igara u tehnologijama *JavaFX*, *Bevy* i *Unity 3D*

diplomski rad osnovnih akademskih studija

Mentor

dr Igor Tartalja, v.prof.

Student

Uroš Filipović
2018/0663

Beograd, 2022.

Apstrakt

Veoma je važno da se pri pokretanju projekta razvoja softvera izabere odgovarajući alat, jer se time može predupredi veliki broj problema i grešaka u razvoju. Ovaj rad teži da razlike između tri izabrane tehnologije i odgovarajućih alata koji se koriste u razvoju 3D video igara stavi u perspektivu iz koje se mogu uporediti. Tehnologije se porede uz pomoć tri implementacije 3D video igre koja predstavlja zadatak laboratorijske vežbe na predmetu na kojem se izučava računarska grafika. Implementacije se upoređuju na osnovu različitih metrika, od kojih su neke objektivne, a neke zasnovane na ličnim iskustvima autora, dok su neke zasnovane na odgovorima anketiranih igrača.

Pored rezultata merenja ovaj rad prikazuje i strukturu, mogućnosti, duh i način korišćena svake od izabranih tehnologija. To uključuje i opis problema do kojih je došlo pri implementaciji i njihova rešenja. Jedan od ciljeva autora je da rad bude od pomoći mladim programerima video igara da lakše odluče u kojoj tehnologiji žele da ostvare svoj san.

Sadržaj

1. Uvod

Interaktivna računarska grafika (IRG) se bavi sintezom slike na računaru koja je kontrolisana interakcijom sa korisnikom i teži da obezbedi što prirodniji način korišćenja softvera ili softverskih paketa. Napredak u ovoj oblasti je možda najviše i doprineo popularizaciji i masovnom broju računarskih sistema današnjice. Primene IRG su mnogobrojne, ali neke od bitnijih su:

- programi za projektovanje (na primer, *IronCAD*);
- dizajn i ilustracije (na primer, *CorelDRAW*);
- video igre (na primer, *The Elder Scrolls V: Skyrim*).

Razvoj 3D video igara je jedan od najvećih tehnoloških izazova današnjice. Od najmanjih nezavisnih (eng. *indie game*) do najvećih *AAA* video igara svaka je u produkciji naišla na veliki broj teških zadataka. Da bi se svi ti zadaci lakše rešili inženjeri su napravili različite alate, a više kompanija razvilo je bogata okruženja za različite tehnologije koja se mogu uspešno koristiti za razvoj 3D video igara.

Veliki deo ljubitelja video igara je u nekom trenutku poželeo/la da napravi svoju "igru iz snova". Vrlo često njihovi projekti kreću unošenjem rečenice: "Kako napraviti 3D video igru" u pretraživač i pre par godina tako je počeo i autor ovog rada. Rezultat pretrage je bezbroj besplatnih ili plaćenih alata koje nude kompanije širom sveta. To dovodi do pitanja, kako da se izabere najbolji alat za određeni projekat?

Cilj ovog rada je da odgovori baš na to pitanje. Upoređivane su 3 tehnologije izabrane kao predstavnici svojih grupa. Prva se oslanja na okruženje *Unity 3D*, jedan od starijih, popularnijih alata za razvoji video-igara i jezik C#, druga koristi alat *Bevy*, koji je manji, tek u razvoju i otvorenog koda i treća je, kao referentna za studente koji slušaju predmet Računarska grafika, biblioteka *JavaFX* u okviru tehnologije za razvoj aplikacija na jeziku *Java*.

Svaka od tehnologija se koristi na drugačiji način. *Unity* ima svoje grafičko okruženje za formiranje i promenu scena, ali za pisanje koda se vezuje sa drugim okruženjima. Kao podrazumevano se koristi *Microsoft Visual Studio*, ali se mogu podesiti i druga, kao i *VSCode* koji je korišćen u ovom radu. Alat *Bevy* je napisan u programskom jeziku *Rust* i ne zahteva korišćenje specifičnog okruženja. Menadžer paketa *Cargo*, koji poziva *rustc* prevodilac, može se pokrenuti iz komandne linije, ali radi udobnosti pri radu može se koristiti i okruženje *VSCode* sa nadogradnjom *RustAnalyzer*, koje je korišćeno u ovom radu. Za implementaciju igre u tehnologiji *JavaFX* je korišćeno *IntelliJ IDEA* razvojno okruženje.

Da bi se ostvario cilj ovog rada, određene su metrike na osnovu kojih će se uporediti tri tehnologije iz perspektive razvoja jednostavne 3D video igre. Prvi rezultat rada su implementacije date igre u sve tri tehnologije. Na te tri igre biće primenjena kolekcija metrika važnih za korisnike igara i njihove programere. Neki od rezultata će biti objektivne metrike, dok će drugi biti subjektivni aspekti proizvoda koji će se meriti anketama. Više o metrikama u sledećem poglavlju.

Dalje u radu će se naći širi opis problema, funkcionalna specifikacija igre, generalni projekat igre, poglavlje sa tri potpoglavlja koja govore o svakoj implementaciji posebno, poređenje tih implementacija i zaključak.

2. Problem

U ovom poglavlju će biti opisan glavni problem rešavan u ovom radu i naznačen način na koji je dati problem rešavan. Takođe se može naći i kratak opis korišćenih metrika.

Veoma je važno da se pri pokretanju projekta izabere odgovarajući alat, jer se time može predupredi veliki broj problema i grešaka u razvoju. Svaki od alata ima svoje prednosti i mane, tako da treba težiti da se izabere alat koji najbolje odgovara potrebama projekta.

Slika 2.1. Logotipi tehnologija: (a) JavaFX; (b) BEVY; (c) Unity 3D

Ovaj rad se bavi poređenjem tri tehnologije (Slika 2.1.) i odgovarajućih alata koji su pogodni za razvoj 3D video igara. Prvenstveno je reč o video igrama koje studenti razvijaju u okviru laboratorijskih vežbi na predmetu na kojem se izučava računarska grafika. Kao što je u Uvodu naznačeno, porediće se alati *Unity*, *Bevy* i biblioteka *JavaFX*.

Prirodan put rešavanja ovog problema je napraviti prototip igre u svim tehnologijama i uporediti razvoj i proizvod, iz različitih perspektiva programera i korisnika (igrača).

