# Kuroi Yuki: razvoj atmosferskog psihološkog horora u Unity okruženju

**Šeparović, Marko**

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike

**MARKO ŠEPAROVIĆ**

# KUROI YUKI: RAZVOJ ATMOSFERSKOG PSIHOLOŠKOG HORORA U UNITY OKRUŽENJU

Diplomski rad

Pula, rujan, 2022.

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike

**MARKO ŠEPAROVIĆ**

# KUROI YUKI: RAZVOJ ATMOSFERSKOG PSIHOLOŠKOG HORORA U UNITY OKRUŽENJU

Diplomski rad

**JMBAG: 0303077993, redoviti student**

**Studijski smjer: Informatika**

**Predmet: Dizajn i programiranje računalnih igara**

**Znanstveno područje: Društvene znanosti**

**Znanstveno polje: Informacijske i komunikacijske znanosti**

**Znanstvena grana: Informacijski sustavi i informatologija**

**Mentor: izv. prof. dr. sc. Tihomir Orehovački**

Pula, rujan, 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Marko Šeparović, kandidat za magistra INFORMATIKE ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Marko Šeparović

U Puli, 06.08., 2022. godine

**IZJAVA O KORIŠTENJU AUTORSKOG DJELA**

Ja, Marko Šeparović dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Kuroi Yuki: razvoj atmosferskog psihološkog horora u Unity okruženju koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 06.08.2022. (datum)

Potpis

Marko Šeparović

# SADRŽAJ

# 1 Uvod

Ovaj diplomski rad će opisati proces izrade atmosferske horor igrice „Kuroi Yuki“ u Unity razvojnom okruženju. Unity je među-platformsko razvojno okruženje primarno korišteno za razvoj video igrica. Ovo okruženje je razvijeno od strane Unity Technologies te je prvi put najavljeno i objavljeno u lipnju 2005. godine. Verzija Unity-ja korištena za izradu igrice za ovaj projekt je 2021.3.5f1.

Ovaj rad opisuje i dokumentira ključne elemente izrade jedne trodimenzionalne atmosferske horor igrice iz prvog lica. Počevši sa skriptama koje omogućavaju pokretanje lika. Pa sve do animacija i upravljanja glazbom u igrici. Unity primarno koristi programski jezik C# te će taj jezik biti korišten pri razvoju Kuroi Yuki igrice.

Kuroi Yuki daje veliki naglasak na psihološki aspekt horor igrica te se ne oslanja na jeftine prepade(eng. Cheap Jumpscares) da bi izazvala strah kod igrača. Glavni pokretač osjećaja straha je osvrtanje na realne opasnosti u svijetu koje su predočene kroz duboku scenografiju i brojne bilješke koje su rasprostranjene po tmurnom prostoru. Apstraktnost je isto tako veoma važan pokretač priče. Igrač ima mogućnost korištenja alata „mind's eye“, koji omogućava glavnom liku da „vidi“ dio alternativne dimenzije koja djeluje kao kontrast uobičajenom krajoliku te služi kao glavna mehanika u igrici. Dopušta igraču da se pomiče na područja na koja nije mogao dospjeti bez tog alata. Ova mehanika potiče igrača da razmišlja i pronađe različite načine da napreduje kroz igricu.

Neke od inspiracija za priču i teme igrice Kuroi Yuki su igrice kao: „Anatomy“ i „Lethargy Hill“ od poznate developerke Kitty Horrorshow, „Who's Lila“ od Garage_Heathen i „Iron Lung“ od developera David Szymanski.

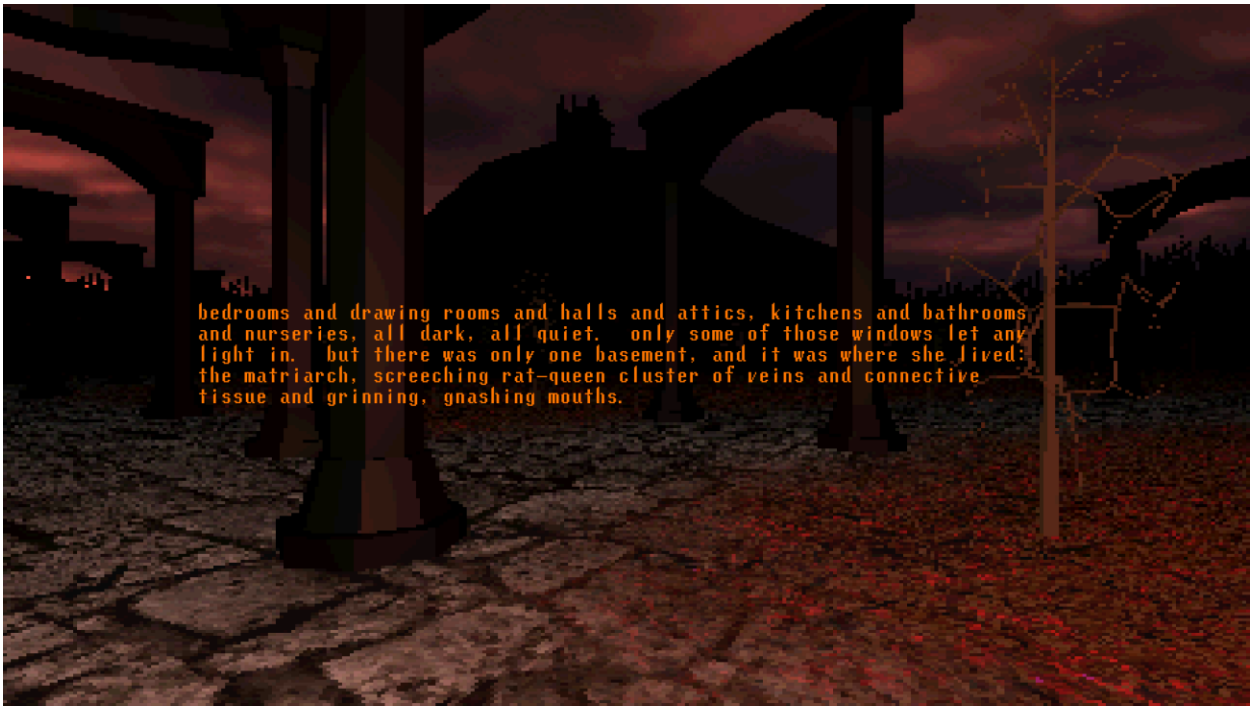
U ovom radu opisani su svi elementi prisutni u razvoju 3D video igre iz perspektive prvog lica. Objašnjen je sustav pokretanja igrača, kamere i ostalih elemenata kojima igrač upravlja tipkovnicom i mišem. Bit će objašnjena logika, proces izrade i kod iza pokretanja glavne neprijateljske sile u igrici, sustava čestica, game managera, audio managera, raycasting sustava te svih ostalih elemenata koji su korišteni u razvoju ove igrice.

# 2 Inspiracije

Kuroi Yuki je igrica inspirirana mnogim indie horror igricama. Glavne od ovih inspiracija su: „Anatomy“ i „Lethargy Hill“ od poznate developerke Kitty Horrorshow, „Who's Lila“ od Garage_Heathen i „Iron Lung“ od developera David Szymanski. U idućem dijelu će biti opisane tematike ovih igrica te direktni način na koji su ove igrice inspirirale igru vezanu za ovaj rad.

## 2.1 Lethargy Hill

Lethargy Hill je nedvojbeno najveća inspiracija igrici koju koja je obrađena u ovome radu, Kuroi Yuki. Ova igrica je razvijena od strane Kitty Horrorshow, developerke koja je poznata po svojim ezoteričnim horror igricama. Izdana 2020. godine u Haunted Cities Vol. 4 kolekciji igrica, Lethargy Hill stoji kao najstrašnija igrica u toj kolekciji. Slika 1 prikazuje snimku ekrana tijekom igranja Lethargy Hill.




*Slika 1: Snimka ekrana iz igre Lethargy Hill*

Priča od Lethargy Hill govori o ženi koja živi u kući na brdu. Ta žena se osjeća usamljeno te povodom toga „otkida dijelove sebe“ i od tih dijelova pravi „dronove“ koji joj pomažu da napravi porodicu. Pravi muža koji se sastoji od zalijepljenih grančica, sestru koja je

konstruirana od duhova s obližnjeg groblja, kćerku čije je tijelo formirano od tisuću paukovih leševa i sina kojeg pravi od svoje krvi. Ove deskripcije su veoma detaljno napisane kroz autorov vrhunski stil pisanja.

Ova igrica je u teoriji veoma jednostavna, ne dopušta pomicanje kamere na Y osi, frame rate je zaključan na 15 frame-ova po sekundi. Rezolucija je namjerno smanjena te su teksture pikselizirane s ciljem davanja specifičnog jezivog utjecaja na igrača. Jedini način interakcije s okolinom je pokretanje po mapi koja se vrti oko brda u sredini dok se tekst koji opisuje priču, pomiče po ekranu.

Ipak ova igrica je najveća inspiracija Kuroi Yuki. Način pripovijedanja koji je sveprisutan u Lethargy Hill te sama tematika igrice je utjecala direktno na iste elemente u Kuroi Yuki. Pri kraju igrice Lethargy Hill, desi se događaj(eng. event) koji je najbolje opisan kao inteligentni prepad(eng. intelligent jumpscare). Sasvim neočekivano, nebo pocrveni te se na istom nebu stvore anđeoske figure koje izrazito nadvisuju igrača te stvaraju izrazit osjećaj nelagode i straha. Ti osjećaji su ono što je cilj velike većine horor igrica. Različite igrice tog žanra postignu taj učinak na različite načine, u ovom radu će biti opisani na koji način je Kuroi Yuki postigao taj rezultat.

Postoji dosta načina na koji se Lethargy Hill i Kuroi Yuki razlikuju. Kuroi Yuki je igrica koja ima dosta drugačiji grafički pristup i stil nego Lethargy Hill. Koristi puno detaljnije teksture i nema zaključan frame-rate. Ovo su naravno tehničke razlike i ne dokazuju višu vrijednost igrice, veći utjecaj na kvalitetu ima stiliziranost igrice, neovisno o grafičkim sposobnostima i pristupima. Igrica obrađena u ovom radu uz grafičke razlike ima i veću mehaničku sposobnost nego Lethargy Hill.

Moguće je gledati u svim smjerovima te igrači imaju mogućnost korištenja alata (Mind's Eye), interakcije s okruženjem (Note System), primanja štete, skakanja i dosta ostalih mogućnosti. Lethargy Hill je ipak „minimalistična“ igrica s glavnim fokusom na priču. Kuroi Yuki je isto tako velikim dijelom usredotočena na priču ali zato ima ogromnu sposobnost igrača da utječe na napredak kroz igricu, dok u igrici od Kitty Horrorshow igrač nema izbora nego da napreduje.

## 2.2 Anatomy

Anatomy je najpoznatija igrica od developerke Kitty Horrorshow te jedna od, ako ne i najstrašnija igrica koju je ona razvila. Igrica je izdana 2016. godine. Igrica sadrži elemente minimalne gameplay mehanike ali maksimalne atmosfere i pripovijedanja veoma jezive priče kao i većina ostalih igrica od ove developerke.

Anatomy, u veoma doslovnom smislu govori o poveznici između fizičkog izgleda kuće i ljudske anatomije. Naravno ova povezanost koja se odnosi na fizičku ljudsku anatomiju a i psihološku analizu ljudske prirode je veoma vješto opisana kroz kasete koje igrač pronalazi u kući i pušta na kasetofonu.

Što učiniti ako kuća u kojoj živimo ne želi da budemo dio nje? Što ako zapuštamo kuću, i ona postane „gladna“ ili „ogorčena“. Anatomy opisuje ove pojmove kroz vrhunsku stilističku naraciju i atmosferu. U igrici je sveprisutan jezivi šum koji podsjeća na stare domaće snimke i drži igrača na konstantnom oprezu. Na slici 2, prikazana je snimka ekrana tijekom igranja igrice Anatomy.

*Slika 2: Snimka ekrana iz igre Anatomy*

Kitty Horrorshow je ekspert u pravljenju prikladne atmosfere u njenim horor igricama. Naravno, nije razočarala u Anatomy.

Gameplay ove igrice sadržava hodanje po jezivoj, živućoj kući i skupljanje kaseta koje igrač vraća do kuhinje i pušta na kasetofonu. Na ekranu je uvijek prisutan filter vizualnog „šuma“ koji insinuira da to što igrač radi je na neki način kasetni snimak. Igrač može razbijati tanjure i otvarati vrata, ima i opciju „sprint“ koja mu omogućava da brže trči kroz kuću.

Anatomy inspirira Kuroi Yuki na više načina. Glavni način bi bio vrhunska naracija očekivana od Kitty Horrorshow i izvanredni dizajn zvuka koji poboljšava jezivu atmosferu igrice za bezbroj puta. Isto tako, Anatomy je ogromna inspiracija temi igrice obrađene u ovom radu jer se obje igrice masivno oslanjaju na psihološki aspekt horor igrica te pokušavaju utjecati na emocije igrača kroz duboke psihološke teme i stvoriti osjećaj kontemplacije i dugoročnog razmišljanja o temama igrice.

Ljudska psiha je dosta opširan pojam i omogućava velik broj interpretacija od strane igrača. To čini psihološke atmosferske horor igrice veoma podložnim za razmišljanje i vrlo pogodnim za induciranje straha kroz načine različite od klasičnih prepada.

Kuroi Yuki čini uravnoteženost između ovakvih igrica, koje se primarno baziraju na naraciji i pripovijedanju te služe priču kao pokretač i igrica koje su mehanički zahtjevne i imaju puno interaktivnih elemenata.

Može se povući zaključak da je Anatomy najviše slična igrici Kuroi Yuki od navedenih inspiracija ali ipak postoje razlike. Neke od tih razlika su grafička sposobnost. Anatomy namjerno koristi „low poly“ teksture da bi postigla jeziv osjećaj što uveliko pomaže ambijentu usamljene kuće. Kuroi Yuki ipak koristi više detaljne teksture i modele s ciljem postizanja jedinstvenog ambijenta razrušenog grada i okolne pustoši. Anatomy nema pozadinsku glazbu nego ima konstantni jezivi šum koji veoma pozitivno djeluje na ciljani osjećaj indie horor igrice. U Kuroi Yuki korištena je pozadinska glazba ali kao sekundarni element, kao nešto na što igrač ne razmišlja ali podsvjesno utječe na njegovu nelagodu unutar tog razrušenog svijeta.

## 2.3 Iron Lung

Iron Lung je atmosferska horor igrica koju je razvio David Szymanski. Izašla je na steam-u 10.3.2022. godine i jedna je od poznatijih indie horor igrica na tržištu. Iron Lung je bila analizirana od strane više youtube-ra koji prave esejske youtube videe te je kritički hvaljena od istih. Slika 3 prikazuje snimku ekrana tijekom igranja Iron Lung.


*Slika 3: Snimka ekrana iz igre Iron Lung*

U Iron Lung se radi o univerzumu u kojem se desio događaj pod imenom „a quiet rapture“. Taj događaj je napravio da svaka nastanjiva planeta i svaka zvijezda misteriozno nestanu. Jedino su preživjeli ljudi koji su u tom trenutku bili u svemirskim postajama i svemirskim brodovima.

Pronađen je mjesec na kojem postoji jezero od krvi, odlučeno je da taj jezivi fenomen mora biti istražen. Jedan zatvorenik na smrtnoj kazni je zavaren u podmornicu i poslan u to jezero da istražuje. Rečeno mu je ako preživi da će biti pušten na „slobodu“.

Igrica Iron Lung koristi „low-poly“ teksture i tmurnu paletu boja koji dočaravaju tu ciljanu jezivu atmosferu. Dizajn zvukova u ovoj igrici je izvanredan i igrač kroz cijelu igru osjeća stravu. Igrač nikad ne izađe iz te male podmornice i unatoč toga, gameplay je veoma interesantan i privlačan.

Gameplay ove igrice je veoma unikatan. Igrač može koristiti 2 gumba za upravljanje podmornice na x i y osama, 2 gumba za rotiranje podmornice i jedan gumb za slikanje s prednjom kamerom podmornice. Jedini način da igrač zna što se dešava izvan podmornice je da koristi tu kameru/fotoaparat. Jedini način da igrač zna gdje se trenutno nalazi je da usporedi trenutnu vrijednost x i y koordinata s istom vrijednošću na papirnoj karti na zidu podmornice.

Na prednjem zidu u unutrašnjosti podmornice se vide 2 mjerača. Lijevi mjeri trenutnu dubinu podmornice dok desni mjeri trenutni udio kisika u podmornici. Na stražnjem zidu igrač vidi ekran koji prikazuje slike sa prednje kamere, i aparat za gašenje požara. Ako u podmornici izbije požar radi visokog tlaka ili kvara u elektronici, igrač mora iskoristiti protupožarni aparat da bi ugasio tu vatru.

Glavni način na koji Iron Lung inspirira Kuroi Yuki je korištenje kreativnih mehanika. Uz to, ogromna je inspiracija kod dizajna zvukova u igrici s ciljem poboljšavanja horor atmosfere. Ambijentalni zvukovi u Iron Lung su sasvim izvanredni te stvaraju dubinu u relativno mehanički ograničenoj igrici. Kuroi Yuki uzima te principe korištenja zvukova za produbljivanje priče i ambijenta te ih koristi u svom dizajnu zvukova. U igrici Iron Lung, neprijatelji su okruženje i strah od nepoznatog. Taj aspekt horor igrica je veoma prisutan u Kuroi Yuki.

Razlike između Iron Lung i Kuroi Yuki su prisutne. Iron Lung je igrica koja namjerno i veoma efektivno izaziva osjećaj straha u zatvorenom okruženju. Igraču nije dopušteno da napusti podmornicu što povećava osjećaj klaustrofobije te čini igrača da se osjeća bespomoćnim. Kuroi Yuki se služi s otvorenom mapom da bi stvorila ambijent razrušenog i stravičnog krajolika. Igrač se može slobodno kretati po mapi te je i potaknut da tako djeluje. Dosta elemenata iz misteriozne priče se mogu puno bolje shvatiti obraćanjem pažnje na okolinu.

# 3 Unity

Za razvoj igrice Kuroi Yuki korišteno je razvojno okruženje „Unity Engine“. Unity je međuplatformsko razvojno okruženje primarno korišteno za razvoj video igrica. Unity se isto tako koristi za razvoj aplikacija, filmova, animacija, arhitektonskih simulacija, itd.

Može se koristiti za većinu stvari za koje je potrebno trodimenzionalno ili dvodimenzionalno modeliranje ili animiranje. Kompanija koja je razvila Unity razvojno okruženje se zove Unity Technologies. U trenutku razvoja igrice obrađene u ovom radu, najnovija verzija Unity-ja je bila verzija 2021.3.5f1 te je korištena za razvoj Kuroi Yuki.

Unity se može koristiti za razvoj 3D i 2D igrica. Pri razvoju se koriste mnoge važne značajke kao 3D renderiranje, detekcija kolizije, mapiranje refleksija, paralaks efekt, naknadna obrada, sustav čestica te mnogi drugi alati. Za razvoj 2D igrica koristimo „sprites“ što su jednostavni 2D objekti koji imaju teksture na njima. Za animiranje 2D scene koristimo sekvencijsko izmjenjivanje ovih sličica. Sprite-ovi nemaju širinu na Z osi.

## 3.1 Povijest Unity-ja

Unity je pokrenut 2005. godine s ciljem „demokratizacije“ razvoja igrica tako što ga čini pristupačnijim developerima. Inicijalno je pokrenut za Mac OS X, a kasnije je dodana podrška za Microsoft Windows i Web Pretraživače (Haas, 2014).

Unity 2.0 je pokrenut 2007. godine s otprilike 50 novih značajki. Ova distribucija je uključila optimizirani terenski pokretač(eng. Terrain Engine) za 3D okruženje, „real-time“ dinamične sjene, usmjerena svjetla(eng. Directional Lights), video playback i puno drugih alata.

Unity 3.0 je distribuiran 2010. godine te je proširio grafičke sposobnosti za osobne računare i igraće konzole. Uz dodavanje podrške za android, Unity 3 je integrirao dodatne alate kao automatsko UV mapiranje, audio filtere, built-in editor drveća, itd.

Unity 4.0 (2012. g.) distribucija je dodala podršku za Adobe Flash, novi animacijski alat nazvan „Mecanim“ i pristup Linux pretpregledu.

Unity 5.0 (2015. g.) distribucija Unity Engine-a je dugo-očekivani korak prema cilju pravljenja razvoja video igara univerzalno pristupačnim. Poboljšani su audio i osvjetljenje te kroz WebGL, unity developeri mogu praviti svoje igrice za kompatibilne web-preglednike bez da igrači moraju skidati plug-ine.

Unity (2017 - sadašnjost). U prosincu 2016. godine, Unity Technologies su najavili da će promijeniti sistem numeriranja verzija Unity-ja od sekvencijalnog na sistem baziran na godini izdanja. Uvedeni su alati kao: timeline, CineMachine, Scriptable Render Pipeline, High Definition Render Pipeline, podršku za virtualnu i poboljšanu realnost, volumetrijske oblake, alate za strojno učenje i mnogo drugih alata (Haas, 2014).