Da bi se alati uporedili potrebno je razviti i skup adekvatnih metrika. Metrike su u ovom radu izabrane tako da mogu da na prikladan način reprezentuju iskustva razvojnog programera i korisnika. Metrike koje su korišćene se mogu podeliti u sledeće dve grupe:

1. bitne za programera:
    a. fleksibilnost i nivo apstrakcije razvojnog alata;
    b. veličina izvornog koda;
    c. veličina projekta;
    d. udobnost korišćenog alata;
    e. potrebno vreme za razvoj;
2. bitne za korisnika:
    a. veličina izvršnog programa;
    b. memorijske performanse izvršnog programa;
    c. broj podržanih platformi;
    d. utisak o kvalitetu proizvoda.

Smatraće se da postoje tri nivoa apstrakcije. Nizak nivo, u kome se direktno komunicira sa drajverom ili bibliotekom za grafiku. Visok nivo, gde se manipuliše entitetima i sakriva matematička podloga. I srednji nivo koji sadrži elemente i njihovu hijerarhiju (odnos roditelj-dete), kao na visokom nivou, ali je potrebno koristiti matrični račun za transformacije i slične koncepte.

Veličina izvornog koda se meri kroz broj datoteka, klasa, metoda i funkcija i linija koda.

Veličina projekta je veličina foldera projekta u MB. Ona uključuje izvorni kod, slike i sve potrebne metapodatke potrebne za rad alata. Ova metrika ne uključuje rezultat i propratne datoteke izvoza projekta.

Udobnost korišćenja alata je subjektivna ocena na osnovu vremena potrošenog za rad na implementacijama. Ocene su u opsegu od jedan do deset, gde je jedan najlošija ocena, a deset najbolja.

Potrebno vreme za razvoj je lična procena vremena potrošenog na svaku implementaciju. Merna jedinica ovog parametra su inženjer-sati.

Veličina izvršnog programa je ukupna veličina svih datoteka potrebnih da bi se igra pokrenula na bilo kom računaru. Ova metrika je izražena u MB.

Memorijske performanse izvršnog programa je maksimalna izmerena količina RAM memorije zauzete za vreme rada igre. Takođe izražena u MB.

Broj podržanih platformi je bitna metrika iz razloga što se povećanjem ovog broja povećava broj ljudi koji može da igra razvijenu igru.

Utisak o kvalitetu proizvoda se meri sprovođenjem ankete i teži da pokaže kako korisnici ocenjuju završni proizvod, opet u nekoliko kategorija:

1. lepota grafike;
2. odzivnost igre;
3. generalna ocena.

I ovde su sve ocene u opsegu od jedan do deset, gde je jedan najlošija ocena, a deset najbolja.

3. Funkcionalna specifikacija

U ovom poglavlju se opisuju funkcionalnosti jednostavne 3D video igre implementirane u 3 tehnologije radi poređenja tih tehnologija za razvoj odgovarajućih igara.

3.1. Inspiracija

Predmetna igra se zasniva na veoma poznatoj igri spretnosti. U ovoj igri okretanjem drvenog lavirinta, treba dovesti lopticu do ciljne rupe i pri tome izbeći sve ostale. Ovaj problem je idealan za pokazivanje specifičnosti okruženja iz razloga što sadrži dosta elemenata koje koriste 3D aplikacije i video igre današnjice.

Slika 3.1. Izgled fizičke igre

3.2. Funkcionalna specifikacija

U svakoj od implementacija potrebno je imati teren koji se rotira oko globalne X ose, pa onda oko lokalne Z ose pritiskom na tastere. Uglovi rotacije su ograničeni na pozitivnih i negativnih trideset stepeni. Nakon otpuštanja tastera teren se polako vraća u prvobitnu, neutralnu, poziciju.. Teren se nalazi u centru scene, odnosno centar terena se nalazi u koordinatnom početku.

Rotacijom terena počinje i pomeranje loptice. Ovaj efekat se postiže menjanjem njenog ubrzanja u odnosu na zadat ugao platforme. Svojim kretanjem loptica može interagovati sa ostalim objektima na terenu, od kojih su neki:

- pravougaone prepreke;
- neželjene rupe;
- završna, tj. ciljna rupa.

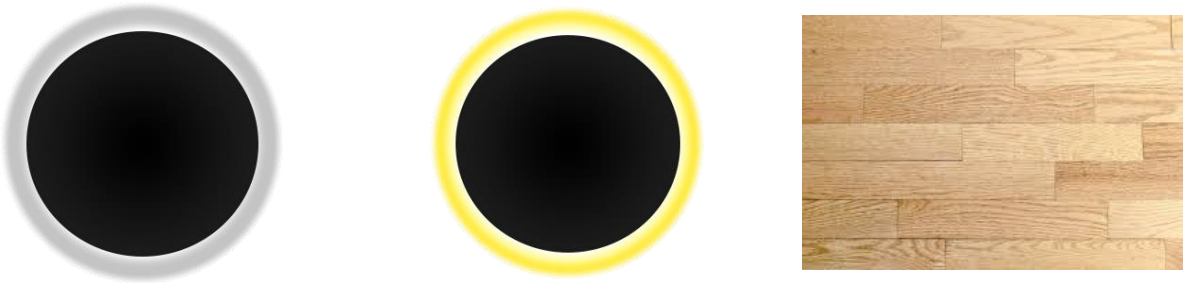
Loptica takođe može ispasti sa terena ukoliko određena sekcija nije ograđena.

Radi bolje preglednosti terena igraču je dozvoljeno pomeranje perspektivne kamere oko centra platforme korišćenjem miša, odnosno držanjem njegovog desnog tastera i

njegovim pomeranjem. Takođe je dozvoljeno približavanje i udaljavanje kamere uz pomoć točkića.