U idućoj sekciji, nekoliko najnovijih alata će biti opisano i njihova uporaba u ovom projektu će biti objašnjena.

## 3.2 Render Pipelines

Unity podržava mogućnost biranja između različitih cjevovoda renderiranja (eng. Render pipelines). To su grafički prikazivači koji izvode niz operacija koje preuzimaju sadržaj scene i prikazuju ih na ekranu (Unity, 2021). Najkorišteniji RP prikazivači u unity projektima su HDRP, URP i built in render pipeline. HDRP (eng. High Definition Render Pipeline) i URP (eng. Universal Render Pipeline) su oboje SRP (eng. Scriptable Render Pipelines) dok ugrađeni (eng. Built In) render pipeline nije skriptabilan, što znači da se ne može upravljati kroz C# kod.

HDRP (eng. High Definition Render Pipeline) je „Visoko kvalitetni“ (eng. High Fidelity) grafički prikazivač (eng. Render Pipeline) koji je razvijeni od strane unity-ja da bi omogućio ciljanje razvoja na moderne platforme koje su kompatibilne s računanjem kompleksnih kalkulacija shadera.

URP (eng. Universal Pender Pipeline) je grafički prikazivač usmjeren prema umjetnicima te dopušta lagano i brzo kreiranje optimizirane grafike koja funkcionira na visokom rasponu platformi. Od mobilnih igrica/aplikacija do računalnih. Za razvoj igrice Kuroi Yuki korišten je Universal Render Pipeline. Radi savršene mješavine visoke grafičke sposobnosti i „light weight“ prirode, zaključeno je da je ovo optimalan izbor.

## 3.3 Volumetrijski Oblaci

Volumetrijski oblaci (eng. Volumetric Clouds) je „volume“ komponenta koja kontrolira postavke iscrtavanja (eng. Rendering) volumetrijskih oblaka koji mogu iscrtavati sjene i primati sjenu i volumetrično svjetlo.

Ova komponenta je jedino dostupna u High Definition Render Pipeline-u. Gledajući da je igrica Kuroi Yuki razvijena u Universal Render Pipeline-u, korišten je alternativni način pravljenja volumetričnih oblaka. Ovaj način obuhvaća modeliranje u aplikaciji za pravljenje modela, Blender, i korištenje URP shadera grafova. Ovaj proces će biti opisan u jednoj od idućih sekcija u radu.

## 3.4 Graf Shader-a

Graf shader-a (eng. Shader Graph) je alat u Unity razvojnom okruženju koji je distribuiran u Unity verziji 2018. Ovaj alat omogućava developerima da prave shadere vizualno koristeći sistem baziran na čvorovima (eng. Nodes) umjesto da utječu na shadere kroz kod. Shaderi su programi koji se pokreću na grafičkoj kartici korisnika.

To su male skripte koje sadržavaju matematička izračunavanja i algoritme za računanje boje od svakog iscrtanog piksela na ekranu. Ove kalkulacije se izvršavaju na osnovi svjetlosnog inputa i konfiguracije materijala. Umjesto pisanja koda, koristeći shader graph mogu se stvarati i spajati čvorovi unutar graf okruženja. Ovaj alat korisnicima daje povratne informacije u pravom vremenu koje prikazuju njihove promjene.

U ovom radu, shaderi i shader grafovi su korišteni u više prilika te će te instance korištenja biti opisane i objašnjene. U Kuroi Yuki shaderi su korišteni za iscrtavanje volumetrijskih oblaka, krvi, itd. Bit će opisan i način da se postigne „pikselizirani“ izgled igrice što može u dosta slučajeva dati pozitivan učinak na stil igrice te napraviti horor igricu strašnijom.

# 4 Kuroi Yuki

Kuroi Yuki je igrica čiji su razvoj i elementi obrađeni u ovom radu. To je igrica žanra „atmosferski psihološki horor“ što znači da se ne oslanja na tradicionalne prepade u horor igricama već istražuje ono što ljudska bića zaista pronalaze strašnim i iskorištava to, uz atmosferske elemente kao zvuk, teksture, ambijent i osvjetljenje da izazove što veći osjećaj straha kod igrača. Slika 4 prikazuje snimku ekrana pri početku igranja Kuroi Yuki.

*Slika 4: Snimka ekrana iz igre Kuroi Yuki*

„Kuroi Yuki“ je japanski naziv za „Crni Snijeg“. Taj se pojam odnosi na radioaktivni pepeo/čestice koje padaju s neba nakon nuklearne eksplozije. Tema igrice Kuroi Yuki se duboko veže s ljudskom okrutnošću, referencira događaje u ljudskoj povijesti koji su uobličile pojam okrutnosti i događaje koji su općepriznati kao tragedije. Naravno na igraču je da interpretira poantu ove priče, ali činjenica je da je općeprisutan osjećaj tuge i gubitka.

U ovoj priči igrač, u doslovnom smislu igra dušu djeteta koja se kreće kroz opustošeni i razoreni svijet. Igrač na početku nema ništa u svom posjedu te se prvobitno kreće s ciljem shvaćanja situacije oko sebe. Igrač nailazi na papir koji mu dublje opisuje smisao svijeta u kojem se nalazi. Ubrzo nakon toga igrač nalazi predmet i uzima ga. Taj predmet je

igračka koju je ta duša imala dok je bila živa. Igračku igrač koristi kao „leću“ kroz koju može gledati u neki bolji svijet. Taj bolji svijet omogućava igraču da prođe kroz prepreke koje prije nije mogao savladati. Duša se nastavlja kretati po ovom svijetu, oko sebe vidi tragedije, jedna od njih je veliki oblak nastao nakon nuklearne eksplozije. Ubrzo nakon toga krene padati „crni snijeg“ koji ošteti igrača te mu smanjuje životne poene. Duša prolazi kroz tragedije i nakon nekog vremena se popne na brdo na kojem je veliko izgoreno drvo. To drvo se zove Smoldering Gazer. Drvo ima veliko oko u sredini te igrač mora izbjeći kontakt s tim pogledom. Kroz brojne bilješke u ovom svijetu insinuirano je da igrač mora ići prema velikom nuklearnom oblaku.

Kad se duša približi oblaku nebo postane crveno, sve postane distorzirano te igrač ne može više koristiti igračku s kojom je napredovao kroz svijet. Glazba postane jeziva i pulsirajuća te se na nebu vide velike crne figure. Igrač dolazi do oblaka na kojem nalazi nešto sasvim zastrašujuće...

U igrici Kuroi Yuki glavni način propagiranja priče je sustav bilješki. Igrač pronalazi bilješke po svijetu koje opisuju svijet i likove koji se u njemu nalaze. Način na koji igrač otvara bilješke je pritiskanjem gumba „E“ na tipkovnici dok gleda u papir.

Mind's Eye je ime „igračke“ koju igrač koristi kako bi napredovao kroz svijet i rješavao prepreke koje se tu nalaze. Ako je neki put blokiran s otpadom, igrač može pogledati u Leću pritiskanjem gumba „F“ i tako gledanjem u „bolji svijet“ koji se zove „Innocent Realm“ može pronaći put koji postoji samo u tom svijetu.

Neprijatelj u ovoj igrici je okruženje. To uključuje crni snijeg koji periodično pada i oštećuje igrača. Svaka pahuljica koja pogodi igrača će mu smanjiti život za određenu količinu. Glavni „Boss“ u igrici je izgorjelo drvo pod imenom Smoldering Gazer. Kad igrač priđe ovom drvetu, ono će mu govoriti „mite“ što na japanskom znači „gledaj“ te će na pulsirajući način „tjerati“ igrača da pogleda u veliko oko u sredini stabla tako što silom pomjera kameru prema oku. Ako igrač susretne pogled s okom, gubi život u igrici.

Igrač ima sposobnosti pokretanja i skakanja. Isto tako može brzo da trči pritiskanjem gumba „shift“ dok drži neki gumb za pokretanje.

# 5 Razvoj Igrice

U nastavku rada će biti opisani svi elementi prisutni pri razvoju igrice Kuroi Yuki u Unity okruženju. Bit će obrađene cjeline: igrač, okruženje, menadžer igre, post processing, sustav čestica, graf shader-a, sustav sačuvanja napretka kroz igricu i tako dalje. Opisivanje razvoja počinje s cjelinom „igrač“ u kojoj će biti opisane sve skripte i elementi povezani s igračem i glavnim likom.

## 5.1 Igrač

Kuroi Yuki je igrica iz perspektive prvog lica, to znači da igrač igra iz perspektive lika. Ovakve igrice dopuštaju da igrač vidi točno ono što lik vidi. Radi ovoga, ne postoji način da se manipulira pogled tako da se dobije „bolji“ pregled svijeta, jedina opcija je gledanje oko sebe. To znači da ako igrač želi pogledati što se nalazi iza njega (lika) onda mora cijeli pogled rotirati.

Ovo utječe na igrivost igrice na više načina. Na primjer ako se u igrici iz perspektive prvog lica neprijatelj nalazi iza nekog kutka, igrač ne može provjeriti da li je ovo istina bez da stavi sebe u opasnost i pokaže se neprijatelju. U većini igrica ovog tipa, ruke/oružje/objekti od lika su skoro uvijek vidljive u pogledu. Glavni lik je karakteriziran najviše kroz način na koji drugi likovi utječu na njega i kroz okruženje.

Igrice iz perspektive trećeg lica, za usporedbu, se odnose na igricu gdje igrač gleda svog lika kao gledalac umjesto da kontrolira lika kroz njegov pogled direktno. Tipično u ovakvim igricama, lik se vidi preko ramena (eng. Over The Shoulder) ili iz perspektive iza leđa (eng. Behind The Back).

Kamera prati lika i dopušta da se vidi više stvari oko njega nego što igrica iz perspektive iz prvog liza dopušta. Ako se ovo usporedi s prijašnjim primjerom, u ovakvim igricama se kamera može rotirati tako da igrač vidi što se nalazi iza kutka sve dok se glavni lik nalazi na sigurnoj poziciji. Gledajući da se lik vidi sve vrijeme, ovakve igrice lakše prikazuju osobnost lika i dopuštaju da se sazna više o načinu kako taj lik hoda, utječe na okruženje i reagira na događaje (Stegner, 2020).

Sada će biti prikazan i objašnjen način na koji je programiran sustav pokretanja kamere i lika u igrici opisanoj u ovom radu. Uz to će biti opisani općeniti elementi i pojmovi korištenja C# skripti u Unity okruženju.

### 5.1.1 Upravljač Mišem

U isječku koda broj 1, prikazan je početni dio skripte MouseController.cs.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MouseController : MonoBehaviour{
    public float mouseSensitivity = 100f;

    public Transform playerBody;

    float xRotation = 0f;

    // Start is called before the first frame update
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
    }
```

*Isječak Koda 1 Skripta MouseController.cs 1. dio*

Na početku skripte za pokretanje kamere se nalazi javna (eng. public) varijabla tipa float koja određuje osjetljivost rotiranja kamere. Kada je varijabla javna (eng. public) to znači da se može uređivati unutar inspektora/editora u Unity-u što omogućava lakšu optimizaciju specifičnih funkcija kao osjetljivost rotacije jer se ove javne varijable mogu mijenjati i kad je igrica u pokrenutom stanju (eng. runtime). Isto tako postoji javna varijabla „transform“ tipa koja omogućava da joj bude dodijeljen objekt u sceni (eng. Game Object) i da se dobiju od njega podaci vezani za transformaciju tog game objekta, ti podaci su pozicija, rotacija i veličina objekta. Nalazi se i ne-javna varijabla za rotaciju na x osi tipa float kojoj je po defaultu dodijeljeno 0f. MonoBehaviour je bazična klasa od koje se derivira svaka Unity skripta. Kad se koristi C#, mora se isključivo derivirati iz MonoBehaviour klase. MonoBehaviour.Start je funkcija koja se poziva samo jednom u životnom vijeku (eng. Lifetime) od skripte te se poziva na frame-u kad je skripta omogućena i prije Update() metode. U ovoj Start funkciji se zaključava kursor na centar

ekrana i pravi se nevidljivim koristeći Cursor.lockstate = CursorLockMode.Locked;. U nastavku je Update() funkcija opisana ispod.

```
    // Update is called once per frame
    void Update()
    {
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity *
Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity *
Time.deltaTime;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);

        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
        playerBody.Rotate(Vector3.up * mouseX);

    }
}
```

*Isječak Koda 2 Skripta MouseController.cs 2. dio*

Monobehaviour.Update metoda se poziva svaki frame, tako da što god je u njoj definirano će biti izvršeno svaki frame u runtime-u. Koristeći input klasu i njenu GetAxis() funkciju, dobivaju se koordinate od x i y osi, te se množe s prije spomenutoj osjetljivosti i još se množe s Time.deltaTime. Time.deltaTime je interval u sekundama od zadnjeg frame-a do trenutnog. Množenje s ovim dopušta da se održi konstantnost između frame-ova što znači da će korisnici s visokim frame-rate-om i s niskim imati isti učinak. U ovom slučaju znači da će oba korisnika imati istu osjetljivost rotiranja kamere ako namjeste na istu vrijednost neovisno o njihovim računalnim performansama.

Mathf.Clamp omogućava da se ograniči rotacija na specifične vrijednosti što se u ovom slučaju odnose na kutove rotiranja kamere. X rotaciju ograničavamo između -90f i 90f što znači da kad igrač pogleda u nebo, ne može nastaviti rotirati kameru prema nazad. Isto tako kad pogleda u pod ne može nastaviti rotirati kameru u tom smjeru.

Na kraju, zapravo se aplicira ova izračunatu rotacija na svakom frame-u tako što se transformira lokalna rotacija GameObject-a na koji je ova skripta postavljena. U ovom slučaju to je „MainCamera“ GameObject koji predstavlja glavnu kameru koja je dijete od „Spirit“ GameObjecta koji predstavlja glavnog lika. Koristi se Quaternion.Euler funkcija za

primjenjivanje rotacije, ova funkcija vraća rotaciju od z stupnjeva po z osi, x stupnjeva po x osi te y stupnjeva po y osi, u tom redoslijedu. Na kraju se još rotira playerBody, a to je GameObject (Spirit) na koji je postavljena glavna kamera.

### 5.1.2 Pokretanje Igrača

U nastavku će biti opisan način pokretanja glavnog lika po okruženju. U ostatku rada će biti izostavljene „using“ linije koda te inicijalizacija MonoBehaviour klase osim u slučaju da je taj dio koda u skripti drugačiji i promijenjen od zadanoga koji generira unity.

Počinje se s inicijalizacijom varijabli i Start() funkcijom u PlayerMovement.cs skripti prikazanoj ispod.

```
public CharacterController controller;

float speed;
public float walkingSpeed = 8f;
public float sprintSpeed = 12f;
public float gravity = -9.81f;
public float jumpHeight = 3f;
public bool isSprinting = false;

public Transform groundCheck;
public float groundDistance = 0.4f;
public LayerMask groundMask;
private bool IsMoving;
private AudioSource audioSource;

public Vector3 move;
Vector3 velocity;

bool isGrounded;

void Start()
{
    audioSource = gameObject.GetComponent<AudioSource>();
}
```

*Isječak Koda 3 Skripta PlayerMovement.cs 1. dio*

CharacterController je klasa u unity-u koja omogućava da se izvršavaju kretanja ograničena kolizijama bez potrebe za dodavanjem rigidbody komponente u objekt. Rigidbody je način da u unity-u se kontroliraju objekti kroz simulacije fizike, a kolizije su

instance sudara nekog gameobjekta s colliderom (komponenta koja određuje fizičke granice nekog objekta) s drugim gameobjectom s colliderom.

Na upravljač lika (eng. CharacterController) ne utječu sile fizičke simulacije u igrici te će pokretati lika samo kad se pozove „Move“ funkcija. Onda će izvršiti pokrete ali će biti ograničeno kolizijama.

U ovoj skripti se nalazi još nekoliko javnih varijabli koje se koriste u funkcijama. AudioSource varijabla referencira zvučnu datoteku stavljenu u isti GameObject u koji je stavljena PlayerMovement.cs skripta. Ta zvučna datoteka je zvuk koraka koji se aktivira kasnije u ovoj skripti. Nastavlja se s Update() funkcijom prikazanoj u nastavku.

```
    void Update()
    {
        //ground check
        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance,
groundMask);

        if(isGrounded && velocity.y < 0)
        {
            velocity.y = -2f;
        }

        //movement
        float x = Input.GetAxis("Horizontal");
        float z = Input.GetAxis("Vertical");
```

*Isječak Koda 4 Skripta PlayerMovement.cs 2. dio*

U Update() funkciji, svaki frame se provjerava da li je glavni lik u kontaktu sa zemljom, to se radi s CheckSphere() funkcijom iz Physics klase. Ako lik dotiče teren/zemlju i y os varijable velocity tipa Vector3 je manja od 0, što znači da lik pada, namještamo da je velocity.y jednaka -2f. To znači da kad je lik blizu zemlje, usporava mu se pad, to je trik koji čini koliziju sa zemljom tijekom padanja učinkovitijom.

Nakon toga dobivaju se x i z vrijednosti osi kroz GetAxis() funkciju kojoj se kao argument daje string reprezentacija osi koja se traži. Ove ključne riječi su navedene u Edit/ProjectSettings/InputManager... Vertical i Horizontal ključne riječi vraćaju vrijednosti osi između -1 i 1 kad igrač pritisne tipke W i S za vertikalnu os i A i D za horizontalnu os. U nastavku je prikazan nastavak Update() funkcije.

```
        //sprint
        if(isGrounded)
        {
            if (Input.GetKey(KeyCode.LeftShift) && z == 1)
            {
                speed = sprintSpeed;
                isSprinting=true;
            }
            else
            {
                speed = walkingSpeed;
                isSprinting=false;
            }
        }
        else
        {
            speed = walkingSpeed;
                                    }
```

*Isječak Koda 5 Skripta PlayerMovement.cs 3. dio*

Dok je igrač u kontaktu sa zemljom, ako pritišće lijevi shift gumb i z je jednako 1 (što znači da drži gumb W ili da mu je lijevi joystick maksimalno pomjeren prema naprijed) onda je varijabla speed (hrv. brzina) jednaka varijabli sprintSpeed (hrv. brzina sprintanja). Uz to varijabla isSprinting je jednaka true, to će kasnije biti korišteno za provjeru da li je lik u sprintu.

Ako jedan od ovih uvjeta nije istinit onda se stavlja da varijabla speed je jednaka varijabli walking speed(hrv. brzina hodanja). Uz to, varijabla isSprinting je jednaka false. Ako igrač nije „grounded“, to jeste ako nije na podu, onda brzina je jednaka brzini hodanja. U isječku koda ispod, prikazuju se ClampMagnitude() i Move() funkcije.

```
        //movement
        move = Vector3.ClampMagnitude(transform.right * x + transform.forward *
z, 1.0f);
        controller.Move(move * speed * Time.deltaTime);
```

*Isječak Koda 6 Skripta PlayerMovement.cs 4. dio*

U varijablu move se stavljaju vektori pokreta pomnoženi s input vrijednostima, uz to na kraju se pozove ClampMagnitude() što ograničava ove pokrete na maksimalno 1.0f.

Ovaj move vektor se stavlja u controller.Move() funkciju koja pokreće lika i uz to se množi s brzinom i Time.deltaTime da zaustavi varijaciju pokreta u odnosu na performanse. U nastavku se regulira skakanje glavnog lika.