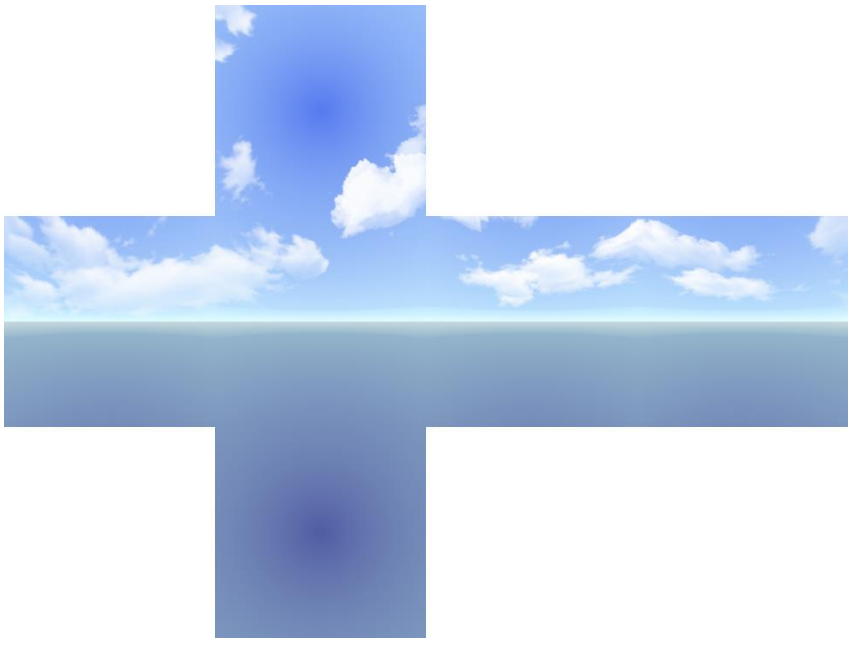
Pored perspektivne postoji i paralelna ortografska kamera koja u svakom trenutku prati lopticu pogledom odozgo. Promena kamere se vrši pritiskom na taster *C* na tastaturi.

Rupe, tabla i prepreke imaju teksturu. Nepoželjne rupe poseduju sivi okvir, dok ciljna ima zlatni. Za tablu i prepreke koristi se tekstura drveta. Nad tablom stoji tačkasto osvetljenje koje obasjava njene elemente, ali se ne pomera sa njom.



Slika 3.2. Teksture: (a) nepoželjna rupa; (b)ciljna rupa; (c) tabla

Kao pozadina scene koristi se tekstura na slici 5.3 koja je primenjena kao nebeska kutija (eng. *Skybox*). Nebeska kutija je kocka sa stranicama koje gledaju ka unutrašnjosti i imaju željenu teksturu, na koju ne utiče svetlost, već je stalno obasjana. Da bi se postigao efekat velikih dimenzija, kocka u svakom trenutku ima istu poziciju kao i aktivna kamera, ali se ne rotira sa njom.

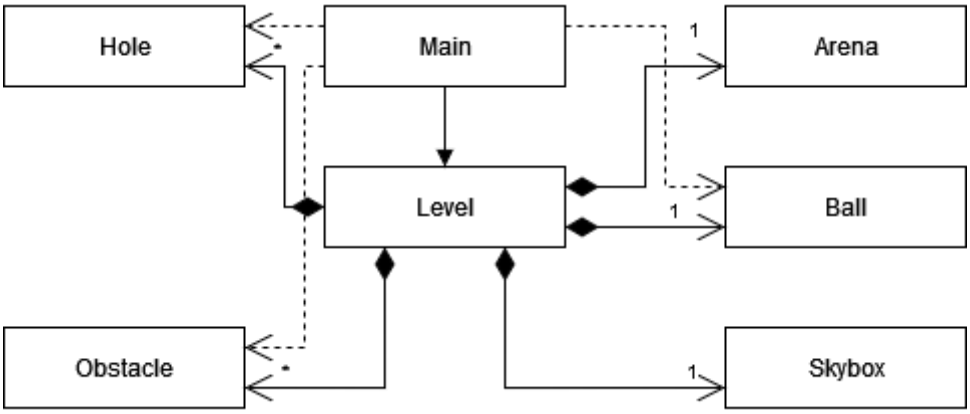


Slika 3.3. skybox.png

Nakon dolaska do ciljne rupe na centru prozora se pojavljuje tekst “YOU WIN!” koji u vremenskom periodu od pet sekundi polako nestaje, nakon čega se igra gasi. Tekst je bele boje.

4. Projekat igre

Ovo poglavlje opisuje konceptualni projekat koji se dalje može razlikovati od implementacije do implementacije zbog specifičnosti svakog alata. Dat je UML (*Unified Modeling Language*) diagram i kratak opis funkcionalnosti klasa.



Slika 4.1. UML diagram projekta

Klasa `Main` sadrži početnu funkciju aplikacije i njen glavni zadatak je da konstruiše objekat klase `Level` pomoću objekata klase `Ball` i nizova objekata klasa `Hole` i `Obstacle`. Pored toga `Main` mora da podesi sve potrebne postavke okvira.

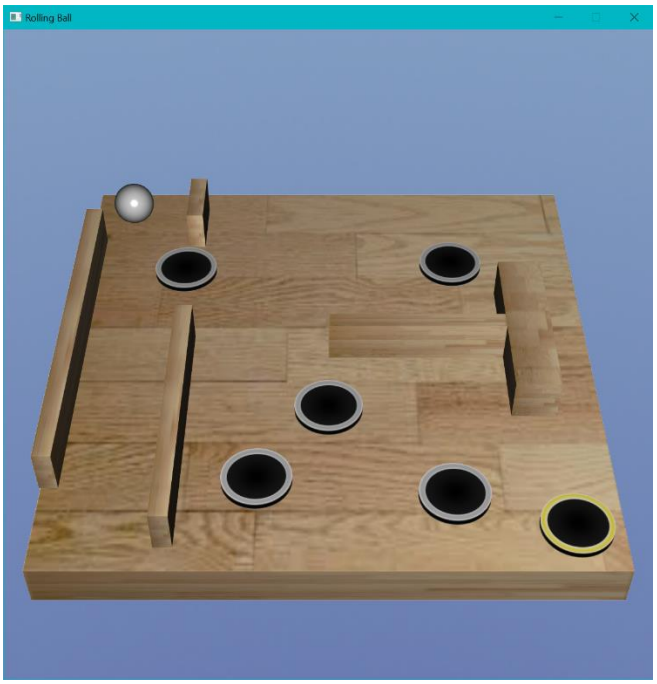
`Level` na osnovu dobijenih podataka priprema scenu za iscrtavanje tako što stvara arenu i kao decu postavlja liste objekata klasa `Obstacle`, `Hole` i `Ball`. Takođe, dodaje i `Skybox`. Funkcija `update` ove klase se poziva periodično pre svakog iscrtavanja i njen zadatak je da izvršava logiku igre tako što poziva `update` funkcije klasa svih sadržanih elemenata igre (`Hole`, `Obstacle`, `Ball`, `Arena`).

`Arena` čuva informacije o nagibu terena i njen `update` menja te uglove u zavisnoti od pritisnutih tastera na tastaturi. `Obstacle` proverava da li je došlo do kolizije sa loptom i adekvatno je razrešava. `Update` funkcija klase `Hole` vraća `boolean` vrednost koja označava da li je lopta upala u rupu. `Ball update` izvršava simulaciju fizike pomeranja loptice u odnosu na uglove arene. `Skybox` obezbeđuje iscrtavanje nebeske kocke opisane u funkcionalnoj specifikaciji.

5. Implementacija igre

U ovom poglavlju se nalazi opis svake implementacije zasebno. Poebna pažnja je posvećena duhu korišćenja svakog alata i zbog velike razlike u njihovim paradigmama ove implementacije se razlikuju od inicijalnog projekta opisanog u prethodnom poglavlju.

5.1. Implementacija u tehnologiji JavaFX



Slika 5.1.1. JavaFX implementacija

JavaFX implementacija sadrži sve klase i relacije prethodno opisanog projekta. Pored opisanih klasa ova implementacija uvodi još par novih: `StageController`, `KeyboardController`, `Timer` i `Utility`.

Klase `StageController` i `KeyobardController` primenjuju projektni uzorak *Unikat* (eng. *Singleton*) i koriste se kao pomoćne klase. `StageController` poseduje referencu na `Stage` objekat i metodom `getScene()` i `setScene(Scene scene)` dohvata ili postavlja trenutno aktivnu scenu. `KeyboardController` uz pomoć `HashMap` objekta beleži sve trenutno pritisnute tastere na tastaturi da bi se dobio sistem za rukovanje ulazom nalik onom u *Unity 3D*. Bez primene klase `KeyboardController`, ako se samo obrađuju događaji, držanjem tastera događaj se dešava periodom mnogo manjom od periode objekta `AnimationTimer`. Ugao se tada povećava stepenasto u velikim inkrementima. `KeyboardController` omogućava da se provera obavi elegantno i pri svakom iscrtavanju pozivanjem `KeyboardController.getInstance().isKeyActive( KeyCode.UP)`.

Klasa `Timer` prati događaje tajmera za animacije iz biblioteke *JavaFX* i poziva `update` funkcije iz svoje liste objekata koji implementiraju `Updatable` interfejs.

U ovoj implementaciji sve prepreke i rupe su minimalno izdignute iznad table da ne bi došlo do greške pri filtriranju piksela na osnovu vrednosti z-bafera.

JavaFX ne omogućava pisanje šejdera (eng*. shader*). Zbog toga u sceni ne postoje senke. Takođe, zbog nedostatka podrške za specijalizovane materijale i šejdere, nebeska kocka je implementirana na specifičan način. Glavna scena projekta je 2D scena koja sadrži tekst za kraj igre i dve 3D podscene. Prva koja se iscrtava je ona koja u sebi sadrži nebesku kutiju, paralelnu i perspektivnu kameru i ambijentalno svetlo, a druga sve elemente igre (arena, loptica, prepreke i rupe), tačkasto osvetljenje, perspektivnu i paralelnu kameru. Tako da je dobijena hijerarhija sledeća:

- Koren glavne 2D scene
  - Koren pozadinske 3D podscene
    - Nebeska kutija
    - Perspektivna kamera A
    - Paralelna kamera A
    - Ambientalno svetlo
  - Koren prednje 3D podscene
    - Arena
      - Tabla
      - Prepreke
      - Rupe
      - Lopta
    - Perspektivna kamera B
    - Paralelna kamera B
    - Tačkasto osvetljenje
  - Tekst za kraj igre

Nebeska kutija je implementirana putem `TriangleMesh` i `MeshView` klasa. Paralelna kamera A, perspektivna kamera A i kutija se u svakom trenutku nalaze u koordinatnom početku korena pozadinske scene.

Pri inicijalizaciji nivoa prvo se kreiraju transformacije koje se koriste za pomeranje perspektivne kamere B oko terena. Te transformacije uključuju (i primenjene su u sledećem redosledu): rotacija kamere oko Y ose, rotacija kamere oko X ose i njena translacija po Z osi. Ove transformacije se menjanju u metodima `handleMouseDrag`, `handleMouseReleased` i `handleScroll` klase `Level`.

Sve tri transformacije su primenjene na perspektivnoj kameri B, dok je na perspektivnoj kameri A primenjena samo transformacija rotacije. Time kamera A uvek ostaje u koordinatnom početku koliko god se kamera B pomerila, ali obe u svakom trenutku gledaju u istom pravcu. Ovim manipulacijama se izbegava efekat paralakse [1] i nebeska kutija izgleda kao da je beskonačno daleko [2].

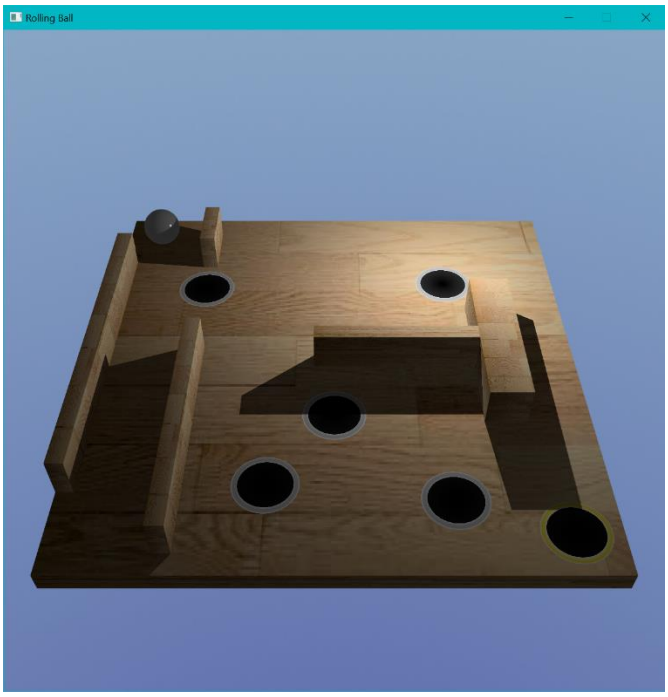
Problem kod ove implementacije je u tome što se pozadinska podscena uvek iscrtava u potpunosti iako postoje pikseli koji nisu vidljivi. Sa druge strane ovaj metod osigurava da

neće doći do problema sa z-baferom, da će se nebeska kocka uvek crtati iza ostalih objekata i da će uvek biti adekvatno osvetljena.