```
        if(Input.GetButtonDown("Jump") && isGrounded)
        {
            velocity.y = Mathf.Sqrt(jumpHeight * -2f * gravity);
        }
        //gravity
        velocity.y += gravity * Time.deltaTime;
        controller.Move(velocity * Time.deltaTime);

        if (Input.GetAxis("Vertical") != 0 || Input.GetAxis("Horizontal") != 0)
IsMoving = true; // better use != 0 here for both directions
        else IsMoving = false;
```

*Isječak Koda 7 Skripta PlayerMovement.cs 5. dio*

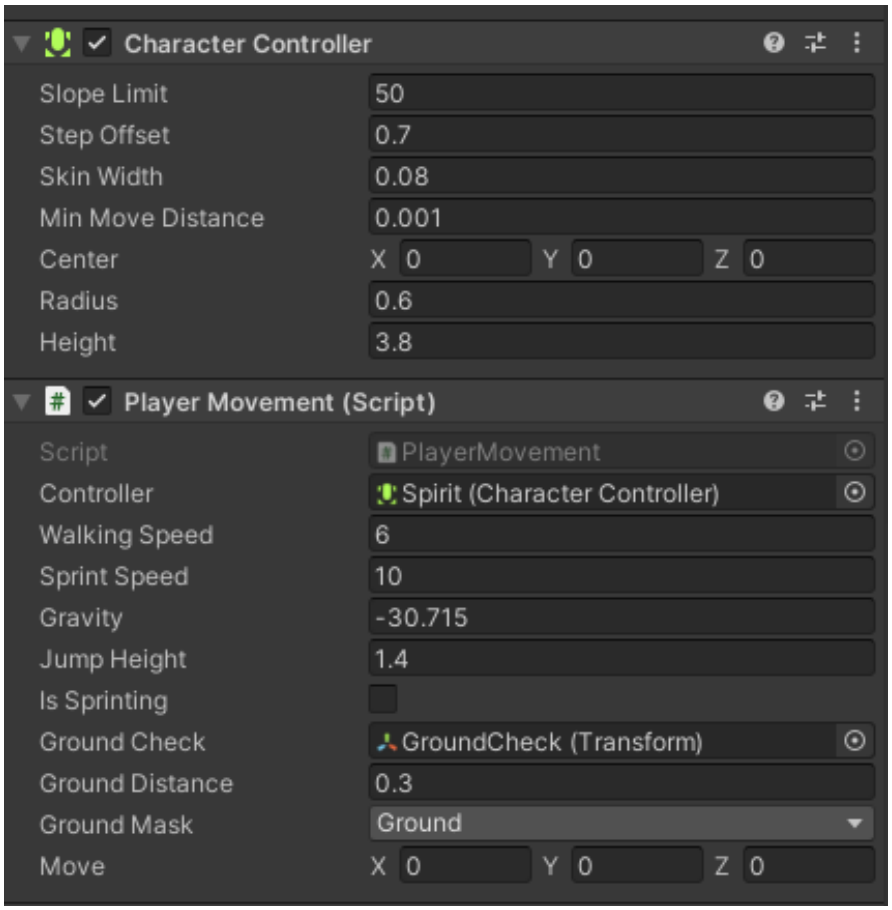
Ako igrač pritisne „Space“ gumb na tipkovnici i isGrounded je istinita, dakle igrač je na zemlji, onda y os od velocity vektor varijable je jednak korijenu visine skoka pomnoženim s gravitacijom i -2f. Tako se dobiva učinak skoka. Onda nakon toga, na svakom frame-u se zbraja ista varijabla s gravitacijom i Time.deltaTime. To simulira gravitaciju i padanje nakon skoka. Napokon nakon izračunavanja svega, to se ubacuje u controller.Move() funkciju i ponovno množi s Time.deltaTime da bi se zaustavila varijacija različitih računalnih performansi. Ako se drži bilo koji od gumbova za pokretanje lika, to znači ako je bilo koja od vrijednosti osi za pokretanje različita od nule, onda varijabla IsMoving je jednaka true. Ako su osi za pokretanje jednake 0, onda je IsMoving jednak false. U isječku koda koji je prikazan ispod, prikazuje se reguliranje zvuka.

```
        if (IsMoving && !audioSource.isPlaying) audioSource.Play(); // if player
is moving and audiosource is not playing play it
        if (isSprinting)
        audioSource.pitch = 1f;
        else
        audioSource.pitch = 0.63f;

        if (!IsMoving || !isGrounded) audioSource.Stop();  }
```

*Isječak Koda 8 Skripta PlayerMovement.cs 6. dio*

Ako se lik pokreće i ako audioSource ne pušta zvuk, onda se pušta zvuk koraka. Ako igrač drži lijevi shift gumb, onda se povisi visina i brzina tona da bi se doprinijelo osjećaju trčanja, a ako igrač hoda onda se snizi.

Ako se lik ne pokreće ili ako nije na zemlji, zaustavlja se zvuk koraka. Na slici 5. vide se javne varijable od skripti CharacterController.cs i PlayerMovement.cs.




*Slika 5 Javne varijable od CharacterController i PlayerMovement skripti*

Vidi se da character controller ima zadane javne varijable na koje se može utjecati. SlopeLimit je stupanj krivine na koju se lik može popeti, height je visina collider-a u obliku kapsule od lika, minMoveDistrance je najmanja distanca koju lik može prijeći u jednom koraku. Radius je poluprečnik collider-a u obliku kapsule od glavnog lika, skinWidth je debljina „kože“/površine kolizije, stepOffset je varijabla koja kontrolira visinu „stepenice“ na koju se lik može popeti, „center“ pokazuje centar kapsule od lika relativno na transform poziciju.

### 5.1.3 Život Igrača

PlayerHealth je skripta koja upravlja sa životnim stanjem glavnog lika. Kad život lika bude jednak nuli, to znači da lik umire i pokreće se death scene (hrv. scena smrti).

Scene u unity-u predstavljaju prostor u kojem se radi sa sadržajem. To su asset-i koji sadržavaju cijelu ili dio igrice ili aplikacije. Može se napraviti cijela igrica u jednoj sceni a mogu se i rasporediti scene po nivoima u igrici. Kuroi Yuki je napravljen u više scena, tako da je igrica u Main Scene, a ostale scene predstavljaju MainMenu, i 3 različite scene za smrt glavnog lika. U isječku koda koji je prikazan u nastavku počinje objašnjenje PlayerHealth.cs skripte.

```
using UnityEngine.SceneManagement;

    public float maxHealth = 100;
    public float currentHealth;

    public void Start()
    {
        currentHealth = maxHealth;
    }

    public void TakeDamage(float damage){

    currentHealth-=damage;
    if(currentHealth<=0){
        SceneManager.LoadScene("Death Scene");
    }
    }
```

*Isječak Koda 9 Skripta PlayerHealth.cs 1. dio*

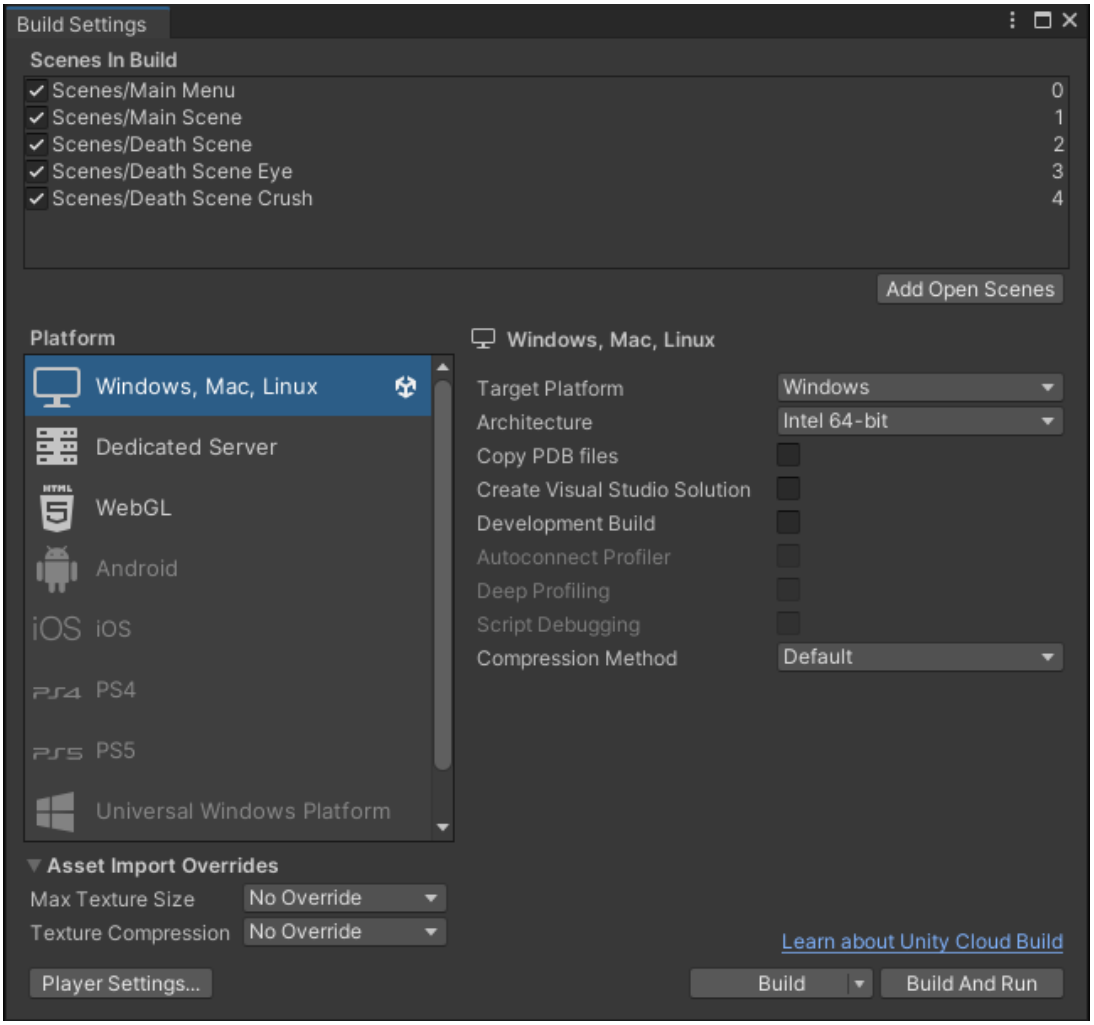
Vidi se da se koristi UnityEngine.SceneManagment klasa koja sadržava funkcije za promjene scena. To je potrebno da bi se učitale scene koje sadržavaju videe koji predstavljaju smrti glavnog lika.

Postoje dvije javne (eng. public) varijable tipa float koje predstavljaju maksimalne životne poene glavnog lika i trenutne životne poene. Na početku, u Start() funkciji jednači se trenutni život s maksimalnim što ima smisla jer cilj je da lik bude potpuno „zdrav“ i neoštećen pri učitavanju scene.

Pravi se javna funkcija povratnog tipa void, TakeDamage() koja će biti pozivana iz drugih skripti da bi oštetila glavnog lika po potrebi. To se radi tako što se kroz argument damage kod poziva funkcije oduzima ta vrijednost od trenutnog života glavnog lika.

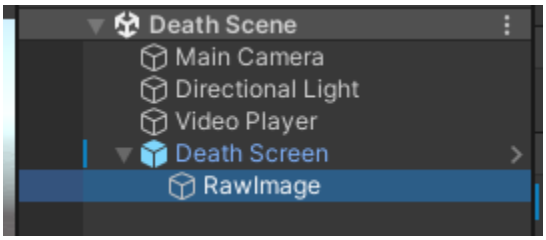
Ako trenutni život dostigne nulu ili bude ispod nule, onda se poziva LoadScene() funkcija kojoj je kao argument dan („Death Scene“) što igrača vodi u istoimenu scenu. Scene se mogu imenovati i staviti u stanje u kojem ih je moguće mijenjati kroz skripte tako što se klikne na File/BuildSettings i onda se klikne na „Add Open Scenes“ za svaku postojeću scenu. Tako se stavljaju u niz (eng. array) scena kroz koji se može iterirati i pristupiti njegovim pojedinačnim elementima.

Na slici ispod vide se „build“ postavke i scene koje se nalaze u igrici Kuroi Yuki.




*Slika 6 Build Settings od igrice Kuroi Yuki*

Scene za smrt glavnog lika su veoma slične te će biti prikazana hijerarhija jedne od njih. Jedina razlika između njih je mp4 video koji se pokreće pri tom specifičnom tipu smrti glavnog lika. U nastavku je prikazana hijerarhija scena smrti.




*Slika 7 Hijerarhija Death Scenes u igrici Kuroi Yuki*

U video playeru postoji video klip koji je napravljen specifično za ovu smrt lika. Taj video se pokrene na inicijalizaciji scene jer je aktivirana „Play On Awake“ opcija. Ovaj mp4 video je projiciran na platno (eng. canvas) koje kao dijete element ima „raw image“ s teksturom iscrtavanja (eng. render texture). Pri završetku videa prelazi se nazad na zadnje sačuvano stanje igrice, sustav sačuvanja napretka kroz igricu te sustav upravljanja scena smrti će biti opisani u sekciji „menadžeri igrice“.

### 5.1.4 Njihanje Kamere

Skripta Headbob.cs se koristi za pokretanje njihanja glave lika, to čini dosta više realistično kretanje glavnog lika u perspektivi iz prvog lica. Gledajući da je igrica opisana u ovom radu horor žanra, ova funkcionalnost pomaže s jezivim aspektom sporog hodanja po strašnom krajoliku. Definiranje varijabli u Headbob.cs skripti je prikazano u isječku ispod.

```
public float walkingBobbingSpeed = 14f;
public float sprintingBobbingSpeed = 16f;
 public float bobbingAmount = 0.05f;
 public PlayerMovement controller;
 float defaultPosY = 0;
 float timer = 0;
 float timer2 = 0;
```

*Isječak Koda 10 Skripta Headbob.cs 1. dio*

Na početku se instancira nekoliko javnih varijabli i dodijeljene su im početne vrijednosti. WalkingBobbingSpeed varijabla utječe na brzinu njihanja glave/kamere glavnog lika dok igrač hoda, dok iduća varijabla (sprintingBobbingSpeed) utječe na isti aspekt ali dok igrač

drži lijevi shift. Naravno referencira se i PlayerMovement skripta jer se mora pristupiti istoj da bi bila dobivena informacija o tome da li glavni lik trči ili hoda. Isto tako ima i bobbingAmount varijabla koja utječe na jačinu/visinu pokreta kamere/glave tijekom instance njihanja. Drugi dio Headbob.cs skripte je prikazan u nastavku.

```
    void Start()
    {
        defaultPosY = transform.localPosition.y;
    }

    void Update()
    {
        if(Mathf.Abs(controller.move.x) > 0.1f || Mathf.Abs(controller.move.z) >
0.1f)
        {
            //Player is moving
            timer += Time.deltaTime * walkingBobbingSpeed;
            timer2 += Time.deltaTime * sprintingBobbingSpeed;
             if(controller.isSprinting==true){
            transform.localPosition = new Vector3(transform.localPosition.x,
defaultPosY + Mathf.Sin(timer2) * bobbingAmount, transform.localPosition.z);
            }
            else{
            transform.localPosition = new Vector3(transform.localPosition.x,
defaultPosY + Mathf.Sin(timer) * bobbingAmount, transform.localPosition.z);
            }

        }
        else
        {
            //Idle
            timer = 0;
            transform.localPosition = new Vector3(transform.localPosition.x,
Mathf.Lerp(transform.localPosition.y, defaultPosY, Time.deltaTime *
walkingBobbingSpeed), transform.localPosition.z);
        }
    }
```

*Isječak Koda 11 Skripta Headbob.cs 2. dio*

Prvo u Start() funkciji se uzima početna pozicija kamere/lika na y osi. Onda na svakom frame-u se provjerava da li lik hoda ili trči. Imaju 2 brojača vremena koji služe kao argument u sinusnoj funkciji i zapravo predstavljaju kut u radijanima.

Ako lik trči, bude instanciran Vektor3 objekt, kao parametre mu je dana trenutna pozicija na X osi, početna pozicija na Y osi zbrojena sa Mathf.Sin() funkcijom kojoj je dan kao argument prikladni brojač vremena i pomnožen je s bobbingAmount varijablom. Mathf.Sin() vraća floating point vrijednost u rasponu od -1 do 1. Kao treći parametar dana je trenutna pozicija na osi Z. Ovo znači da ovaj Vector3 utječe samo na Y(vertikalnu) os. Vrijednost ovog vektora biva dodijeljena lokalnoj poziciji GameObject-a što znači da utječe na taj GameObject.

Ako igrač sprinta, odnosno drži lijevi shift, radi se ista stvar samo što je dodijeljen drugi timer koji je izračunat koristeći sprintingBobbingSpeed naspram walkingBobbingSpeed koji je korišten u slučaju da igrač hoda.

Na kraju kad se igrač prestane pomjerati, timer se resetira na 0 i lokalnoj poziciji bude dodijeljena vektorska vrijednost koja lika vraća u standardnu poziciju. To znači da se koristi defaultPosY za vraćanje na standardno. Koristi se i Mathf.Lerp() funkcija da bi lika vratili u poziciju kroz vrijeme umjesto da se to uradi instantno, jer da se uradi bez korištenja ove funkcije, svaki put kad bi igrač pustio tipku left shift, „teleportirao“ bi se na početnu poziciju te bi to izgledalo sasvim neprirodno.

Funkcija Mathf.Lerp() vraća interpoliranu float vrijednost između dvije float vrijednosti. Te dvije float vrijednosti su trenutna pozicija glave/kamere i defaultPosY pozicija što je početna pozicija. To se množi s Time.deltaTime i walkingBobbingSpeed da bi bio postignut taj glatki prijelaz.

Ova skripta se koristi na glavnoj kameri, na drugoj kameri i na bell objektu koji će biti objašnjeni kasnije u radu.

### 5.1.5 Gledanje u Objekt

Look at target skripta omogućava da se preusmjeri pogled glavnog lika na neki važan događaj u igrici. Ova funkcionalnost se u Kuroi Yuki koristi više puta. Na početku igrice potrebno je da igrač pogleda u veliku nuklearnu eksploziju da bi bila naglašena dramatična i jeziva atmosfera. Druga instanca korištenja ove skripte se ponavlja više puta i to je kod jednog od glavnih neprijatelja u igrici, The Smoldering Gazer, taj koncept će biti daljnje objašnjen kasnije u radu.

U razvoju Kuroi Yuki često su korištene korutine (eng. coroutines). Korutina je način da se zadaci riješe preko više frame-ova. U unity-u, korutina je metoda koja zaustavlja izvršavanje i vraća kontrolu Unity-u ali onda nastavlja gdje je stala na idućem frame-u. Korutine se mogu koristiti i da se nastavi izvršavati dio koda sve dok se ne specificira zaustavljanje izvršavanja. Korutine sadržavaju „yield“ instrukciju koja će čekati određeno vrijeme koje bude specificirano (Carrillo, 2022).

U ovom radu, korutine su korištene u više slučajeva. Svaki put kad treba nešto izvršavati izvan pojedinačnih frame-ova, kao na primjer, popunjavanje cooldown trake nakon korištenja sposobnosti „minds eye“ koju igrač posjeduje, čekanje određeni period prije pokretanja specifične animacije, itd. Sve instance korištenja korutina će biti spomenute i objašnjene u radu. Bit će pokrenuto objašnjavanje s prvom instancom koju koristimo u LookAt() korutini. U isječku ispod pokazano je definiranje varijabli.

```
    public Transform Target;
    public Transform TargetEye;
    private Coroutine LookCoroutine;
```

*Isječak Koda 12 Skripta LookAtTarget.cs 1. dio*

Na početku LookAtTarget.cs skripte definirane su dvije javne i jedna privatna varijabla. Target varijabla je referenca na GameObject nuklearne eksplozije na koju je potrebno da igrač pogleda pri početku igrice. TargetEye je oko od neprijatelja „Smoldering Gazer“ u kojeg je potrebno da igrač gleda periodično u borbi protiv tog neprijatelja.

LookCoroutine je privatna varijabla tipa Coroutine koja sadrži referencu na korutinu koju kasnije definiramo. U isječku koda 13, prikazana je StartRotating() funkcija.

```
public void StartRotating(string targetSelection, float Speed, float
lookingAtTime )
    {
        if(targetSelection == "nuke")
        {
            if(LookCoroutine != null)
            {
                StopCoroutine(LookCoroutine);
            }
            LookCoroutine = StartCoroutine(LookAt(Target, Speed, lookingAtTime));
            AudioManager.Instance.Play(SoundType.LookAtNuke);
        }
         else if(targetSelection == "eye")
        {
            if(LookCoroutine != null)
            {
                StopCoroutine(LookCoroutine);
            }
            LookCoroutine = StartCoroutine(LookAt(TargetEye, Speed,
lookingAtTime));
        }
    }
```

*Isječak Koda 13 Skripta LookAtTarget.cs 2. dio*

Definirana je javna funkcija povratnog tipa void StartRotating() koja će biti pozivana iz drugih skripti po potrebi gledanja u specifični GameObject te su joj definirana 3 argumenta. TargetSelection koji je tipa string i označava objekt u koji je potrebno gledati, float Speed što je brzina okretanja igrača prema tom objektu i float lookingAtTime što označava vrijeme koje je igrač forsiran da gleda u taj objekt. Okretanje igrača prema objektu je veoma korisno za horror igrice jer omogućava veću kreativnost kod zastrašivanja igrača.

U slučaju Kuroi Yuki, ova funkcija je korištena u 2 slučaja. Prvi slučaj se desi kad je targetSelection jednak „nuke“. Tako je označeno funkciji da je potrebno da lik gleda u nuklearni oblak koji je sveprisutan u igrici. Drugi je slučaj kad like gleda u „eye“ što je oko od neprijatelja „Smoldering Gazer“, cijela funkcionalnost ovog neprijatelja će biti objašnjena u radu.

U oba slučaja ima veoma sličan slijed naredbi. Prvo je provjereno da li LookCoroutine nije jednaka null. Znači provjereno je da li postoji već pokrenuta korutina, ako postoji onda

biva zaustavljena koristeći StopCoroutine() funkciju. Ne trebaju imati dvije iste korutine pokrenute u isto vrijeme.

Nakon toga korutina LookCoroutine bude dodijeljena varijabli tako što je pokrenuta koristeći StartCoroutine() funkciju, kao argument ovoj funkciji je dana LookAt() korutina koja je u nastavku definirana i prikazana.