Tekst za kraj igre se dodaje u glavnu, 2D, scenu na kraju igre. Efekat nestajanja se postiže korišćenjem `FadeTransition` klase nakon čijeg završetka se poziva `Platform.exit()` i igra se gasi.

Izvoz projekta se vrši korišćenjem `jpackage` zadatka (eng. *task*) *Gradle* alata za izgradnju (eng. *build*) projekta, koja je generisana od strane *Intellij IDEA* okruženja. Rezultat je .msi datoteka za instalaciju aplikacije koja ne mora da se koristi i app direktorijum koji sadrži .exe same aplikacije, java izvršni paket i druge pomoćne datoteke.

5.2. Implementacija u tehnologiji BEVY



Slika 5.2.1. BEVY implementacija

Rust je programski jezik koji podržava više paradigmi (eng. *multi-paradigm*), objavljen u maju 2015. godine [3]. Na statistici *StackOverflow* pokazao se kao najomiljeniji jezik 2022 [4]. Za ovaj jezik se vezuje veliki broj projekata otvorenog koda i jedan od njih je *BEVY*. Sve igre u alatu *BEVY*, kao i sam alat, pisane su u programskom jeziku *Rust*. Ovaj jezik ima sledeće bitne karakteristike:

1. Programi pisani na jeziku se prevode u mašinski kod;
2. Strogo statički tipiziran;
3. Odvajanje struktura podataka od koda;
4. Automatsko oslobađanje memorije bez “sakupljača đubreta”;
5. Provera korektnosti svih pristupa podacima.

Kod se prevodi u mašinske instrukcije platforme na kojoj program treba se izvršava. Tokom ovog procesa prevodilac takođe vrši i optimizaciju rešenja, tako da su programi pisani u ovom jeziku veoma brzi.

Jezik je strogo statički tipiziran što znači da tip svake promenljive mora biti jednoznačno odrediv u vreme prevođenja. Ovim mehanizmom se izbegavaju mnogobrojne greške u pristupanju podacima, kave su česte u jeziku *Python*.

Razdvajanje podataka i njihove obrade (čitanja i upisa) je postignuto tako što se strukture podataka definišu u `struct` ili `enum` blokovima, osobine (eng. *traits*) se opisuju `trait` konstruktima koji načelno sadrže potpise funkcija i metoda, a kod kojim se implementiraju propisane funkcije osobina, struktura i nabrajanja se piše u `impl` blokovima.

Stavke 4 i 5 su osigurane od strane dela prevodioca koji se naziva *ispitivač pozajmica* (eng. *borrow checker*) i mehanizma vlasništva (eng. *ownership*). Srž mehanizma je sledeći skup pravila [5]:

1. Svaka vrednost (eng. *value*) u *Rust* programskom jeziku ima vlasnika (eng. *owner*);
2. U svakom trenutku postoji samo jedan vlasnik te vrednosti;
3. Kada vlasnik izađe iz opsega (eng. *scope*), vrednost se oslobađa.

Rust podržava reference. Uzimanje reference na vrednost se naziva pozajmljivanje (eng. *borrow*). Sve reference i promenljive su podrazumevano konstantne i ne mogu da menjaju vrednost na koje pokazuju ili čiji su vlasnik. Promena se dozvoljava tek ako se vlasnik ili referenca eksplicitno označe ključnom rečju `mut`. Ovaj mehanizam ima istu svrhu kao `const` ključna reč iz jezika C++, ali se upotrebljava na obrnut način radi veće sigurnosti. *Ispitivač pozajmica*, pored provere isteka životnog veka vrednosti, takođe osigurava da važe sledeća pravila [6]:

1. U svakom trenutku, za određenu vrednost može da postoji tačno jedna promenljiva referenca ili proizvoljan broj nepromenljivih;
2. Reference uvek moraju biti valjane (moraju da ukazuju na vrednost, da ne bi bile „viseće“).

Postoji podrška za generičke tipove, kao i pametne pokazivače kao što su `Box<T>` i `Rc<T>` [7] [8]. Obični pokazivači se retko koriste i dozvoljeni su samo u blokovima nebezbednog koda `unsafe { … }` [9].

Rust ne poseduje klase i interfejse, ali poseduje pomenute strukture i osobine koje se mogu koristiti na objektno orijentisan način [10]. U jeziku *Rust* ne postoji izvođenje struktura, a osobine mogu imati podrazumevanu implementaciju metoda. Osobine mogu zahtevati da struktura kojoj je data osobina pridružena poseduje još neku osobinu, kako bi koristile njene funkcionalnosti od kojih zavise [11].

Sve ove osobine čine *Rust* veoma sigurnim jezikom za jednonitno i posebno višenitno programiranje. Koriste ga na stotine kompanija širom sveta za projekte gde su brzina i sigurnost izvršnog koda primarni [12].

BEVY je, u trenutku pisanja ovog rada, u svojoj verziji 0.8 i pokazuje veliki potencijal. Zamišljen je kao alat za razvoj igara orjentisan ka podacima koji primenjuje arhitekturni uzorak entitet-komponenta-sistem (eng*. Entity Component System*, skr. ECS). ECS predstavlja paradigmu projektovanja jezgara za video igre koja podrazumeva da svaki objekat u igri bude jedan entitet (`Entity`), predstavljen jedinstvenim primerkom strukture u jeziku *Rust*, i da različite osobine tog entiteta predstavljaju njegove komponente (`Component`). Komponente se definišu kao strukture jezika *Rust* koje implementiraju osobinu `Component`. Iz razloga što broj komponenti brzo raste kroz razvoj igre i što su određene komponente logički vezane, dozvoljeno je njihovo grupisanje u snopove (eng. *bundle*) korišćenjem strukture `Bundle`. Mehanike se dodaju kroz sisteme (`System`), tj. funkcije koje se pozivaju pre početka igre ili pre svakog iscrtavanja [13]. Sistemi se dodaju pomoću `add_system` metoda (ili srodnih metoda) strukture `App` prosleđivanjem napomenute funkcije [14]. Ovaj pristup je značajno drugačiji od onog koji se primenjuje programiranjem na jeziku *Java* uz korišćenje biblioteke *JavaFX*, pa implementacija odstupa od projekta iz četvrtog poglavlja.