```
    private IEnumerator LookAt(Transform targetSelection, float Speed, float
lookingAtTime)
    {
        Quaternion lookRotation =
Quaternion.LookRotation(targetSelection.position - transform.position);
        float time = 0;
        while (time < lookingAtTime)
        {
            transform.rotation = Quaternion.Slerp(transform.rotation,
lookRotation, time*Speed);

            time += Time.deltaTime;
            yield return null;
        }
    }
```

*Isječak Koda 14 Skripta LookAtTarget.cs 3. dio*

Korutine se definiraju koristeći IEnumerator ključnu riječ. Ova korutina je nazvana LookAt() i definirana su joj tri argumenta, Transform targetSelection što je pozicija objekta u koju igrač treba gledati, float Speed što je brzina kojom je igrač okrenut prema objektu i float lookingAtTime što je vrijeme gledanja u objekt.

Quaternion je struktura unutar Unity-ja koja predstavlja rotacije. Quaternion.LookRotation() je funkcija koja omogućava da se jedan GameObject rotira prema drugom, to jeste da ga „gleda“. Quaternion.Slerp() je funkcija koja predstavlja sferičnu rotaciju između 2 objekta. Može se koristiti Slerp da se rotira objekt određenom brzinom kroz određeno vrijeme (French, 2021).

Prvo se dodjeljuje rotacija u varijablu lookRotation tako što se koristi Quaternion.LookRotation() funkcija. Kao argument, bude dodijeljena razlika između pozicije objekta u kojeg je potrebno da igrač gleda i pozicija našeg glavnog lika. Ovo vraća rotaciju koja treba biti izvršena da bi se ovo postiglo.

Dok je vrijeme manje od vremena gledanja, znači da je ograničeno izvršenje ove rotacije na vrijeme koje je određeno za gledanje. Sve dok je to istina, rotira se GameObject glavnog lika sferično prema ciljanom objektu koristeći Quaternion.Slerp() funkciju. Njoj su kao argumenti dani trenutna rotacija glavnog lika, rotacija prema objektu koja je prije izračunata i vrijeme pomnoženo s brzinom gledanja.

Vrijeme je zbrojeno s Time.deltaTime što rješava problem ubrzanog vremena zbog boljih performansi te je napisana instrukcija yield return null. U ovom slučaju to znači da se nastavlja na idući frame. Ovime je postignuto rotiranje tijela prema ciljanom objektu, pozvana je StartRotating() funkciju iz drugih skripti po potrebi, ovo će biti prikazano u nastavku.

### 5.1.6 Sudaranje Igrača

U Unity-ju postoje dvije verzije registriranja sudaranja objekata. Može se koristiti collideri i okidači (eng. triggers). U razvoju Kuroi Yuki, najkorisniji su se pokazali okidači. Collideri koriste funkciju OnCollisionEnter() da bi prepoznali sudar dva GameObject-a. Oba objekta moraju imati collider-e te da bi ovo funkcioniralo jedan od objekata mora imati rigidbody komponentu spojenu na sebe.

Od collidera se prave okidači tako što se aktivira isTrigger opcija u editoru. Okidači koriste OnTriggerEnter() funkciju te se smatraju „prolaznom kolizijom“. Objekti se ne odbijaju jedno od drugo ali događaji se još uvijek mogu okidati kad se kontakt desi (Amilin, 2021). Radi kompaktnosti, neće biti prikazana cijela PlayerCollision.cs skripta već ključni elementi. Ostatak funkcionira na isti princip kao što je prikazano. Prvobitno je u nastavku prikazano definiranje varijabli u PlayerCollision.cs skripti.

```
public LookAtTarget lookAtTargetScriptGameObject;
  public Raycast raycastScriptGameObject;
  private bool smolderingGazerActive = false;
  private Coroutine smolderingGazerCoroutine;
  public float timeBetweenEyePulses = 1.5f;
  public CameraShake cameraShake;
  public AudioMixer audioMixer;
  public PlayerMovement playerMovementScriptObject;
  public float slowedDownWalkingSpeed;
  public float slowedDownSprintSpeed;
```

```
    public float spedUpWalkingSpeed;
    public float spedUpSprintSpeed;
    private bool windPlaying = true;
    public Animator finalCameraCutsceneAnimator;
    public GameObject finalCutsceneCamera;
    public GameObject mainCamera;
    public GameObject fadeInCanvas;
    public GameObject crosshair;
    private Coroutine WaitForSecondsThenDoCoroutine;
    public GameObject eyeLight;
    public Material eyeMaterial;
    [ColorUsageAttribute(false,true,0f,8f,0.125f,3f)]
    public Color eyeColorActive;
    [ColorUsageAttribute(false,true,0f,8f,0.125f,3f)]
    public Color eyeColorInactive;
```

*Isječak Koda 15 Skripta PlayerCollision.cs 1. dio*

Vidi se da postoji puno varijabli jer se ova skripta koristi za sve vrste kolizija u igri. U idućem isječku koda bit će prikazana OnTriggerEnter() funkcija.

```
// collision (triggers, not colliders)
    void OnTriggerEnter(Collider collision){
        if(collision != null && collision.gameObject.CompareTag("BlackSnow"))
            {
                GameManager.Instance.blackSnowEvent(collision.gameObject);
            }
        if(collision != null && collision.gameObject.CompareTag("LookAtNuke"))
            {
                lookAtTargetScriptGameObject.StartRotating("nuke", 0.2f, 2f);
                collision.gameObject.SetActive(false);
            }
        if(collision != null && collision.gameObject.CompareTag("CrushDeath"))
            {
                SceneManager.LoadScene("Death Scene Crush");
            }
        if(collision != null && collision.gameObject.CompareTag("SaveData"))
            {
                GameManager.Instance.SaveData();
                Debug.Log("Data saved.");
            }
    }
```

*Isječak Koda 16 Skripta PlayerCollision.cs 2. dio*

Koristi se funkcija OnTriggerEnter() da se sazna kad igrač dotakne neki okidač s tijelom glavnog lika. Zna se točno što je dotaknuto tako što se uspoređuju etikete (eng. tags) od

GameObjekata. Ako je etiketa od tog okidača „BlackSnow“ onda se pozove blackSnowEvent() funkcija (bit će objašnjeno u black snow sekciji).

Ako igrač dodirne okidač i njegova etiketa je „LookAtNuke“ onda se poziva StartRotating() funkcija koja je objašnjena u prijašnjem segmentu. Nakon pozivanja ove funkcije poziva se SetActive() funkcija i dan mu argument false, to čini da ovaj okidač bude jednokratan i nestane nakon što uradi ono što treba. Nije u cilju da svaki put kad igrač prođe s likom pored okidača da pogleda u nuklearni oblak.

### 5.1.7 Protresanje Kamere

Protresanje kamere potiče osjećaj nelagode i straha te čini velik ali ne dovoljno cijenjen dio razvijanja horror igrice. Gubitak kontrole nad kamerom, iako na kratko, čini cijelo iskustvo više jezivim, što je jedan od glavnih ciljeva horor igrica. U nastavku će biti prikazan CameraShake.cs skripta.

```
    public GameObject mainPostProcessing;
    public GameObject jumpscarePostProcessing;

    public IEnumerator Shake(float duration, float magnitude, bool playSound)
    {
        Vector3 originalPos = transform.localPosition;
        float elapsed = 0.0f;
        mainPostProcessing.SetActive(false);
        jumpscarePostProcessing.SetActive(true);
        if(playSound)
        {
            float random = Random.value;
            if(random < .3)
            AudioManager.Instance.Play(SoundType.Jumpscare1);
            else if(random < .7)
            AudioManager.Instance.Play(SoundType.Jumpscare2);
            else
            AudioManager.Instance.Play(SoundType.Jumpscare3);
        }
```

*Isječak Koda 17 Skripta CameraShake.cs 1. dio*

Ovdje se nalazi još jedna korutina koja je korištena u ovom projektu. Korištena je jer je potrebno čekati da određeno vrijeme prođe prije nego što se ekran prestane tresti. Na početku je uzeta originalna pozicija objekta kojeg treba protresti. Aktivira se naknadna

obrada (eng. post processing) koja je zaslužna za trenutke prepadanja. Naknadna obrada će biti objašnjena u jednom od idućih segmenata ovog rada. Nakon toga je nasumično pušten jedan od 3 jeziva zvučna efekta. Shake korutina se može pozvati iz bilo koje druge skripte, poziva se po potrebi. U isječku ispod se vidi nastavak LookAtTarget.cs skripte.

```
        while(elapsed < duration)
        {
            float x = Random.Range(-1f, 1f) * magnitude;
            float y = Random.Range(-1f, 1f) * magnitude;
            transform.localPosition = new Vector3(x, y, originalPos.z);

            elapsed += Time.unscaledDeltaTime;
            yield return null;
        }
        mainPostProcessing.SetActive(true);
        jumpscarePostProcessing.SetActive(false);
        if(playSound)
        {
            AudioManager.Instance.Stop(SoundType.Jumpscare1);
            AudioManager.Instance.Stop(SoundType.Jumpscare2);
            AudioManager.Instance.Stop(SoundType.Jumpscare3);
        }
        transform.localPosition = originalPos;
    }
```

*Isječak Koda 18 Skripta CameraShake.cs 2. dio*

Sve dok je proteklo vrijeme manje od trajanja ovog prepada, trese se monitor tako što se računa nasumična pozicije te se aplicira na lokalnu poziciju našeg lika. Na kraju originalna naknadnu obradu bude vraćena, zvukovi su zaustavljeni te je vraćen pogled na originalnu poziciju.

### 5.1.8 Raycast

Raycast u Unity-u je fizička funkcija koja projicira zraku (eng. ray) u scenu, i vraća boolean vrijednost ako je meta pogođena. Kad se ovo desi, informacije o tom događaju kao distanca pozicija i referenca na Transform komponentu od objekta, mogu biti sačuvani u Raycast Hit varijablu za daljnje korištenje (French, 2021).

U ovom projektu je korišten Raycast da bi bilo određeno kad igrač gleda u bilješke na podu. Bilješke su papiri na kojima se nalaze monolozi od NPC-ova(non playable characters) u igrici koji propagiraju priču i uvode psihološki horor aspekt u Kuroi Yuki.

Ovaj sustav će biti opisan u UI sekciji. Koristi se Raycast isto da bi bilo određeno kad igrač gleda u oko od Smoldering Gazer-a, jer kad igrač pogledamo u oko, onda se okida tranzicija na EyeDeathScene scenu. Još jedna instanca korištenja Raycasta u Kuroi Yuki je kod aktiviranja poluge koja otvori vrata od tunela. Isto tako s Raycast-om bivaju podignuti Cursed Cube i Minds Eye Bell objekti s poda. U isječku ispod se vidi Update() funkcija.

```
void Update()
    {
        var ray = PlayerCamera.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;

        if(!pauseMenu.GetComponent<PauseMenu>().returnPausedState())
        if(Physics.Raycast(ray, out hit, RaycastRange, selectableLayer))
        {
            var selection = hit.transform;
            var selectedObject = selection.gameObject;
            var selectionRenderer = selection.GetComponent<Renderer>();

            if (!doOnce)
                {
                    CrosshairChange(true);
                }
```

*Isječak Koda 19 Skripta Raycast.cs 1. dio*

U Raycast.cs skripti su definirane razne varijable, njihova definicija nije prikazana jer može se može zaključiti kroz razumijevanje funkcija i njihovo korištenje u funkcijama. ScreenPointToRay() je funkcija koja vraća zraku (eng. ray) koja ide iz kamere kroz točku

u ekranu. Kursor je zaključan u sredini ekrana tako da Raycast uvijek pokazuje na ono što igrač gleda u igrici.

Ova funkcija se koristi i rezultat bude stavljen u ray varijablu. Nakon toga bude provjereno da li je igrica zaustavljena i ako nije, nastavlja se skripta. Nakon toga postoji Physics.Raycast() funkcija u kojoj je razvijen ostatak logike.

Ova funkcija izvršava komande definirane u njoj ako igrač gleda u interaktivni objekt. Kad igrač gleda u interaktivni objekt, pozove se samo jednom, kasnije definirana funkcija CrosshairChange() koja promijeni boju nišana (eng. Crosshair) u crvenu da indicira da igrač gleda u objekt na kojeg može uticati. U isječku ispod se vidi nastavak Raycast.cs skripte.

```
            // when looking at notes
            if (selection.CompareTag(interactableTags[0]))
            {
                if(Input.GetKeyDown(KeyCode.E) ||
Input.GetKeyDown(KeyCode.Escape))
                {
                    if(!isPaused && Input.GetKeyDown(KeyCode.E))
                    {
                       notes.GetComponent<NoteSystem>().showNote(selectedObject);
                        audioMixer.SetFloat("WalkingVolume", -80f);
                        PauseMovement();
                    }
                    else if(isPaused && (Input.GetKeyDown(KeyCode.E) ||
Input.GetKeyDown(KeyCode.Escape)))
                    {
                       notes.GetComponent<NoteSystem>().hideNote(selectedObject
                        audioMixer.SetFloat("WalkingVolume", 0f);
                        ResumeMovement();
                    }
                }
            }
```

*Isječak Koda 20 Skripta Raycast.cs 2. dio*

Bit će prikazan samo jedan slučaj detektiranja specifičnih objekata jer je za ostale korištena ista logika. U ovom dijelu koda je testirano koji tag ima GameObject u kojeg igrač gleda, ako ima specifični tag (u ovom slučaju „note“), onda se testira da li je igrač pritisnuo gumb E ili ESCAPE gumb na tipkovnici, nakon toga, ako igrica nije zaustavljena

i igrač je pritisnuo gumb E, onda je pozvana showNote() funkcija koja prikazuje bilješku (sustav bilješki će biti objašnjen kasnije u radu). Zvuk hodanja se sasvim utiša i pozove se PauseMovement() funkcija kako bi se zaustavila sva mogućnost pokretanja dok igrač čita bilješku. U protivnom, ako je igrica zaustavljena i igrač pritisne E ili ESC gumb, pozove se hideNote() funkcija koja sakriva bilješku, pojačan je zvuk hodanja i omogućena je sposobnost hodanja tako što se poziva ResumeMovement() funkcija. U isječku koda u nastavku je prikazana logika smrti od neprijatelja.

```
// when looking at the eye
            if (selection.CompareTag(interactableTags[1]))
            {
                if(!doOnce)
                {
                    Debug.Log("Dead by Eye!");
                    SceneManager.LoadScene("Death Scene Eye");
                }
            }
```

*Isječak Koda 21 Skripta Raycast.cs 3. dio*

Ako igrač gleda u oko od Smoldering Gazer neprijatelja, onda se scena promijeni na „Death Scene Eye“ u kojoj se na početku scene pokrene video koji je specifičan za taj tip smrti u igrici. CrosshairChange() funkcija je prikazana u isječku koda ispod.

```
    void CrosshairChange(bool raycastHitting)
    {
        if (raycastHitting && !doOnce)
        {
            crossHair.color = crossHairRed;
        }
        else
        {
            crossHair.color = crossHairGray;
            isCrosshairActive = false;
        }
    }
```

*Isječak Koda 22 Skripta Raycast.cs 4. dio*

CrosshairChange() funkcija služi da bi se promijenila boja nišana kad igrač gleda u interaktivni objekt, kad prestane gledati, boja se vrati u zadanu.

## 5.2 Mind's Eye

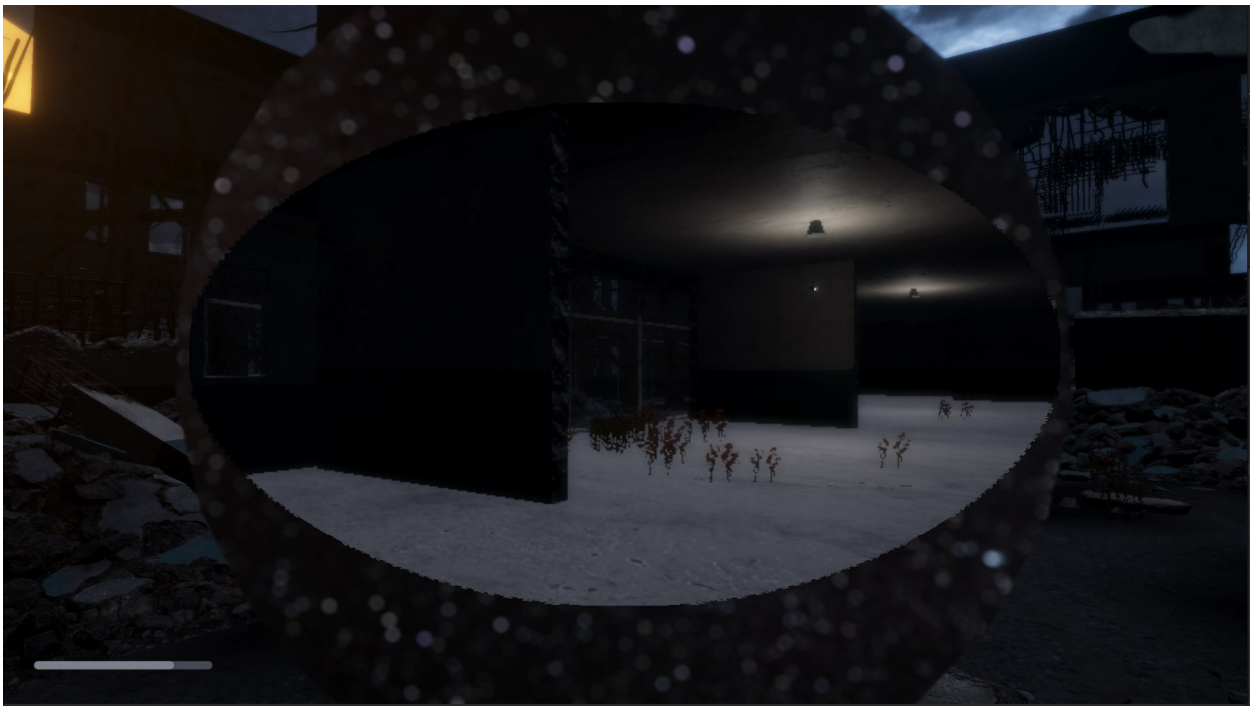
Mind's eye je ključna mehanika u igrici Kuroi Yuki, ta mehanika omogućava igraču da pogleda u alternativni svijet koji je manje uništen nego trenutni. Kroz ovu sposobnost, igrač može djelovati na taj svijet tako da rješava probleme i napreduje kroz igricu.

Glavni lik aktivira ovu sposobnost kroz objekt nazvan „Mind's Eye Bell“. Nakon što ga pokupi, zazvoni na zvono nakon čega se aktivira sposobnost.

Sustav „drugog svijeta“ koji je paralelan s originalnim napravljen je na način dupliciranja terena/objekata i ostalih elemenata. Cijeli sustav će biti opisan u ovom segmentu. Ova sposobnost ima i svoj „cooldown“ (vrijeme koje se ne može koristiti nakon što se potroši). Način razvijanja logike i funkcionalnosti ove komponente će biti opisana u nastavku. Na slici ispod se vidi sposobnost Mind’s Eye dok još nije aktivirana.

*Slika 8 Mind's Eye Deaktiviran*

U ruci se vidi zvonce, a s lijeve strane se vidi traka koja označava vrijeme koje igrač posjeduje za korištenje ove sposobnosti. Na slici ispod se vidi sposobnost Mind's Eye kad je aktivirana.



*Slika 9 Mind's Eye Aktiviran*

Može se primijetiti da se kroz „oko“ vidi drugi svijet u kojem ima puno tajni i rješenja za zagonetke u igrici.

Za razvoj ove sposobnosti korišteni su brojni elementi. Glavni način na koji ovo funkcionira jeste da postoji kopija okruženja koja je promijenjena da izgleda više industrijalizirano te manje razrušeno. Ova kopija je praćena sa sekundarnom kamerom koja je u hijerarhiji alternativnog svijeta u Unity-ju kreirana. Sekundarna kamera prati glavnu kameru s velikim offsetom na y osi.

Ta kamera šalje informacije o tome što „vidi“ na sliku iscrtavanja (eng. render image) koja je onda stavljana na platno (eng. canvas) i prikazana kroz UI prvoj kameri. Uz to korištene su animacije podizanja zvona i otvaranja oka te animacije spuštanja zvona i zatvaranja oka. Sustav čekanja da se zvono opet „napuni“ pravimo kroz korutine.

Prvo će biti prikazana logika aktiviranja sposobnosti kroz MindsEyeController.cs skriptu u kojoj se nalaze activateEye(), deactivateEye(), useCooldown(), regenCooldown() funkcije i korutine.

U isječku koda ispod prikazuje se definiranje varijabli u MindsEyeController.cs skripti.

```
using UnityEngine.UI;
    public GameObject secondaryCamera;
    public GameObject mindsEye;
    public bool itemActive = false;
    public GameObject[] disableableObjects;
    public Animator eyeMask;
    public Animator bell;
    public bool bellActivatable = false;
    public GameObject pauseMenu;
    // item cooldown
    private Coroutine itemUseCooldownCoroutine;
    private Coroutine itemRegenCooldownCoroutine;
    public Slider cooldownBar;
    public int maxCooldown = 100;
    public int currentCooldown;
    private WaitForSeconds regenTick = new WaitForSeconds(0.1f);
    public float cooldownForRegen = 3f;
    public int cooldownUsagePerTick = 2;
```

*Isječak Koda 23 Skripta MindsEyeController.cs 1. dio*

Korišten je using UnityEngine.UI da bi UI klasa bila unijeta unutar naše skripte. Ta klasa se može koristiti da bi bilo moguće utjecati na okidače za animacije koji se u nastavku skripte aktiviraju. Definirane su razne varijable koje će biti korištene za aktiviranje Mind's Eye sposobnosti. U isječku je prikazana Start() funkcija MindsEyeController.cs skripte.