BEVY alat je organizovan u module. Svaki modul implementira `Plugin` interfejs i uz pomoć `build` funkcije učita sve potrebne podatke i doda sve potrebne sisteme. Tako je ova implementacija organizovana u nekoliko modula: `SkyboxPlugin`, `ArenaPlugin`, `BallPlugin`, `ObstaclePlugin`, `HolePlugin`, `BallAnimPlugin`, `SplashPlugin` i `LevelPlugin`. Svaki od modula se nalazi u posebnoj datoteci i sadrži logički povezane strukture sa osobinama. Na primer, modul `ArenaPlugin` se nalazi u datoteci `arena.rs` u kojoj se struktura (entitet) `Arena` nalazi zajedno sa osobinama (komponentama) `Rotator` i `ReturnAnimation`, kao i resursom `ArenaRes`.

Strukture sa sufiksom *-Res* poseduju osobinu resursa i pomažu u učitavanju potrebnih modela i tekstura. One se dodaju u `build` metodima svojih modula pomoću `init_resource` metoda aplikacije. Potrebno je da implementiraju `FromWorld` osobinu koja se može iskoristiti da se pozove `AssetServer` i učitaju podaci. Svi delovi projekta su organizovani na ovaj ili sličan način.

Nebeska kutija je implementirana kao kocka sa stranicama čije su normale okrenute ka unutrašnjosti i koja koristi posebno definisan materijal. Materijali se koristi za senčenje modela na kojima su primenjeni, tako što se kroz njih navode potrebni parametri i programi (eng. *shader*). *BEVY* ostvaruje komunikaciju sa grafičkom kartom računara pomoću *wgpu* grafičke biblioteke. Biblioteka *wgpu* pomenute parametre prosleđuje do grafičke kartice kroz vezne grupe (`BindGroup`). Da bi to bilo moguće potrebno je da *BEVY*, odnosno *wgpu* poznaje raspored tih parametara. Raspored se opisuje kroz `BindGroupLayout` strukturu.

Od verzije 0.8 definicija novog materijala u *BEVY* alatu je znatno uprošćena i vrši se kroz strukturu koja implementira osobine `TypeUuid`, `Clone`, `AsBindGroup` i `Material`. Struktura nosi ime `SkyboxMaterial`. Osobina `TypeUuid` se koristi za navođenje jedinstvenog identifikatora strukture [15] potrebnog da bi se struktura koristila kao sredstvo (eng*. asset*). Osobina `Clone` omogućava kloniranje strukture. Osobina `AsBindGroup` omogućava da se struktura pretvori u veznu grupu (`BindGroup`) i njen raspored (`BindGroupLayout`) [16]. Implementiranjem funkcija osobine `Material` se naznače datoteke koje sadrže programe za senčenje.

Pošto je potrebno da nebeska kutija ima teksturu, ona se mora navesti u samoj strukturi `SkyboxMaterial` na sledeći način:

```
pub struct SkyboxMaterial {
    #[texture(0)]
    #[sampler(1)]
    cubemap: Option<Handle<Image>>,
}
```

Struktura `SkyboxMaterial` poseduje polje `cubemap` tipa `Option<Handle<Image>>`. Primerak strukture `Handle<Image>` se dobija kao rezultat učitavanja slike iz datoteke, a nabrajanje `Option<T>` omogućava da tekstura ne mora da se zada pri kreiranju materijala, tj. da je opciona. `Option<T>` je još jedna od specifičnosti programskog jezika *Rust* i predstavlja pojam najbliži *null* referenci ili pokazivaču koji ovaj jezik poseduje. To je zapravo generički `enum` sa dve varijante (eng. *variant*): `None` i `Some(T)`. Tako da, ako je potrebno naznačiti teksturu, prosledila bi se vrednost `Some(handle)`, ako ne, onda samo `None` [17].

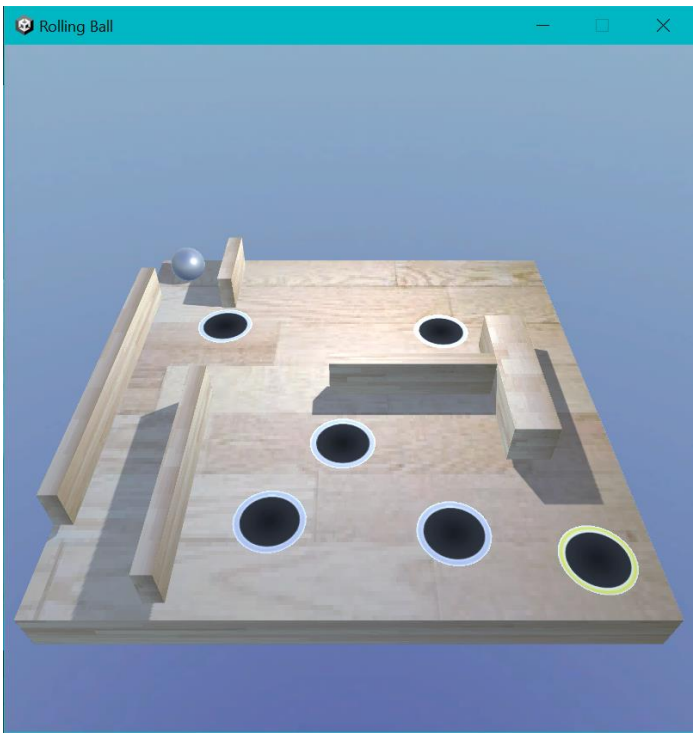
`#[texture(0)]` i `#[sampler(1)]` su makroi koji se koriste da bi *BEVY* znao na kojem indeksu u veznoj grupi treba da postavi teksturu, a na kojem njen odabirač da bi generisao `BindGroupLayout` za dati materijal.

U ovom slučaju *verteks* program obezbeđuje ignorisanje translatornog kretanja kamere uklanjanjem delova inverzne matrice pogleda ili prizora[1] (eng. *View matrix*) $A_{4\times4} = (a_{ij})$, tj. upisivanjem nule na mesta elemenata koji predstavljaju translaciju (a_{41}, a_{42} i a_{43}) [18]. Takođe obezbeđuje da se kutija crta iza svih objekata upisivanjem jako male vrednosti z-koordinate izlaznih `vec4` koordinata. Te koordinate predstavljaju poziciju tačke u koordinatnom sistemu NDC (*Normalized Device Coordinates*), koji se koristi za odsecanje tačaka koje se neće videti na ekranu [19] [20]. *Fragment* program ignoriše osvetljenje i vrši odabiranje zadate teksture koja je navedena u materijalu [2]. Tekstura se nalazi u veznoj grupi sa indeksom 1 na nultom mestu, a odabirač na prvom. Inverzna matrica pogleda i ostali parametri koje *BEVY* sam prosleđuje se nalaze u veznoj grupi sa indeksom 0. Oba programa za senčenje su pisana u *wgsl* jeziku koji je nešto noviji i više orijentisan ka veb platformi. Međutim, *BEVY* podržava i *glsl*.

BEVY omogućava izvoz projekta na veliki broj platformi pokretanjem `cargo build --release` komande sa dodatnom `--target` opcijom. Podržane platforme uključuju *Windows*, *Linux*, *macOS* i veb.

[1] BEVY definiše matricu pogleda kao transformacionu matricu kamere koja je postavlja na odgovarajuće globalne koordinate uz odgovarajuću rotaciju. Zbog ove konvencije se koristi inverzna matrica pogleda.

5.3. Implementacija u tehnologiji Unity



Slika 5.3.1. Unity 3D implementacija

Unity 3D pruža vizuelni editor scena koji dosta olakšava razvoj. Svaki objekat u igri poseduje niz skripti koje se mogu videti i podesiti u *Inspector* prozoru. Parametri su obično javna polja klase, ali mogu se implementirati i razni specijalizovani interfejsi. Ovim putem se uglavnom povezuju objekti koji programski interaguju i nisu dinamički dodati.

Dobra strana ovog alata je u tome što je veliki broj često potrebnih mehanika već implementiran. Ovo uključuje nebesku kutiju, fiziku i animacije.

Zbog prisutnosti 3D fizičkog sistema, da bi se lopta kretala potrebno je samo primeniti rotaciju nad terenom nakon čega, zbog gravitacije, loptica kreće da se pomera. Detekcija izlaska van terena se vrši putem detekcije kolizije sa nevidljivim kutijama povezanih sa arenom.

Rupe su realizovane na sličan način. One poseduju cilindar, podešen kao okidač, koji nakon detekcije kolizije poziva `OnTriggerEnter(Collider collider)` metod skripte zakačene za lopticu gde se proverava vrsta rupe, tj. da li je rupa ciljna ili nije. Na osnovu te informacije poziva se adekvatna funkcija za manipulaciju loptice i pobuđuje adekvatan `UnityEvent`, da bi se informisali ostali objekti i adekvatno odreagovali.

6. Poređenje razvijenih igara

Ovo poglavlje vrši upoređivanje tri alata odnosno odgovarjućih implementacija kroz diskusije i metrike. Prvo će bit reči o nivou apstrakcije, fleksibilnosti i udobnosti alata, nakon čega će se razmatrati neke objektivne metrike i rezultati sprovedene ankete.

Na osnovu ove podele *JavaFX* bi predstavljala srednji nivo, dok bi *BEVY* i *Unity* predstavljali visoki nivo apstracije iz nekoliko razloga. *JavaFX* koristi klase izvedene iz klase `Transform` kao glavni način za pomeranje, skaliranje, rotiranje i slične manipulacije nad oblicima i zahteva poznavanje matematičke podloge da bi se adekvatno primenile [21], dok druga dva alata pokušavaju da uproste proces različitim funkcijama kao što su `Transform::look_at(&mut self, target: Vec3, up: Vec3)` i `Transform.LookAt(Transform target)`, za *BEVY* i *Unity* respektivno.

Sa druge strane, kada bi se fleksibilnost podelila na ista tri nivoa, visok, srednji i nizak, *Unity* i *BEVY* alati bi se nalazili na visokom, a *JavaFX* niskom nivou. Ovakva ocena se može potkrepiti činjenicom da prva dva alata nude mogućnost dodavanja novih materijala kroz pisanje specijalizovanih šejder programa, dok *JavaFX* nema tu mogućnost.

	JavaFX	BEVY	Unity3D
Broj fajlova	13	10	6
Broj klasa/struktura	11	32	6
Broj metoda/funkcija	34	41	19
Broj linija koda	812	1094	201
Veličina projekta	3.05 MB	14.4 MB	337 MB
Veličina izvršnog programa	84.3 MB	21.8 MB	71.7 MB
Zauzeće memorije	143.8 MB	114.0 MB	76.9 MB
Broj podržanih platformi	8	4 + 2	22

Tabela 6.1. Tehničke karakteristike

U Tabeli 6.1. se nalaze izmerene vrednosti metrika navedenih u drugom poglavlju. Iz tabele se može videti da *Unity* ima ubedljivo najmanju količinu koda, verovatno zahvaljujući velikom broju implementiranih primitiva i vizuelnog editora scena, ali najveću kolekciju izvršnih fajlova. *BEVY* vodi iako ima najkomplikovaniji kod ima najmanji izvršni fajl i koristi konstantnih 114 MB ram memorije. *JavaFX* ima najmanju veličinu projekta, ali je u sredini po svim ostalim metrikama.

Broj podržanih platformi je možda i najvažnija metrika. *JavaFX* podržava ugrađene uređaje (eng. *embedded devices*), desktop platforme (*Windows*, *Linux* i *macOS*), veb, Android, iOS i TV [22]. *BEVY* podržava sve navedene desktop operativne sisteme i veb (kroz *WebAssembly*) platforme, a dodatno podržava *Android* i *iOS* (u izradi) [23]. *Unity 3D* poseduje spektar od dvadeset dve podržane platforme od malih ugrađenih *Linux* uređaja, preko konzola i veba, sve do moćnih desktop mašina [24].