```
    void Start()
    {
        currentCooldown = maxCooldown;
        cooldownBar.maxValue = maxCooldown;
        cooldownBar.value = maxCooldown;
    }
```

*Isječak Koda 24 Skripta MindsEyeController.cs 2. dio*

U Start() funkciji jednači se trenutni cooldown s maksimalnim. Isto tako, slajderu koji predstavlja preostalo vrijeme korištenja sposobnosti definirana je maksimalna vrijednost i trenutna vrijednost mu je isto tako stavljena maksimalna. Slajder je prikazan u slici ispod.

*Slika 10 Cooldown Slider na kojem je preostalo oko 66% vremena*

Na isječku koda ispod je prikazana Update() funkcija MindsEyeController.cs skripte.

```
void Update()
{
    if(bellActivatable)
    if(!pauseMenu.GetComponent<PauseMenu>().returnPausedState())
    if(Input.GetKeyDown(KeyCode.F))
    {
        if(currentCooldown > 0)
        {
            if(!itemActive)
            {
                activateEye();
            }
            else
            {
                deactivateEye();
            }
        }
        else{
            Debug.Log("Wait for cooldown!!!");
        }
    }
}
```

*Isječak Koda 25 Skripta MindsEyeController.cs 3. dio*

U Update() funkciji na svakom frame-u bude provjereno da li je bellActivatable jednako true. To znači da je provjereno da li je igrač podigao zvonce s poda u igrici i da li ga može koristiti. Nakon toga je provjereno da li je igrica zaustavljena, ako nije onda skripta može nastaviti. Zadnja provjera je da li je igrač pritisnuo gumb F na tipkovnici i da li je trenutni cooldown (hrv. vrijeme hlađenja) veći od nule, ako jeste i sposobnost nije aktivirana, onda funkcija koja ju aktivira je pozvana.

Ako je sposobnost već aktivirana onda funkcija koja ju deaktivira je pozvana. Zadnji else se odnosi na provjeru da li je trenutni cooldown veći od nule, ako nije onda se ispisuje poruka koja upozorava da je isteklo vrijeme za korištenje sposobnosti, te da je potrebno čekati da se napuni.

U nastavku će biti prikazana funkcija čiji je zadatak aktiviranje ove sposobnosti. Funkcija koja deaktivira sposobnost napisana je na isti način samo s obrnutim vrijednostima. Zbog redundancije, neće biti prikazana funkcija za deaktiviranje sposobnosti.

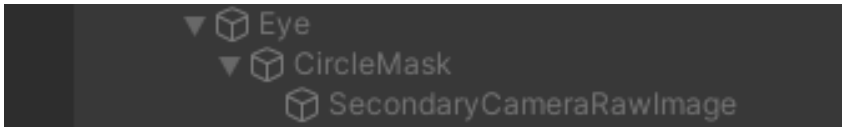
U isječku koda u nastavku, prikazana je activateEye() funkcija.

```
    public void activateEye(){
        if(itemRegenCooldownCoroutine != null)
        StopCoroutine(itemRegenCooldownCoroutine);
        itemUseCooldownCoroutine =
StartCoroutine(UseCooldown(cooldownUsagePerTick));

        secondaryCamera.SetActive(true);
        mindsEye.SetActive(true);
        itemActive = true;
```

*Isječak Koda 26 Skripta MindsEyeController.cs 4. dio*

U javnoj (eng. public) funkciji activateEye(), prvo je provjereno da li je pokrenuta korutina za regeneriranje (punjenje) cooldown-a. Ako jeste, onda će biti zaustavljena jer nije cilj više regenerirati cooldown kad se pokrene sposobnost. Odmah nakon toga se pokreće UseCooldown() korutina koja počne trošiti cooldown od sposobnosti.

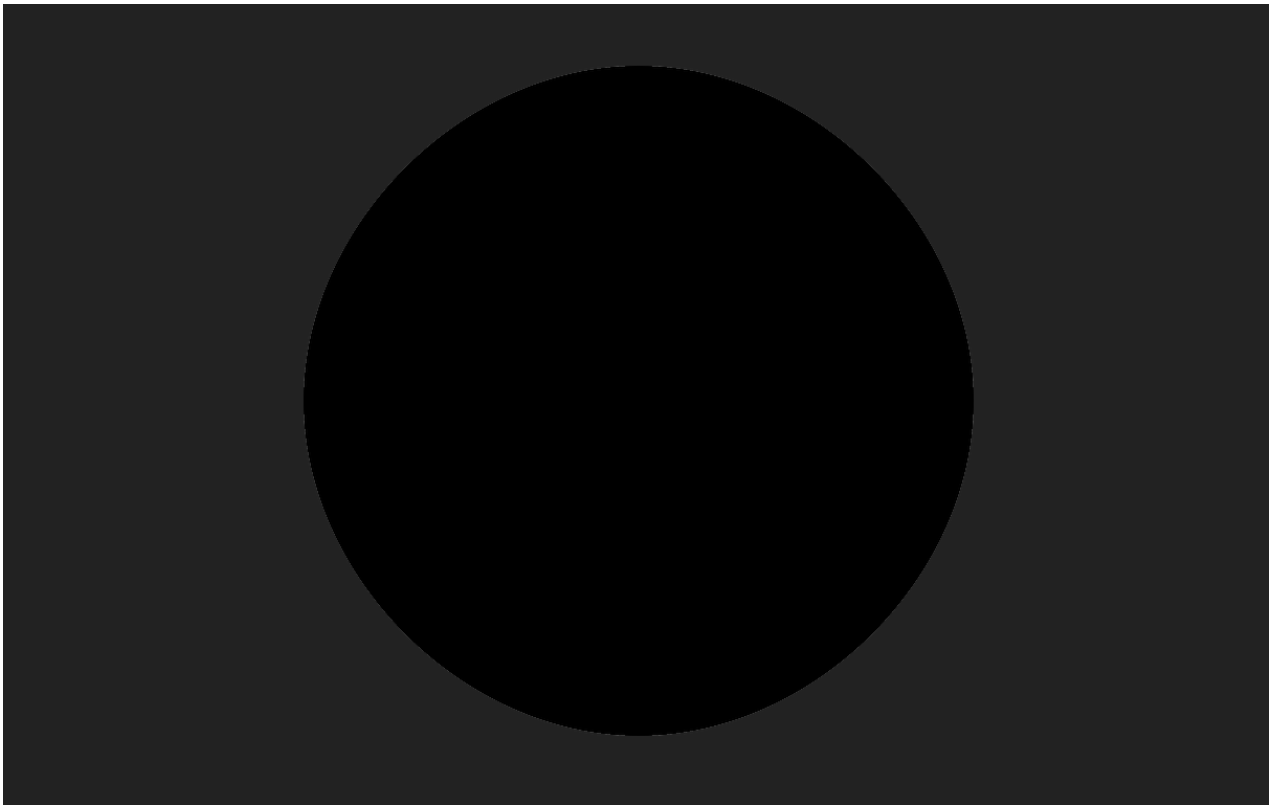
Na kopiji od okruženja koja se zove „Innocent Realm“ koja će biti opisana u „okruženje“ sekciji, postoji druga kamera koja je korištena da bi taj drugi svijet bio prikazan dok se koristi sposobnost. Ova kamera je aktivirana i isto tako je aktiviran mindsEye objekt. MindsEye varijabla predstavlja UI platno koje sadrži masku (eng. mask) i raw image. Ova dva elementa se koriste zajedno da bi se postigao efekt „oka“. U slici 11 prikazana je hijerarhija Mind's Eye GameObject-a.




*Slika 11 Hijerarhija Mind's Eye GameObject-a*

CircleMask je platno koje sadržava Mask komponentu, sliku kruga s PNG ekstenzijom i Animator komponentu. Mask komponenta je način na koji modificiramo izgled child (hrv. dječijih) elemenata objekta koji posjeduje mask komponentu.

Maska ograničava child (hrv. dječije) elemente tako da budu u obliku parent (hrv. roditeljskog) elementa. Znači ako je dijete veće nego roditelj onda će se samo dio djeteta vidjeti, onaj dio koji fizički stane unutar roditelj elementa. Slika koja je maska, dakle slika koja ograničava dijete je slika kruga s ekstenzijom PNG te je prikazana u nastavku.

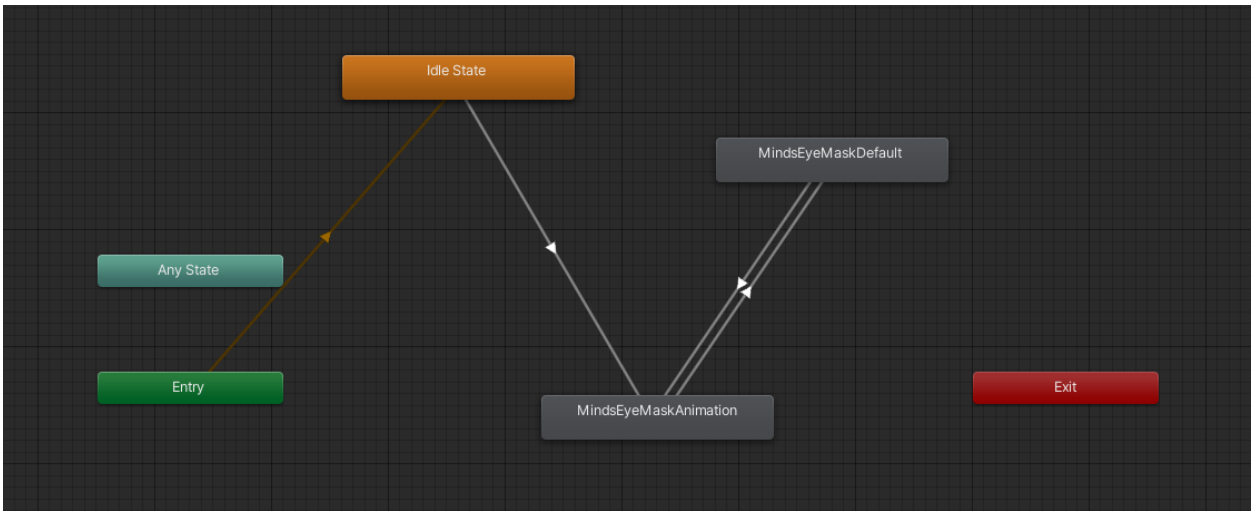


*Slika 12 Maska koja ograničava Mind's Eye*

U Unity-u, animator je komponenta koja sadrži upravljač animatora (eng. animator controller). Upravljač animatora sadrži animacije te omogućava da se uređuju i održavaju setovi animacija (animation clips) i prave tranzicije između njih koje se mogu pokrenuti pomoću okidača (eng. triggers). U većini slučajeva normalno je da postoji više animacija i da je cilj prelaziti iz jedne u drugu.

U slučaju Mind's Eye sposobnosti, postoje dvije animacije. Jedna je ona koja aktivira sposobnost, znači otvori oko a druga je koja deaktivira sposobnost, znači zatvori oko. U isto vrijeme pokreću se i animacija podizanja zvonca i onda pri deaktiviranju sposobnosti pokreće se animacija spuštanja zvonca.

Ispod će biti prikazan upravljač animacija samo za oko jer funkcionira jednako kao animator za zvonce, jedina razlika je da su animacije (eng. animation clips) različite. Nakon toga će biti prikazan nastavak koda u MindsEyeController.cs skripti.




*Slika 13 Animator Controller za Mind's Eye*

```
        eyeMask.SetTrigger("ActiveEyeMask");
        bell.SetTrigger("ActiveBell");
        AudioManager.Instance.Play(SoundType.MindsEyeOpen1);
        foreach (GameObject obj in disableableObjects){
            Collider[] colChildren = obj.GetComponentsInChildren<Collider>();
            foreach(Collider coll in colChildren)
            {
                coll.enabled = false;
            }}}
```

*Isječak Koda 27 Skripta MindsEyeController.cs 5. dio*

Idle State je stanje u kojem je oko zatvoreno. MindsEyeMaskAnimation je animacija za otvaranje oka dok je MindsEyeMaskDefault animacija za zatvaranje oka. Na tranziciji između Idle State i MindsEyeMaskAnimation i tranziciji između MindsEyeMaskDefault i MindsEyeMaskAnimation postoji okidač „ActiveEyeMask“. Na tranziciji između MindsEyeMaskAnimation i MindsEyeMaskDefault se nalazi okidač „InactiveEyeMask“.

U skripti se pokreće animacija uz SetTrigger() funkciju kojoj kao je kao argument dano ime okidača kojeg je cilj pokrenuti. Vidi se da su pokrenute dvije animacije, to su animacija za otvaranje oka i podizanje zvonca. Odmah nakon toga kroz menadžer zvuka (eng. audio menadžer) koji će kasnije biti objašnjen, pokreće se zvuk otvaranja oka (kod

deaktiviranja sposobnosti pušta se zvuk zatvaranja oka). Odmah nakon toga, svi collideri u colChildren listi se isključuju, jer je simulirano prolaženje kroz objekte koji se ne nalaze unutar alternativnog svijeta. Ispod su prikazane UseCooldown() i RegenCooldown() funkcije.

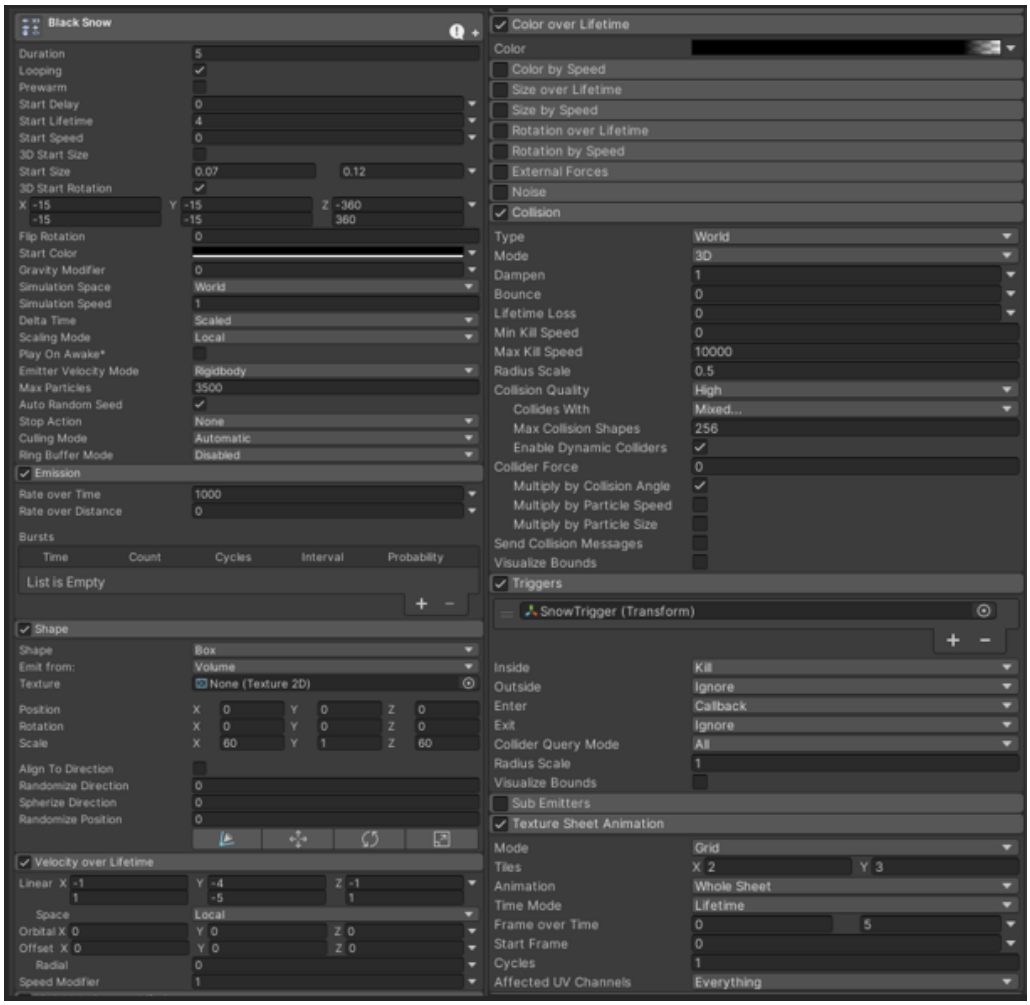
```
IEnumerator UseCooldown(int amount)
{
    while(currentCooldown>0)
    {
    currentCooldown -= amount;
    cooldownBar.value = currentCooldown;
    yield return regenTick;
    }
    deactivateEye();
}
IEnumerator RegenCooldown()
{
    yield return new WaitForSeconds(cooldownForRegen);
    while(currentCooldown<maxCooldown)
    {
    currentCooldown += maxCooldown / 100;
    cooldownBar.value = currentCooldown;
    yield return regenTick;
    }
}
```

*Isječak Koda 28 Skripta MindsEyeController.cs 6. dio*

U nastavku postoje dvije korutine koje upravljaju cooldown-ima. UseCooldown() korutina ima while petlju koja se izvršava sve dok je trenutni cooldown veći od 0. U toj petlji, svaki „regenTick“, što je period između iteracija, smanji trenutni cooldown za količinu koja se šalje kao argument kad korutina bude pozvana. Ova promjenu je prikazana na slajderu tako što mu je promijenjena vrijednost. Na kraju, kad trenutni cooldown dostigne 0, oko bude deaktivirano. Za regeniriranje cooldowna korištena je korutina RegenCooldown() u kojoj se prvo čeka nekoliko sekundi prije regeniriranja, a onda je pokrenuta while petlja koja se izvršava sve dok je trenutni cooldown manji od maksimalnog cooldown-a. Unutar ove petlje, opet koristeći regenTick vrijeme između iteracija, povećava se trenutni cooldown za vrijednost maksimalnog cooldown-a podijeljeno sa 100, znači za 1% svaki „tick“. Nakon tog, ta promjena na slajderu je prikazana tako što mu je vrijednost izjednačena s trenutnim cooldown-om.

## 5.3 Crni Snijeg

Sustav čestica je način da čestice budu simulirane unutar Unity-a. Sustav čestica po default-u emitira čestice u nasumičnim smjerovima, ovdje je cilj da postavke budu namještene na način koji odgovara našoj igrici. U Kuroi Yuki čestice i sustav čestica su korišteni da bi bio simuliran snijeg (crni snijeg). U horor igricama, čestice se često koriste da bi se postigli jezivi efekti uključujući krv, dim, itd. Ovakvi efekti pojačavaju jezivi ambijent te u slučaju Kuroi Yuki stvaraju osjećaj nelagode tako što okružuju glavnog lika s crnim snijegom koji je prijetnja životu glavnog lika. Prvo će biti prikazane postavke ovog sustava čestica a onda će biti prikazane skripte koje upravljaju pokretanjem, zaustavljanjem sustava i pravljenje štete igraču. U slici ispod su prikazane postavke sustava čestica crnog snijega.




*Slika 14 Black Snow Particle System*

Vidi se da je kod općenitih postavki, vrijeme trajanja čestica jednako 5 sekundi. Vidi se da se sustav ponavlja gledajući da je „looping“ postavka uključena. Postoje varirajuće veličine pojedinačnih čestica te one imaju varirajuću rotaciju. Maksimalni broj čestica u jednom trenutku je 3500. Početna boja (eng. start color) je namještena na crnu, što predstavlja radioaktivni snijeg/pepeo koji nastane nakon nuklearne eksplozije.

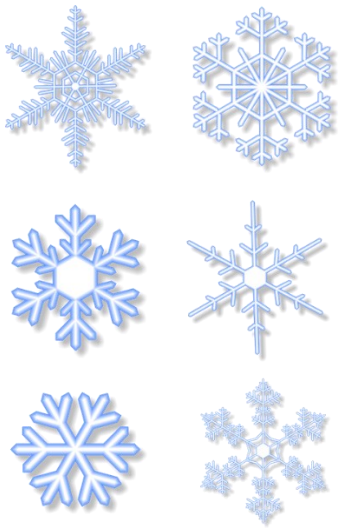
U „emmision“ dijelu postavki vidi se da je rate over time jednak 1000. To znači da je u svakoj sekundi izbačeno 1000 čestica. U „shape“ dijelu postavki vidi se da je oblik izbacivača čestica jednak kutiji (eng. box) i da su čestice izbačene iz zapremine te kutije.

Brzina čestica kroz vrijeme je varirajuća na svim osima. „Color over lifetime“ govori koje će boje biti čestica kroz vrijeme. Vidi se da se pri kraju tog životnog vremena (eng. lifetime) alfa vrijednost postepeno smanjuje na nulu. To znači da kad snijeg padne na pod polako nestane kroz nekoliko sekundi.

Kod kolizije se vidi da svaka čestica ima uključenu koliziju. Polupreč nik kolizijske kugle oko čestice je namješten na 0.5f. Kolizija se koristi da bi bilo detektirano kad pahuljica dotakne igrača i onda se taj događaj koristi da bi glavni lik bio oštećen. Idući dio postavki, „triggers“ je korišten za postavke okidača kojeg pogađaju pahuljice.

Vidi se da je dodijeljen GameObject pod imenom „SnowTrigger“ kao okidač. To je okidač koji se nalazi oko glavnog lika i predstavlja njegovo tijelo ili točnije, njegovo tijelo u aspektu sudaranja s pahuljicama.

Texture sheet animation je modul koji je dio sustava čestica i pristupa se kroz ParticleSystem klasu. Ovaj modul omogućava da se teksture tretiraju kao rešetka (eng. grid) odvojenih pod-slika koje mogu biti puštene jedna za drugom kao animacija. Ovo omogućava da se za više sličica za animaciju koristi samo jednu datoteka što dosta pomaže pri smanjivanju veličine igrice. Sliku od koje su izvađene sličice (eng. sprites) su podijeljene na 6 dijelova, dvije kolone i tri reda. U nastavku će biti prikazane slike pahuljica koje su slike animacija teksture te slika crnog snijega. Nakon toga je prikazan isječak koda unutar BlackSnowController.cs skripte.



*Slika 15 Slika Pahuljica za Texture Sheet Animation* *Slika 16 Black Snow*

```
public Transform Player;

 // Update is called once per frame
 void Update () {
     if (!Player)
         return;

     gameObject.transform.position = Player.transform.position;
 }
```

*Isječak Koda 29 Skripta BlackSnowController.cs 1. dio*

U BlackSnowController.cs skripti postoji javna varijablu Transform tipa. Ta varijabla je referenca na Player GameObject što je glavni lik (spirit). U Update() funkciji se na svakom frame-u provjerava da li postoji glavni igrač. Glavni igrač ne bi postojao u slučaju da je taj GameObject uništen. Ako ne postoji, onda se izlazi iz Update() funkcije. Svaki frame, pozicija od objekta na kojem je skripta se mijenja, a to je držač ili roditeljski objekt od sustava čestica crnog snijega.

Mijenja se pozicija tako da se napravi da je jednaka poziciji glavnog igrača. To znači da izbacivač (eng. emitter) za čestice snijega prati igrača. Ovo se na ovaj način radi zbog optimizacije jer nema smisla da cijeli svijet bude pokriven snijegom kad 90% toga igrač ne bi vidio u danom trenutku. Kutija koja izbacuje čestice je manja ali prati igrača tako da se ne primijeti iz perspektive igrača. U nastavku se nalazi isječak koda iz SnowDamage.cs skripte.

```
    public new ParticleSystem particleSystem;
    public float damage = 2f;
    public GameObject Player;
    public CameraShake cameraShake;

    void OnParticleTrigger()
    {
    List<ParticleSystem.Particle> enter = new List<ParticleSystem.Particle>();
    int numEnter =
particleSystem.GetTriggerParticles(ParticleSystemTriggerEventType.Enter, enter);

    for (int i = 0; i < numEnter; i++)
    {
        float random = Random.value;
        if(random < .3)
        AudioManager.Instance.Play(SoundType.BlackSnowBurn);
        else if(random < .7)
        AudioManager.Instance.Play(SoundType.BlackSnowBurn2);
        else
        AudioManager.Instance.Play(SoundType.BlackSnowBurn3);
        StartCoroutine(cameraShake.Shake(.25f, .6f, true));

        Player.GetComponent<PlayerHealth>().TakeDamage(damage);
    }
    }
```

*Isječak Koda 30 Skripta SnowDamage.cs 1. dio*

SnowDamage.cs skripta se koristi da bi skidanje života igraču bilo regulirano kad dotakne pahuljicu. Prvo je definirano nekoliko varijabli, jedna od njih je tipa ParticleSystem koja referencira BlackSnow sustav čestica.

U OnParticleTrigger() funkciji, koja se poziva kad bilo koja čestica iz sustava čestica postigne uvjete u modulu okidača, dakle kad čestica pogodi igrača, prvo je definirana lista čestica. Onda je korištena GetTriggerParticles() funkcija koja vraća broj čestica napisane u prije spomenutoj listi, dakle broj čestica koje su pogodile igrača. Nakon toga, pokreće se for petlja koja iterira kroz svaku instancu pahuljice. Za svaku pahuljicu generirana je nasumična vrijednost između 0 i 1 i onda na osnovu rezultata pušta se jedan od tri zvuka „prženja“ pahuljice. Nakon tog je pozvana Shake() funkcija da bi ekran bio protresen kad lika udari pahuljica. Zadnja stvar što je urađena je da je pozvana TakeDamage() funkcija koja skida određenu količinu života s glavnog lika.

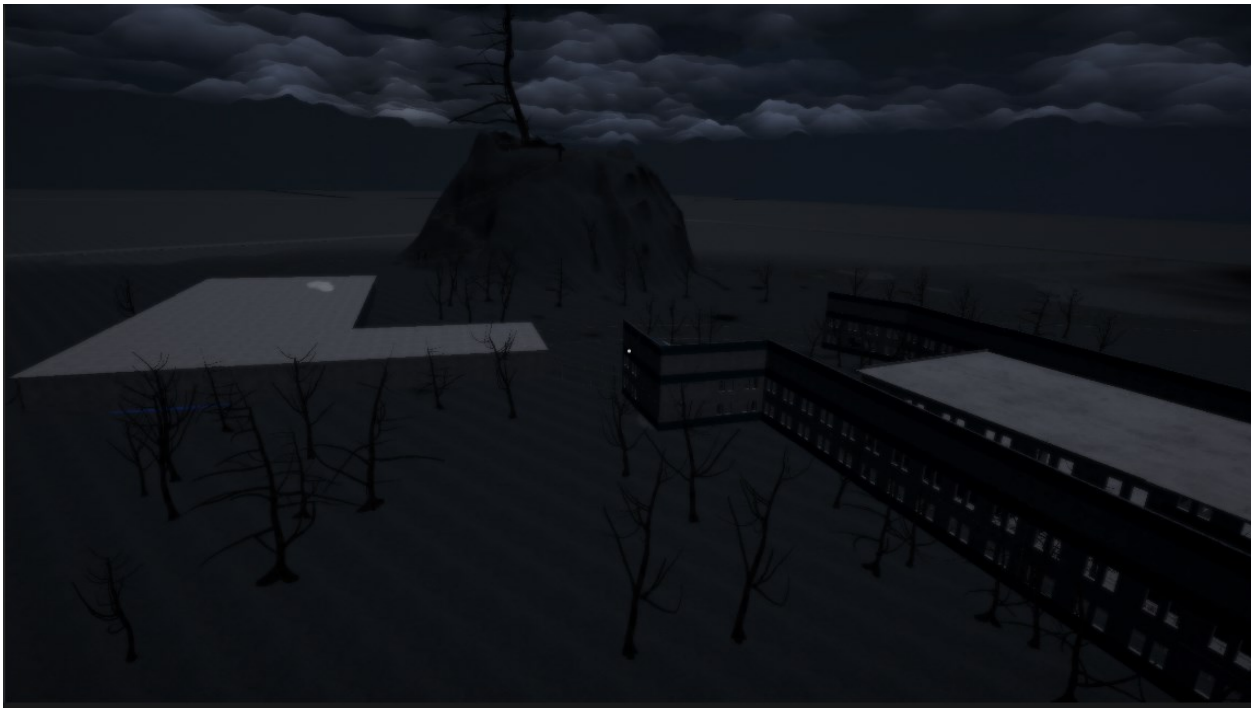
## 5.4 Okruženje

Okruženje u Kuroi Yuki obuhvaća cijeli krajolik i sve objekte koje se nalaze na njemu. Okruženje je ciljano napravljeno u tmurnom stilu da bi pojačalo osjećaj straha i koristi razne elemente da bi pojačalo taj ambijent. Po svijetu se nalaze razni artefakti koje imaju svoje simbolično značenje. Svijet je isto tako popunjen mrtvim tijelima što označava da se glavni lik nalazi u sasvim razrušenom i paklenom okruženju. U pozadini se nalazi velika nuklearna eksplozija koja je u „zaustavljenom vremenu“. Veliko tamno drvo se nalazi na brdu koje gleda na svijet oko sebe.

Vidi se zatvoreni prostor u kojem je „jezero krvi“. Oblaci se pomjeraju iznad igrača te povremeno zaustavljaju mjesečinu. Svi ovi elementi će biti opisani u ovoj sekciji te će biti objašnjeno kako su napravljeni, kako funkcioniraju i kako utječu na ambijent i atmosferu igrice Kuroi Yuki. Velik dio održavanja straha i neizvjesnosti u igricama je održavanje ravnoteže između predvidljivosti i nepredvidivosti (Cammarata, 2016). Igrica obrađena u ovom radu ovaj učinak izvršava kroz različite elemente unutar okruženja. U nastavku se nalazi slika koja prikazuje okruženje Kuroi Yuki.

*Slika 17 Okruženje Igrice Kuroi Yuki*

Alternativni svijet koji vidimo kroz Mind's Eye sposobnost, je u puno boljem stanju od glavnog okruženja. Može se zaključiti da ovaj svijet predstavlja stanje prije eksplozije i tragičnih događaja. Ovdje umjesto jezera krvi postoji obični bazen. Eksplozija nije prisutna. Ispod se nalazi slika koja prikazuje okruženje Kuroi Yuki alternativnog svijeta.



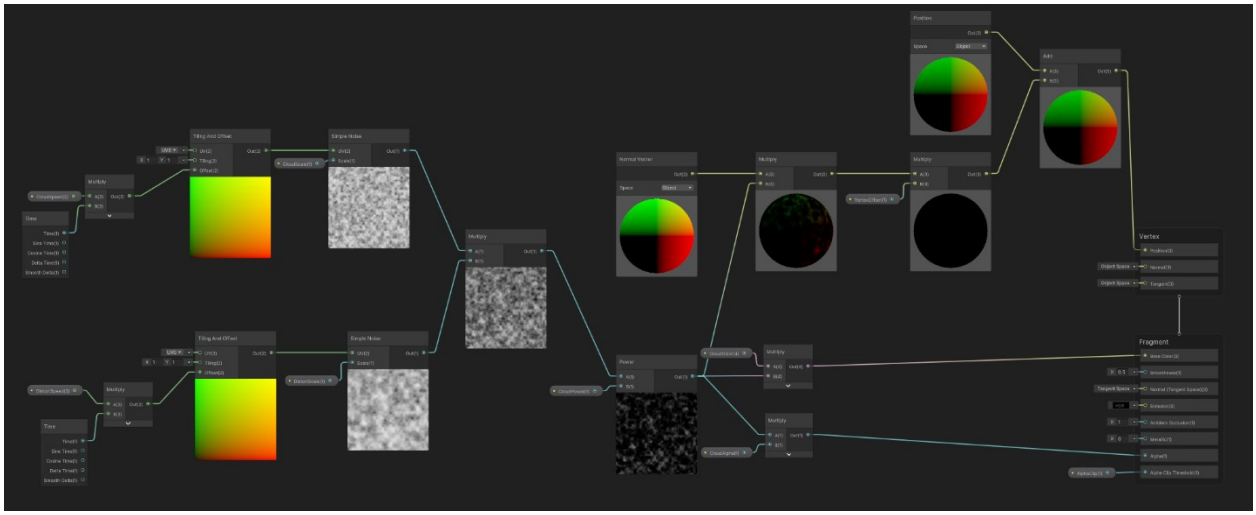
*Slika 18 Okruženje Alternativnog Svijeta Igrice Kuroi Yuki*

Unity graf shader-a (eng. Shader Graph) je korišten više puta u Kuroi Yuki. Korišten je za pravljenje shader-a (eng. Shaders) za oblake, nuklearnu eksploziju i za jezero krvi i bazen. Shaderi su mali programi koji su pokrenuti na grafičkoj kartici, oni rade stvari kao mapiranje tekstura (eng. texture mapping), osvjetljenje ili bojenje objekata. Obično su shaderi postojali samo u kodu.

Graf shader-a mijenja to i dovodi način da se prave shadere bez koda, to je dosta učinkovitije i pristupačnije. Shader graph omogućava da se koriste čvorovi (eng. nodes) koji predstavljaju različite funkcionalnosti koje korisnici mogu spojiti s drugim čvorovima na interaktivnom grafu (Lilett, 2021).

### 5.4.1 Oblaci

Oblaci u igrici obrađenoj u ovom radu su napravljeni koristeći prije spomenuti graf shader-a. Oblaci dodaju dosta osobnosti igrici te čine da se nebo ne čini prazno. Prvo će biti prikazan izgled „Clouds“ grafa shader-a a nakon toga opisani ključni elementi.



*Slika 19 Cloud Shader Graph*

Postoje dvije početne grane koje se množe kako bi se stvorili oblaci s distorzijom. Gornja grana predstavlja oblake s šumom (eng. noise) a druga distorziju koja je pomnožena s prvobitnim oblacima, što daje granulirane oblake. Početni, čvor je „time“ koji je pomnožen s CloudSpeed varijablom. Nakon toga postoji TilingAndOffset čvor čiji izlaz nastavlja u SimpleNoise čvor.

Time čvor daje pristup više float-ova, svi koji se mijenjaju kroz vrijeme. Izlaz ovog čvora je vrijeme u sekundama od kad je scena počela. Multiply čvor uzme 2 ulaza i pomnoži ih. TilingAndOffset se koristi da se pomakne (eng. to offset) i podijeli UV mapa na ploče (eng. tiles).

Ulaz UV0 je set UV-ova na koje je ovaj efekt primijenjen. Postoje Tiling i Offset ulazi koji primaju Vector2 vrijednosti. Tiling ulaz kontrolira koliko puta se tekstura kopira na objektu a Offset ulaz se može koristiti da se tekstura povlači po objektu. UV mape su dvodimenzionalne koordinate s kojima je tekstura mapirana na trodimenzionalni model.

Slova U i V se koriste jer X,Y i Z su već korišteni da se referenciraju osi od objekata u trodimenzionalnom prostoru (Lilett, 2021).

Simple Noise čvor generira šumne šablone (eng. noise patterns) nazvane „value noise“ koristeći UV kao ulaz da se mapira šum na teksturu i Scale ulaz da se promijeni veličina teksture u oba smjera. Izlaz je float između 0 i 1. Dalje se vidi da postoji Power čvor koji se množi s javnim floatom i njegov izlaz se množi s bojom i alfa vrijednosti oblaka. Power čvor uzima dva floata i vraća kao izlaz prvi ulaz potenciran s drugim. Ti izlazi se šalju u Base Color i Alpha fragment kontrolere koji direktno djeluju na shader.

S druge strane se Power izlaz šalje pomnožen s floatom u čvor množenja te se množi s NormalVector čvorom. Normal Vector čvor uzima vektor okomit na površinu te dopušta da se biraju različiti prostori (eng. spaces) kao world ili object space i onda vraća samo jedan izabrani vektor. Umnožak Powera i NormalVectora se množi s VectorOffset floatom i zbraja s Position čvorom. Izlaz se šalje u Position opciju u Vertex kontroleru.

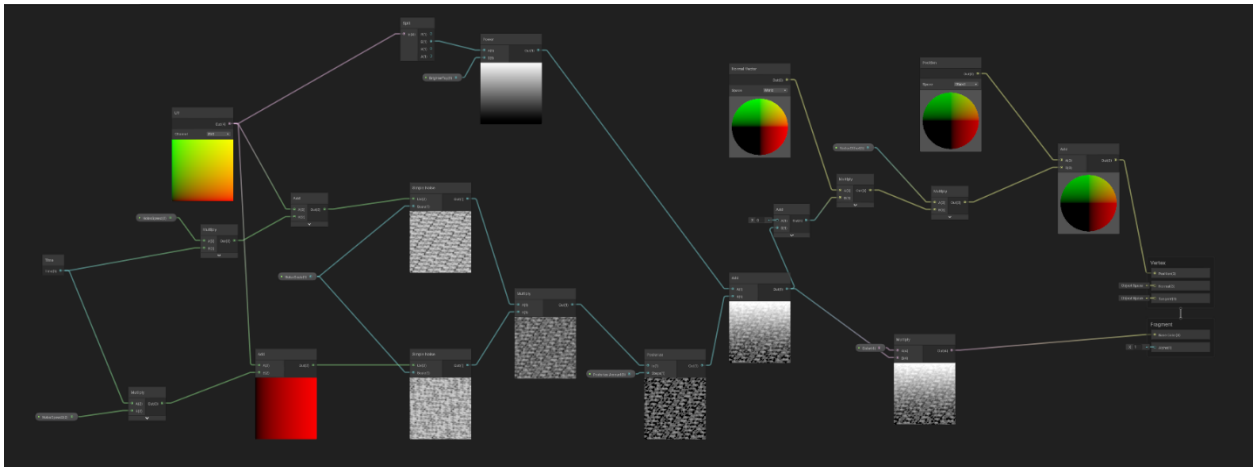
Manipulacija vrhova omogućava da se modificira geometrija iscrtane slike (ÇAMÖNÜ, 2020).

Konačni shader se stavlja u materijal koji se stavlja na mesh napravljen u Blenderu. Mesh je kolekcija lica, kutova, i vrhova koji čine trodimenzionalni oblik. Na primjer, mesh od kocke bi imao 6 lica 8 vrhova i 12 kutova (Fuentes, 2021). U 3D modeliranju, vrh (eng. vertex) je točka u trodimenzionalnom prostoru. Jedan vrh ima tri koordinate, vrijednost za svaku od X, Y i Z osi (Mobsby, 2021). Na slici ispod su prikazani opisani oblaci.

*Slika 20 Oblaci u Kuroi Yuki*

### 5.4.2 Nuklearna Eksplozija

Idući shader je Nuke shader koji je korišten za nuklearnu eksploziju. Na slici ispod će biti prikazan graf shader-a te će biti opisani čvorovi koji nisu bili opisani u prijašnjem shader-u.



*Slika 21 Nuke Shader Graph*

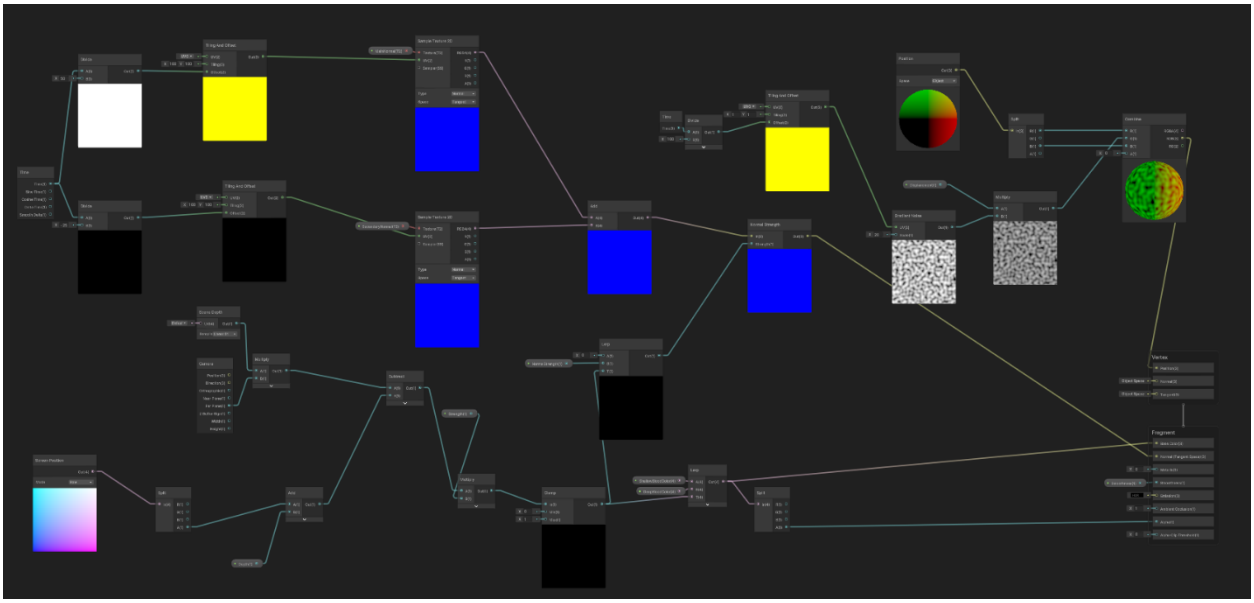
UV čvor se koristi da se dobiju UV koordinate vrha ili fragmenta. Posterize čvor uzima ulaznu vrijednost i korak (eng. step) vrijednost. Čvor će ograničiti (eng. clamp) raspon inputa između 0 i 1 i pretvoriti vrijednost u broj tako da može jedino primiti vrijednosti jednake broju koraka koji su definirani plus jedan. S ovim čvorom se postiže efekt granuliranja i pravljenja „oštrijih“ granica na teksturi. Na slici ispod je prikazana eksplozija.

*Slika 22 Nuklearna Eksplozija u Kuroi Yuki*

Shader se primjenjuje na materijal te onda materijal na mesh od gljive. Gljiva ima dosta sličan oblik nuklearnom oblaku te se zbog toga takav oblak zove „Mushroom Cloud“(hrv. gljivasti oblak). Gljiva se povećava do veličine eksplozije.

### 5.4.3 Jezero Krvi

Blood lake ili jezero krvi je tijelo krvi koje se nalazi u tunelu na mapi. Koriste se shaderi kako bi se dobio izgled krvi. U nastavku će biti prikazan izgled ovog grafa shader-a i nakon toga će biti opisani novi pojmovi.



*Slika 23 Blood Lake Shader Graph*

Čvor Divide, uzima dva float ulaza i vraća njihov količnik. Screen Position je čvor koji dobiva poziciju piksela na ekranu, jedan Vector4 izlaz predstavlja poziciju ekrana. Scene Depth je čvor koji se koristi da se pristupi „depth buffer-u“ što je mjera daljine iscrtanog piksela od ekrana.

Čvor Camera je podržan jedino u URP (eng. Universal Render Pipeline) te dopušta pristup rasponu postavki povezanih s kamerom koja se trenutno koristi za iscrtavanja, kao naprimjer, pozicija u prostoru ili vektor prednjeg smjera.

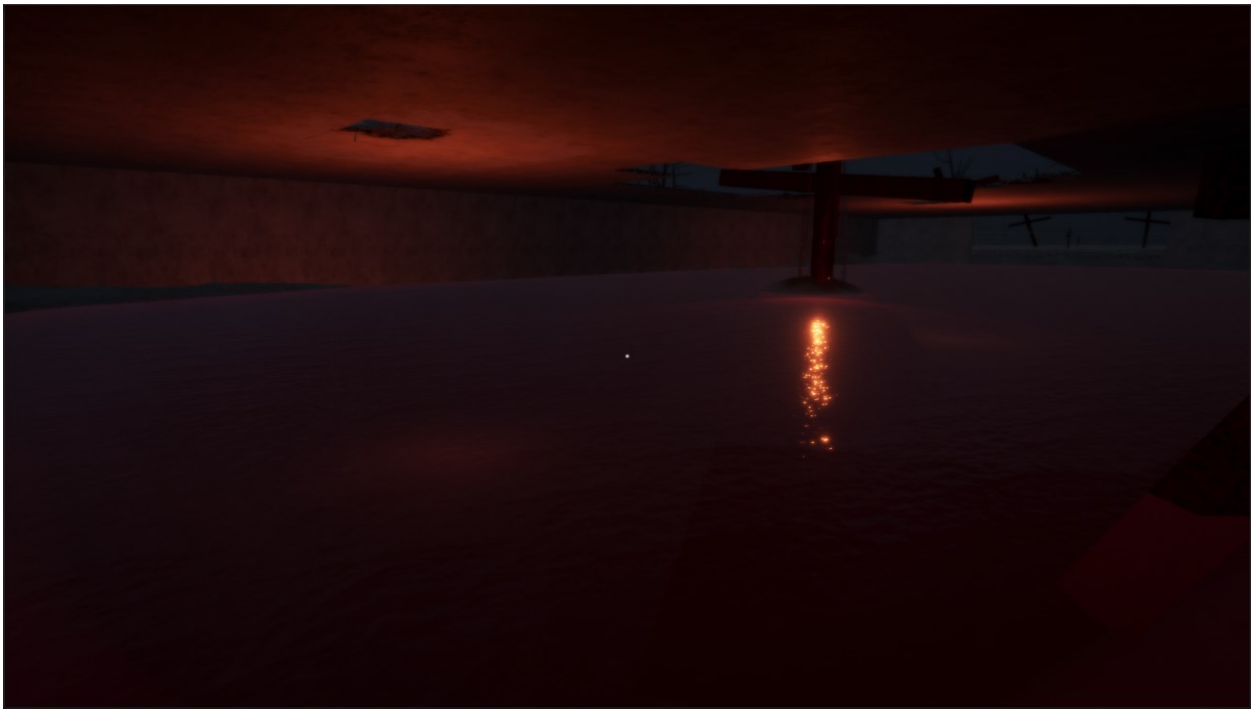
Sample Texture 2D čvor uzima tri ulaza. Teksturu koju uzorkuje (eng. Sampling), UV koordinate i stanje uzorkovanja (eng. Sample State). Izlaz je ili RGBA boja teksture ili bilo koja od RGBA pojedinačnih komponenti (red, green, blue, alpha) (Lilett, 2021). Gradient

Noise generira malo više kompleksan i sofisticiran šum pod nazivom Perlin Noise koristeći iste ulaze kao SimpleNoise čvor i kao izlaz ima jedan floating point broj. Subtract uzima 2 inputa i oduzima A od B, vraća tu razliku kao izlaz. Split uzima Vector4 kao ulaz i kao izlaz ima četiri kanala od vektora kao različiti float brojevi.

Ako se stavi vektor s manje od 4 komponente kao ulaz, onda će viška komponente biti 0. Combine omogućava da se ubace 4 vrijednosti u R, G, B, A ulaze i čvor će kombinirati ove individualne elemente u vektore. Lerp čvor je skraćenica za „Linear Interpolation“.

Ovaj čvor će uzeti dva ulaza, A i B koji mogu biti vektori do 4 komponente. Interpolacija crta ravnu liniju između A i B i vraća jednu float vrijednost. Ta float vrijednost je korištena za svaku od komponenata od A i B. Izlaz je vrijednost koja je odabrana. Normal Strength je čvor koji uzima set inputa kao Vector3 i skalira njihove jačine kroz Strength Float Input.

Jačina od 1 ostavlja ulaz nepromijenjen dok 0 vraća kompletno ravnu mapu sa svim normalima usmjerenim prema gore (Lilett, 2021). Na slici ispod je prikazano jezero krvi.



*Slika 24 Blood Lake*

#### 5.4.3.1 Sustav Gušenja

Igrač ne može zauvijek biti unutar jezera krvi. Postoji sustav gušenja koji kad dođe do 0, lik umire i pokreće se prikladna smrtna scena. Ovaj sustav i mehanika stvara velik osjećaj straha. Strah je racionalan osjećaj koji je pokrenut s nekom neizvjesnom prijetnjom gdje se žrtva osjeća kao da su njihovi trenutni resursi nedovoljni za suočavanje s prijetnjom (Nielsen, 2013). U ovom slučaju taj resurs je preostali kisik/život. Sustav gušenja u jezeru krvi napravljen je na sličan način na koji je napravljen cooldown sustav za Mind's Eye sposobnost. U isječku koda ispod je prikazana Start() funkcija DrowningSystem.cs skripte.