	JavaFX	BEVY	Unity3D
Potrebno vreme za razvoj	31 inž. sati	41 inž. sati	22 inž. sati
Udobnost korišćenja alata	6	7	8

Tabela 6.2. Subjektivne individualne procene

Tabela 6.2. sadrži subjektivne individualne procene izdvojenog vremena za svaku implementaciju ne uključujući vreme potrošeno na učenje korišćenja samih alata. U njoj se takođe nalaze i subjektivne ocene autora rada o udobnosti korišćenja alata u intervalu od 1 do 10.

	JavaFX	BEVY	Unity3D
Prosečna ocena lepote grafike	5,95	8,81	8,33
Prosečna ocena odzivnosti	6,48	8,33	8,38
Prosečna generalna ocena	6,14	8,52	8,38

Tabela 6.3. Rezultati ankete

Tabela 6.3. predstavlja prosečne ocene implementacija u različitim kategorijama. Anketa je sprovedena putem *Google Upitnik* platforme. Bilo je 21 različitih ispitanika. Na malom uzorku korisnika se jasno vide prednosti alata *BEVY* i *Unity3D* u odnosu na *JavaFX*.

7. Zaključak

Svaka tehnologija, odnosno odgovarajući alat, se pokazala dobrom iz nekog aspekta, ali *Unity* se pokazao kao najbolji izbor. Postojanje vizuelnog okruženja i intuitivnog API (eng*. Application Programming Interface*) dosta ubrzava razvoj.

BEVY, iako još uvek mlad projekat, se pokazao veoma stabilnim. Modularna struktura i otvoren kod omogućavaju duboko razumevanje rada i, po potrebi, lako debagovanje i izmenu. Iako je potrebno više vremena za razvoj, veliki deo potencijalnih grešaka pri radu programa se primeti u vreme prevođenja i obavezno popravi zbog prirode programskog jezika *Rust*. To rezultuje u sigurnijem i čistijem kodu.

JavaFX je odličan alat za razvoj desktop aplikacija koje sadrže dobar korisnički interfejs sa nekim 3D aspektima. Za razvoj video igara nedostaje veliki broj odlika i ne postoje adekvatni načini kojim se mogu implementirati (npr. nebeska kutija).

U budućnosti se planira ispitivanje većeg broja alata i prikupljanje odgovarajućih rezultata. Dodatno bi se moglo ispitati kako se ovi alati pokazuju na većim projektima na kojima radi više ljudi u dužem vremenskom periodu. U tom slučaju bi se mogao pratiti i broj bagova koji su nastali i vreme koje je bilo potrebno da se oni uklone.

8. Literatura

[1] „Pojam paralakse,“ [Na mreži]. Available: https://www.opsteobrazovanje.in.rs/astronomija/paralaksa/.

[2] J. d. Vries, „Tutorijal o implementaciji nebeske kutije u OpenGL tehnologiji,“ [Na mreži]. Available: https://learnopengl.com/Advanced-OpenGL/Cubemaps.

[3] D. Madunuwan, „Github hostovan sajt za učenje Rust programskog jezika,“ 2022. [Na mreži]. Available: https://learning-rust.github.io/docs/a1.why_rust.html.

[4] StackOverflow, „StackOverflow anketa o programskim jezicima,“ 2022. [Na mreži]. Available: https://insights.stackoverflow.com/survey/2021#technology-most-loved-dreaded-and-wanted.

[5] S. Klabnik i C. Nichols, „What is Ownership? The Rust Programming Language,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html.

[6] S. Klabnik i C. Nichols, „References and Borrowing, The Rust Programming Language,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html.

[7] S. Klabnik i C. Nichols, „Using Box<T> to Point to Data on the Heap,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch15-01-box.html.

[8] S. Klabnik i C. Nichols, „Rc<T>, the Reference Counted Smart Pointer,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch15-04-rc.html.

[9] S. Klabnik i C. Nichols, „Unsafe Rust,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html.

[10] S. Klabnik i C. Nichols, „Object-Oriented Programming Features of Rust,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch17-00-oop.html.

[11] S. Klabnik i C. Nichols, „Traits: Defining Shared Behavior,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch10-02-traits.html.

[12] S. Klabnik i C. Nichols, „Introduction,“ [Na mreži]. Available: https://doc.rust-lang.org/book/ch00-00-introduction.html.

[13] „Module bevy::ecs::system,“ [Na mreži]. Available: https://docs.rs/bevy/latest/bevy/ecs/system/index.html.

[14] „Struct bevy::app::App,“ Bevy documentation, [Na mreži]. Available: https://docs.rs/bevy/latest/bevy/app/struct.App.html#method.add_system.

[15] „Crate type_uuid,“ [Na mreži]. Available: https://docs.rs/type-uuid/latest/type_uuid/.

[16] „Trait bevy::render::render_resource::AsBindGroup,“ [Na mreži]. Available: https://docs.rs/bevy/latest/bevy/render/render_resource/trait.AsBindGroup.html.

[17] „Module std::option,“ [Na mreži]. Available: https://doc.rust-lang.org/std/option/.

[18] v. l. Tartalja, „Transformacije u 3D i projekcije,“ 27 3 2017. [Na mreži]. Available: https://rti.etf.bg.ac.rs/rti/ri5rg/materijali/predavanja/09%20Transformacije%20u%203D%20i%20Projekcije.pdf.

[19] „Vertex Processing,“ [Na mreži]. Available: https://gpuweb.github.io/gpuweb/#vertex-processing.

[20] „Primitive Clipping,“ [Na mreži]. Available: https://gpuweb.github.io/gpuweb/#primitive-clipping.

[21] „Class Transform,“ [Na mreži]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/transform/Transform.html.

[22] Gluon, „Products,“ [Na mreži]. Available: https://gluonhq.com/products/.

[23] „Bevy engine,“ [Na mreži]. Available: https://bevyengine.org/.

[24] „What platforms are supported by Unity?,“ [Na mreži]. Available: https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity-.