```
    public Slider breathBar;
    public int maxBreath = 150;
    public int currentBreath;
    private WaitForSeconds regenTick = new WaitForSeconds(0.1f);
    public GameObject breathBarGameObject;
    public MindsEyeController mindsEyeController;
    void Start(){
        currentBreath = maxBreath;
        breathBar.maxValue = maxBreath;
        breathBar.value = maxBreath;
    }
```

*Isječak Koda 31 Skripta DrowningSystem.cs 1. dio.*

Definiraju se varijable, jedna od njih referencira slajder koji prikazuje koliko je daha ostalo dok je lik pod jezerom krvi. U Start() funkciji jednači se trenutni dah s maksimalnim dahom i te vrijednosti se prenose na UI Slider. U nastavku je prikazana i opisana HoldBreath() korutina.

```
    public IEnumerator HoldBreath(int amount){
        breathBarGameObject.SetActive(true);
        while(currentBreath>0){
        if(!mindsEyeController.itemActive){
            currentBreath -= amount;
            breathBar.value = currentBreath;
        }
        yield return regenTick;
        }
        SceneManager.LoadScene("Death Scene Drown");
    }
```

*Isječak Koda 32 Skripta DrowningSystem.cs 2. dio.*

Definira se HoldBreath() korutina u kojoj je na početku aktiviran slajder koji prikazuje koliko je igraču daha ostalo. Nakon toga, sve dok je trenutni dah veći od 0, i ako igrač ne koristi Mind'sEye sposobnost, onda se u svakoj iteraciju oduzima količina od trenutnog daha te se dodijeli ta vrijednost na slajder. Ako trenutni dah dođe do nule, onda glavni lik umire i učitava se prikladnu scena smrti. Dok lik koristi Mind's Eye sposobnost, potrebno je da se zaustavi oduzimanje daha, to dodaje dodatni element interaktivnosti ove sposobnosti. U isječku koda ispod je prikazana BreatheIn() korutina.

```
public IEnumerator BreatheIn()
    {
        breathBarGameObject.SetActive(false);
        while(currentBreath<maxBreath)
        {
        currentBreath += maxBreath / 50;
        breathBar.value = currentBreath;
        yield return regenTick;
        }
    }
```

*Isječak Koda 33 Skripta DrowningSystem.cs 3. dio.*

U BreatheIn() korutini je urađena obrnuta stvar. Sakriven je slajder i onda, sve dok je trenutni dah veći od maksimalnog daha, u svakoj iteraciji je trenutnom dahu dodano 2% vrijednosti maksimalnog daha. Ista ta vrijednost je preslikana na slajder. Obje korutine su menadžirane unutar PlayerCollision.cs skripte u funkcijama koje se pokreću kad igrač ulazi i izlazi iz jezera krvi, ove funkcije su prikazane u isječku koda ispod.

```
if(collision != null && collision.gameObject.CompareTag("GetInWater")){
            . . .
             if(breatheInCoroutine != null)
             StopCoroutine(breatheInCoroutine);
             holdBreathCoroutine = StartCoroutine(drowningSystem.HoldBreath(1));
         }
if(collision != null && collision.gameObject.CompareTag("GetOutOfWater")){
            . . .
             if(firstTimeInWater){
                StopCoroutine(holdBreathCoroutine);
                if(breatheInCoroutine != null)
                StopCoroutine(breatheInCoroutine);
                breatheInCoroutine = StartCoroutine(drowningSystem.BreatheIn());
             } }
```

*Isječak Koda 34 Skripta DrowningSystem.cs 4. dio.*

### 5.4.4 The Smoldering Gazer

The Smoldering Gazer je glavni neprijatelj u Kuroi Yuki. To je veliko izgoreno stablo koje ima veliko oko na dnu. Glavni način na koji ovaj neprijatelj napada igrača je da ga tjera da ga pogleda u oko tako što mu vuče kameru na pulsirajući način prema oku. Smoldering Gazer ima domet koji kad igrač uđe u njega, pokreće neprijatelja.

Ako igrač pogleda u oko od Smoldering Gazera, igrač umire i pušta se prikladna scena smrti. Igrač je veoma usporen kad je u dometu ovog neprijatelja i treba otići iza njega da bi došao po ključ (chaos cube) koji mu pomaže da napreduje u igrici.

U slici 25 će biti prikazan izgled ovog neprijatelja te će ispod, u isječku koda, biti prikazan način na koji ovaj neprijatelj funkcionira.

*Slika 25 The Smoldering Gazer*

```
// when you enter the range of the gazer
   if(collision != null && collision.gameObject.CompareTag("RaycastLongRange"))
          {
              raycastScriptGameObject.RaycastLongRange();
              if(!smolderingGazerActive)
              {
              Debug.Log("Entered range!");
```

```
                eyeMaterial.SetColor("_EmissionColor", eyeColorActive);
                eyeLight.SetActive(true);
playerMovementScriptObject.slowDownMovement(slowedDownWalkingSpeed,slowedDownSpri
ntSpeed);
                audioMixer.SetFloat("JumpscaresVolume", -2f);
                AudioManager.Instance.Play(SoundType.EyeWakeUp);
                smolderingGazerActive = true;
                if(smolderingGazerCoroutine != null)
                {
                    StopCoroutine(smolderingGazerCoroutine);
                }
                smolderingGazerCoroutine = StartCoroutine(EyePullPulse());
```

*Isječak Koda 35 Skripta PlayerCollision.cs, dio za SmolderingGazer-a 1. dio.*

Kad igrač uđe u doseg SmolderingGazer-a poziva se funkcija koja povećava domet Raycast-a i ako neprijatelj nije aktivan, može se aktivirati Gazer.

Namještena je boja oka od neprijatelja na svjetliju i aktivirano mu je svjetlo koje je pozicionirano na oku. Poziva se funkciju koja uspori pokrete igrača tako da oteža prolazak pored neprijatelja i poveća intenzitet bitke protiv neprijatelja. Pokrenuti su prikladni zvukovi i ako smolderingGazerCoroutine varijabla nije null, onda je zaustavljena. Nakon toga je odmah pokrenuta EyePullPulse() korutina koja je prikazana u isječku koda ispod.

```
IEnumerator EyePullPulse()
{
     while (true)
     {
// random time between pulses (clamped)
float randomTimeAmplifier = Random.Range(0.6f, 1.2f);
float timeBetweenEyePulsesAmplified = timeBetweenEyePulses * randomTimeAmplifier;

                 yield return new WaitForSeconds(timeBetweenEyePulsesAmplified);
                  float random = Random.value;
                  if(random < .5)
                  AudioManager.Instance.Play(SoundType.EyeLook1);
                  else
                  AudioManager.Instance.Play(SoundType.EyeLook2);
                 StartCoroutine(cameraShake.Shake(.65f, .5f, true));
               lookAtTargetScriptGameObject.StartRotating("eye", 0.27f, 0.65f);
                       yield return null;
     }}}}
```

*Isječak Koda 36 Skipta PlayerCollision.cs, dio za SmolderingGazer-a 2. dio.*

Na početku korutine postoji while petlja koja traje dok se korutina ne zaustavi. U svakoj iteraciji se generira nasumični broj koji će djelovati kao pojačivač moći neprijatelja. Nakon toga se čeka nekoliko sekundi i onda se počne puštati jedan od dva zvuka kojeg proizvodi neprijatelj kad napada.

Započinje protresanje ekrana i počne rotacija igrača prema oku. Sve komande zajedno čine jedan napad od Smoldering Gazer-a. U nastavku je prikazan način smanjenja dometa neprijatelja.

```
// when you exit the range of the gazer
if(collision != null && collision.gameObject.CompareTag("RaycastShortRange"))
            {
                raycastScriptGameObject.RaycastShortRange();
                 if(smolderingGazerActive)
                {
                    Debug.Log("Exited range!");

                    eyeMaterial.SetColor("_EmissionColor", eyeColorInactive);
                    eyeLight.SetActive(false);

playerMovementScriptObject.speedUpMovement(spedUpWalkingSpeed,spedUpSprintSpeed);

                    audioMixer.SetFloat("JumpscaresVolume", 11f);
                    AudioManager.Instance.Play(SoundType.EyeFallAsleep);

                    smolderingGazerActive = false;
                    StopCoroutine(smolderingGazerCoroutine);
            }}}
```

*Isječak Koda 37 Skipta PlayerCollision.cs, dio za SmolderingGazer-a 3. dio.*

Nakon što igrač izađe iz dometa neprijatelja, smanjen mu je domet Raycast-a te je isključena svjetlost i promijenjena boja kod neprijateljskog oka. Ubrzani su pokreti i pokrenuti prikladni zvukovi. Korutina pulsirajućeg napada je zaustavljena.

### 5.4.5 The Propagator

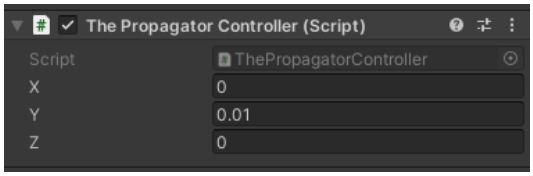
The propagator je jedan od pasivnih neprijatelja u Kuroi Yuki. To je najveći artefakt u okruženju i jedan je od zadnjih objekata do kojeg igrač može doći. Igrač kod prijašnjeg neprijatelja „Smoldering Gazer“ uzima objekt koji se zove „Chaos Cube“, idući cilj je da odnese taj objekt do Propagatora. Nakon što odnese i postavi kocku na određeno postolje, pokreće se zadnji događaj pod imenom „The Final Event“. Ovaj događaj će biti opisan u zadnjoj sekciji u radu. Na slici Ispod je prikazan Propagator artefakt te je nakon toga, prikazana Update() funkcija u ThePropagatorController.cs skripti.

*Slika 26 The Propagator Kuroi Yuki*

```
public float x,y,z;
    void Update()
    {
        transform.Rotate(x, y, z);
    }
```

*Isječak Koda 38 Skripta ThePropagatorController.cs 1. dio.*

U skripti ThePropagatorController je kontrolirana rotacija propagatora. Dodani su argumenti u transform.Rotate() s vrijednostima određenim u inspektoru. U slici ispod su prikazane javne varijable definirane u isječku koda iznad.




*Slika 27 The Propagator Controller Javne Varijable*

## 5.5 Unity UI

Unity UI (user interface) je skupina alata za razvijanje sučelja za igrice i aplikacije u Unity-u. To je UI sustav baziran na GameObject-ima koji koristi komponente da rasporedi, pozicionira i stilizira korisničko sučelje. U nastavku ove sekcije će biti opisane instance korištenja UI elemenata u igrici opisanoj u ovom projektu, neke od tih instanci su Main Menu, Pause Menu, Note System, itd.

### 5.5.1 Sustav Bilješki

Bilješke (eng. notes) su papiri na kojima su napisani monolozi od NPC-ova (non playable characters) u igrici koji propagiraju priču i uvode psihološki horor aspekt u Kuroi Yuki. Priča u igricama je aparat koji se koristi da potpuno uključi igrača u igricu koristeći bogat zaplet koji tjera igrača da pobijedi sve prepreke da bi iskusio zadovoljavajuće rješenje (Griffin, 2019).

Ovo se u Kuroi Yuki najviše postiže sustavom bilješki. Bilješke su popraćene glasovnom glumom i prepadima koji su razvijeni u Shake() funkciji. U ovom slučaju se koristi DelayedShake() funkcija koja funkcionira na isti način kao Shake() samo s periodom vremena koji prođe prije prepada. Ispod je prikazan prvi dio NoteSystem.cs skripte.

```
    bool isPaused = false;
    AudioSource noteAudio;
    public DelayedCameraShake delayedCameraShake;
    float firstDelay;
    float secondDelay;
    private Coroutine delayedShakeCoroutine1;
    private Coroutine delayedShakeCoroutine2;
    public AudioMixer audioMixer;
    public void showNote(GameObject note){
        GameObject noteUI = note.transform.GetChild(0).gameObject;
        noteUI.SetActive(true);
        AudioManager.Instance.Play(SoundType.NoteOpen);
        noteAudio = note.GetComponent<AudioSource>();
        noteAudio.Play();
        audioMixer.SetFloat("SoundEffectsVolume", -20f);
```

*Isječak Koda 39 Skripta NoteSystem.cs 1. dio*

U NoteSystem.cs skripti su definirane varijable. Jedna od njih je varijabla audioMixer tipa AudioMixer. AudioMixer će biti opisan u Menadžeri sekciji. Definirana je showNote()

funkcija koja pokreće postupak prikazivanja bilješke u koju igrač gleda na ekran, prisutna je i hideNote() funkcija ali neće biti opisana jer funkcionira na veoma sličan način kao showNote() samo s obrnutim vrijednostima. Radi redundancije će biti izostavljena. Na početku funkcije dohvaćeno je UI platno od bilješke i aktivirano.

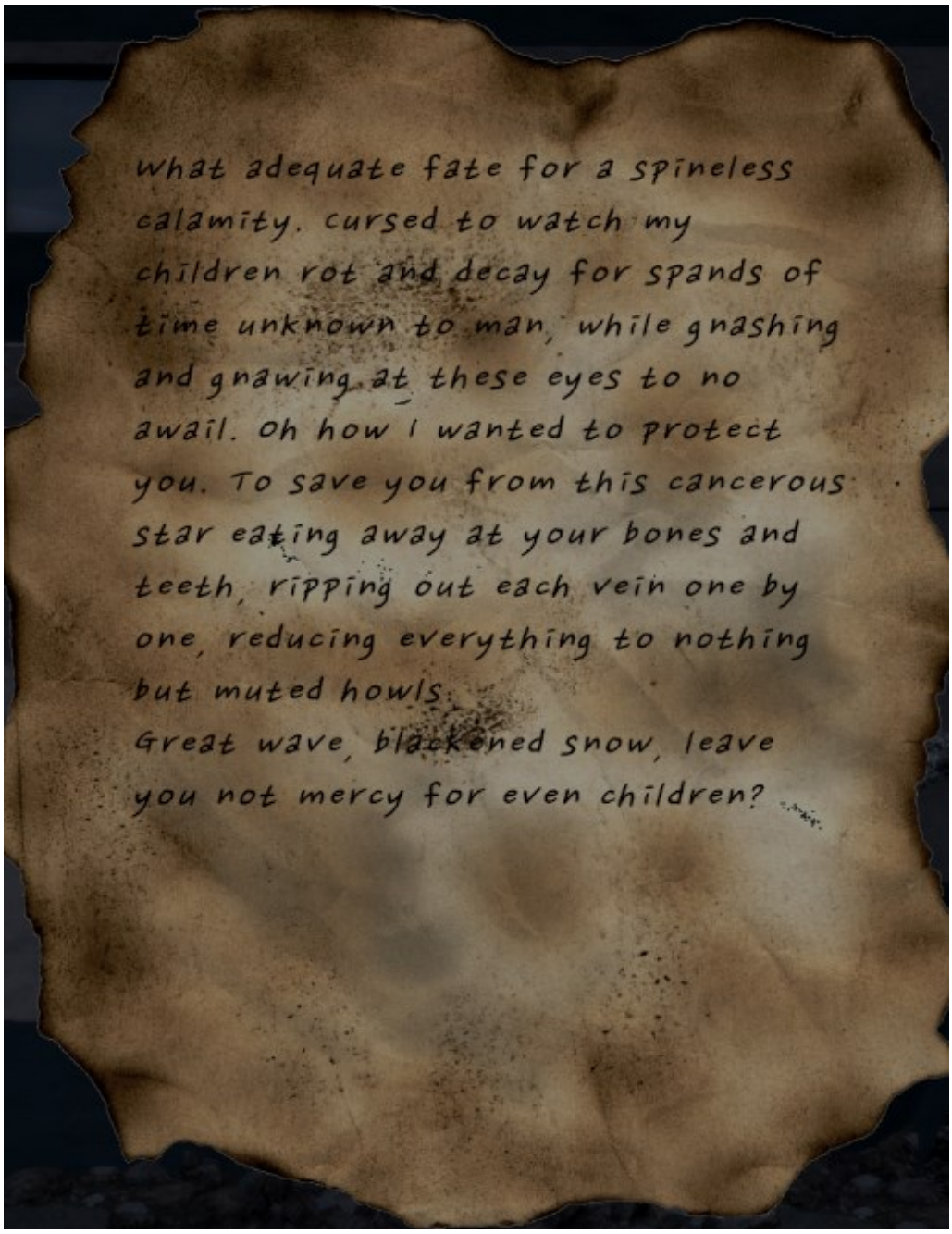
Pokrenut je zvuk otvaranja bilješke i dohvaćen izvor zvuka(eng. AudioSource) za glasovnu glumu. Pokrenuta je glasovnu glumu te su utišani zvučni efekti da bi se bolje mogao čuti glas. U nastavku je prikazan drugi dio NoteSystem.cs skripte te nakon toga slika jedne bilješke.

```
        if(note.name == "Note1")
        (firstDelay, secondDelay) = (14.95f, 29.35f);

        else if(note.name == "Note2")
        (firstDelay, secondDelay) = (0f,0f);

        else if(note.name == "Note3")
        (firstDelay, secondDelay) = (6.656f,19.155f);

        delayedShakeCoroutine1 =
StartCoroutine(delayedCameraShake.DelayedShake(.30f, .5f, false, firstDelay));

        delayedShakeCoroutine2 =
StartCoroutine(delayedCameraShake.DelayedShake(.35f, .5f, false, secondDelay));

        Debug.Log("note shown");
        isPaused = true;
}
```

*Isječak Koda 40 Skripta NoteSystem.cs 2. dio*

Provjerava se koja je bilješka otvorena i u odnosu na to, pozvana je funkcija DelayedShake() te su joj kao argumenti dani vrijeme trajanja protresanja ekrana, jačina potresa, true ako je potrebno pustiti zvuk prepada dok se trese ekran i false ako nije i onda first delay za prvi prepad i second delay za drugi prepad u drugom pozivu funkcije. Na kraju je u konzolu ispisano da je bilješka prikazana i isPaused varijabla je true.

Na slici prikazanoj u nastavku se primijeti stil pisanja inspiriran od igrica Lethargy Hill i Anatomy. Kroz ovakav način pripovijedanja napreduje priča Kuroi Yuki.




*Slika 28 Note 1 u Kuroi Yuki*

U bilješkama se koriste grube i u nekim slučajevima odbojne deskripcije apstraktnih pojmova. Ostavljeno je na igračima da sami izvuku značenje ovih bilješki. Svaka deskripcija se može povezati sa životnom pričom neigrivih likova u Kuroi Yuki. Primijeti se da u igrici postoji jednak broj bilješki i mrtvih tijela razbacanih po mapi.

### 5.5.2 Glavni Meni

Kad igricu započne, prva scena koja je pokrenuta je Main Menu scena. U ovoj sceni se nalazi UI element koji predstavlja glavni meni u kojem se može ili započeti igrica ili ući u postavke gdje se može namjestiti glasnoća zvuka, kvaliteta i rezolucija igrice te ostale postavke. Treći gumb u glavnom meniju omogućava izlazak iz igrice. U nastavku, na slici je prikazan glavni meni, a ispod je isječak koda odgovoran za meni.


*Slika 29 Main Menu Kuroi Yuki*

```
public void PlayGame()
    {
        SceneManager.LoadScene("Main Scene");
        string path = Application.persistentDataPath + "/player.save";
        if(File.Exists(path))
        {
            GameManager.Instance.LoadData();
        }}
    public void QuitGame()
    {
        Debug.Log("Quit");
        Application.Quit();
    }
```

*Isječak Koda 41 Skripta MainMenu.cs 1. dio*

U MainMenu.cs skripti se nalaze dvije funkcije, PlayGame(), koja pokreće glavnu scenu i učitava spašene podatke. I QuitGame() koja izlazi iz igrice. Ispod se nalaze dva isječka koda iz SettingsMenu.cs skripte.

```
    Resolution[] resolutions;
    void Start(){
        resolutions = Screen.resolutions;
        resolutionDropdown.ClearOptions();
        List<string> options = new List<string>();
        int currentResolutionIndex = 0;
        for (int i = 0; i < resolutions.Length; i++){
            string option = resolutions[i].width + "x" + resolutions[i].height;
            options.Add(option);
            if(resolutions[i].width == Screen.currentResolution.width &&
resolutions[i].height == Screen.currentResolution.height){
                currentResolutionIndex = i;
            }
}
        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();
    }
```

*Isječak Koda 42 Skripta SettingsMenu.cs 1. dio*

U SettingsMenu.cs skripti se nalazi resolutions lista tipa Resolution. U Start() funkciji se prvo dodjeljuju sve dostupne rezolucije u ovu listu. Utječe se na UI „drop-down list“ tako što mu je dodijeljena jedna po jedna od mogućih rezolucija. U nastavku se nalaze četiri funkcije s kojima se dodjeljuju postavke u settings meniju te slika koja prikazuje ovaj meni.

```
    public void SetResolution(int resolutionIndex){
    Resolution resolution = resolutions[resolutionIndex];
    Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);
    }
    public void SetVolume(float volume){
        audioMixer.SetFloat("Volume", volume);
    }
    public void SetQuality(int qualityIndex){
        QualitySettings.SetQualityLevel(qualityIndex);
    }
    public void SetFullscreen(bool isFullscreen){
        Screen.fullScreen = isFullscreen; }
```

*Isječak Koda 43 Skripta SettingsMenu.cs 2.dio*


*Slika 30 Settings Menu Kuroi Yuki*

### 5.5.3 Meni Pauziranja

Dok je igrač u igrici, kad pritisne ESCAPE gumb na tipkovnici, zaustavlja se igrica te se prikazuje UI element pauzirane igrice prikazan na slici ispod. Ovdje igrač može kliknuti na tri gumba, prvi je da nastavi igricu, drugi da se vrati u glavni meni a treći je da izađe iz igrice.


*Slika 31 Pause Menu Kuroi Yuki*

U isječcima koda ispod, prikazana je PauseMenu.cs skripta.

```
void Update(){
var noteViewState = noteSystem.GetComponent<NoteSystem>().returnNoteViewState();
      if (Input.GetKeyDown(KeyCode.Escape) && !noteViewState){
          if (isPaused)
          {
              Resume();
          }
          else
          {
              Pause();
          }}}
```

*Isječak Koda 44 Skripta PauseMenu.cs 1. dio*

U Update() funkciji prvo se provjerava da li trenutno igrač gleda bilješku. Ako ne gleda i ako pritisne ESCAPE gumb na tipkovnici poziva se Pause() funkcija. Ako je igrica pauzirana, poziva se Resume() funkciju a ako nije pauzirana onda se poziva Pause() funkciju da bi bila pauzirana.

```
    void Pause(){
        Cursor.lockState = CursorLockMode.None;
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
        audioMixer.SetFloat("WalkingVolume", -80f);
        audioMixer.SetFloat("SoundEffectsVolume", -20f);
    }
```

*Isječak Koda 45 Skripta PauseMenu.cs 2. dio*

Pause() funkcija započinje tako što se otključava kursor. Nakon toga se aktivira GameObject od UI elementa Pause Menu-a. Vrijeme se zaustavi i utišaju se željene grupe u audioMixeru. Resume() funkcija je ista kao Pause() ali s obrnutim vrijednostima.

```
    public void LoadMenu(){
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
        audioMixer.SetFloat("WalkingVolume", 0f);
        audioMixer.SetFloat("SoundEffectsVolume", 0f);
        SceneManager.LoadScene("Main Menu");
        StopAllAudio();
    }
```

*Isječak Koda 46 Skripta PauseMenu.cs 3. dio*

U LoadMenu() funkciji radi se sve kao u Resume() funkciji i još se zaustave svi zvukovi i učita se „Main Menu“ scena. U isječku koda ispod je prikazana QuitGame() funkcija.

```
public void QuitGame()
    {
        Debug.Log("Quit!");
        Application.Quit();
    }
```

*Isječak Koda 47 Skripta PauseMenu.cs 4. dio*

U QuitGame() funkciji izlazi se iz igrice tako što se koristi Application.Quit() funkcija.

## 5.6 Naknadna Obrada

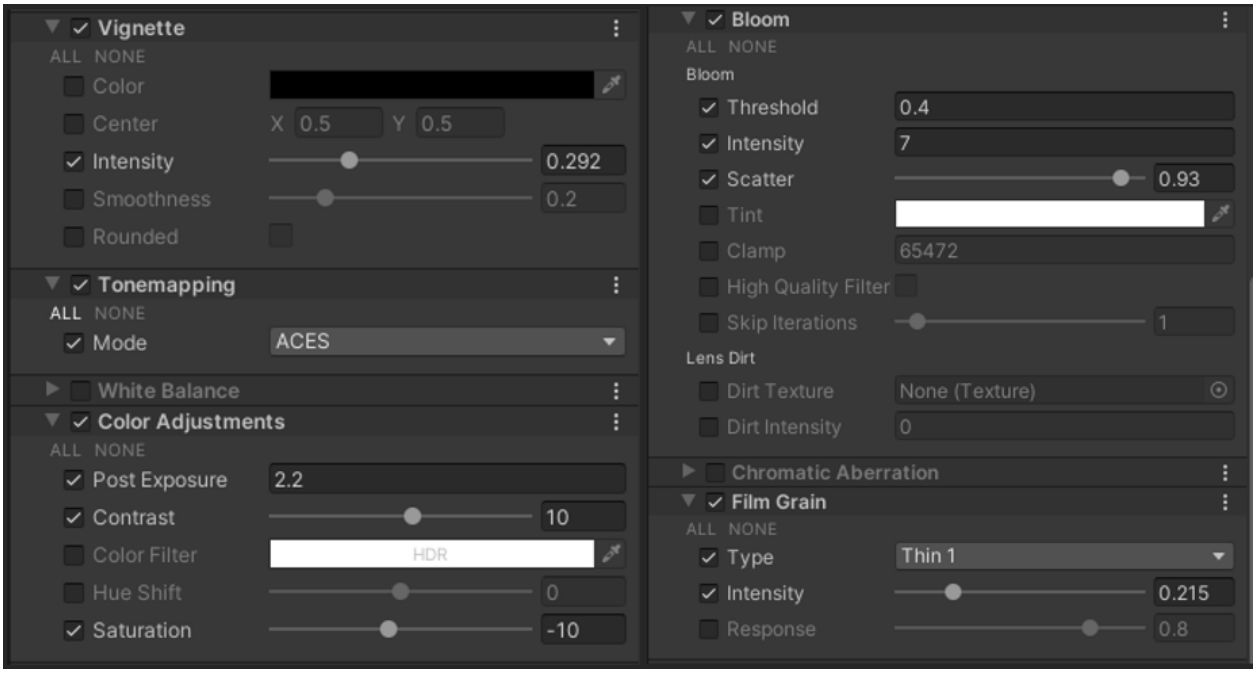
Naknadna obrada (eng. post processing) opisuje efekte slika koji se mogu aplicirati na scenu u igri. Mogu se primijeniti efekti naknadne obrade kao završni detalje na igrici. Ovi efekti mogu znatno poboljšati izgled igrice te naglasiti željeni ambijent (Oyedele, 2022).

Na primjer, u Kuroi Yuki, korišteni su ovakvi efekti da bi bila naglašena jeziva atmosfera i bio poboljšan općeniti izgled igrice. U nastavku će biti prikazana igrica Kuroi Yuki bez naknadne obrade i s njom, te će biti opisani korišteni efekti.

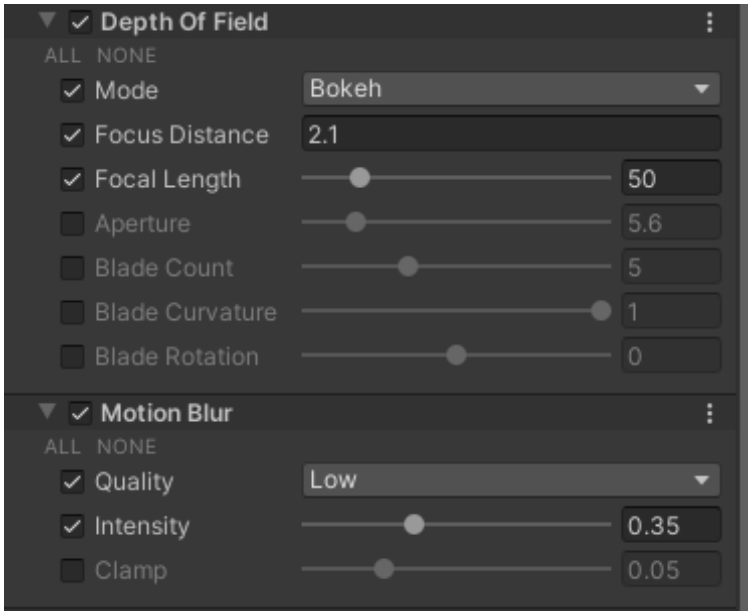
*Slika 32 Igrica Kuroi Yuki sa Post Processing*

*Slika 33 Igrica Kuroi Yuki bez Post Processing*

Vidi se da naknadna obrada znatno poboljšava ambijent u igrici i dodaje dubinu bojama/atmosferi. Ispod na slikama 34 i 35 se nalaze postavke za naknadnu obradu.




*Slika 34 Kuroi Yuki Post Processing 1*




*Slika 35 Kuroi Yuki Post Processing 2*

Vignette je efekt koji dodaje tamni okvir oko ekrana gdje je slika prikazana. Ovime se postiže odličan dramatični efekt te se veoma često koristi u horor igricama. Tonemapping remapira vrijednosti boja HDR (Hight Definition Range) i mijenja izgled na način da scena s manjim rasponom boja dobiva veći raspon boja. Ovaj efekt omogućava kontroliranje ekspozicije boje, kontrasta i zasićenosti boje. Isto tako se može utjecati na nijanse boja (eng. HUE) (Oyedele, 2022).

Bloom čini objekte svjetlećim ako njihova boja pređe prag koji definiramo. Ovaj efekt dodaje dosta svjetlosti u scenu tako što dodaje svjetleći efekt oko svjetlijih boja. Film Grain efekt dodaje čestice na ekran koje dodaju učinak starog filma, znači čini sliku zrnastom. Depth of Field zamagljuje sve što je dalje od specifične distance da bi zadržalo fokus na specifične stvari u sceni. Motion Blur je često korišten post-processing efekt koji simulira zamagljenost slike koja se desi kad se objekt pomjera brže od vremena izloženosti (eng. exposure time) kamere. Ovaj efekt pomaže sceni da izgleda prirodnije jer replicira ono što ljudsko oko vidi. Uz spomenute efekte naknadne obrade, za poboljšavanje atmosfere i ambijenta, u Kuroi Yuki se koristi osvjetljenje. Namještanje svjetlosti i sjena može imati značajan efekt na osobu na podsvjesnoj razini. Smjer, kontrast, suptilnost i boja svjetlosti mogu značajno promijeniti poruku koju scena šalje igraču (Allén, 2021).

## 5.7 Menadžeri

Menadžeri su skupina skripti i komponenata u projektu koje se koriste za neke općenite funkcionalnosti koje pravimo s ciljem mogućnosti pozivanja kroz cijelu igricu. Menadžeri koji su korišteni u Kuroi Yuki će biti u nastavku opisani te će biti objašnjen način na koji su oni specifični za igricu ovog žanra.

### 5.7.1 Menadžer Igre

GameManager je skripta koja je korištena da bi se stvorila mogućnost upravljanja nekih generičnih funkcionalnosti u igrici. GameManager je Singleton. Singleton je klasa/skripta koja postoji samo jednom u igri i nema više referenciranih ili napravljenih kopija od nje. Ova šablona (eng. design pattern) pomaže da se ova klasa poziva u bilo kojoj drugoj skripti bez da bude svaki put referencirana. U nastavku se nalazi kod skripte GameManager.cs.

```
#region Singleton
    private static GameManager instance;
    public static GameManager Instance
    {
        get
        {
            return instance;
        }}
    void Start(){
        string path = Application.persistentDataPath + "/player.save";
        if(File.Exists(path)){
            LoadData();
        }
        AudioManager.Instance.Play(SoundType.BackgroundTheme);
        AudioManager.Instance.Play(SoundType.Wind);
        fadeInCanvas.SetActive(true);
        fadeInCoroutine = StartCoroutine(fadeThePlayerIn());
    }
```

*Isječak Koda 48 Skripta GameManager.cs 1. dio*

Na početku je definirana Singleton regija i napravljena GameManager instanca. U Start() funkciji provjerava se da li postoji save file i ako postoji, učitava se i „load-a“ igrica. Pokreću se odabrani zvukovi i pokreće se korutinu koja izblijedi kanvas i polako igrača uključi u igricu. U nastavku se nalazi kod u Awake() funkciji GameManager.cs skripte.

```
void Awake()
{
    if (instance == null)
    {
        instance = this;
    }
    else
    {
        Destroy(gameObject);
        return;
    }
    string path = Application.persistentDataPath + "/player.save";
    if(File.Exists(path))
    {
        LoadData();
    }}
```

*Isječak Koda 49 Skripta GameManager.cs 2. dio*

Start() funkcija se pozove na frame-u kad je skripta omogućena prije nego Update() funkcije isto kao i Awake() funkcija. Razlika između Start() i Awake() je da se Start() poziva samo jednom u životnom vijeku skripte ali Awake() se poziva svaki put kad je objekt koji drži skriptu inicijaliziran. Ako je instanca jednaka null, instanca se jednači s klasom, ako nije onda se uništava GameManager. Ponovo na svakom Awake(), učitava se igrica ako postoji save-file. U nastavku se nalazi blackSnowEvent() funkcija.

```
    public void blackSnowEvent(GameObject snowTrigger){
        if(snowActivated == false){
            snowActivated = true;
            snowTrigger.SetActive(false);
            blackSnowParticleSystem.Play();
            Debug.Log("Black Smoke Begin.");
            AudioManager.Instance.Play(SoundType.BlackSnowAtmospheric30);
            AudioManager.Instance.Play(SoundType.GeigerCounterWeak);

            StartCoroutine(waitThenStop(duration));
        }}
```

*Isječak Koda 50 Skripta GameManager.cs 3. dio*

BlackSnowEvent() funkcija kreće tako što se provjerava da li je ovaj događaj već pokrenut. Ako nije onda se pokreće događaj i briše collider koji pokreće crni snijeg. Nakon toga aktivira se sustav čestica i puste prikladne zvukove te se pokreće waitThenStop() korutina koja je prikazana u isječku u nastavku.

```
    IEnumerator waitThenStop(int arg){
        yield return new WaitForSeconds(arg);
        Debug.Log("Black Smoke End.");
        blackSnowParticleSystem.Stop();
        AudioManager.Instance.Stop(SoundType.BlackSnowAtmospheric30);
        AudioManager.Instance.Stop(SoundType.GeigerCounterWeak);
        snowActivated = false;
    }
```

*Isječak Koda 51 Skripta GameManager.cs 4. dio*

WaitThenStop() korutina omogućava da nakon što je pokrenuta blackSnowEvent() funkcija, čeka se određeno vrijeme pa je zaustavljena. U nastavku je prikazana SaveData() funkcija u GameManager.cs skripti.

```
    public void SaveData()
    {
        SaveSystem.SavePlayer(mindsEyeController, theFinalEvent);
    }
    public void LoadData()
    {
        PlayerData data = SaveSystem.LoadPlayer();
        mindsEyeController.bellActivatable = data.hasBell;
        if(mindsEyeController.bellActivatable)
        {
            bellFloor.SetActive(false);
            bellHand.SetActive(true);
            bellCooldownSlider.SetActive(true);
        }
        else
        {
            bellFloor.SetActive(true);
            bellHand.SetActive(false);
            bellCooldownSlider.SetActive(false);
        }
        Vector3 position;
        position.x = data.position[0];
        position.y = data.position[1];
        position.z = data.position[2];
        playerHealth.transform.position = position;

        theFinalEvent.holdingChaosCube = data.holdingChaosCube;
    }
    #endregion
```

*Isječak Koda 52 Skripta GameManager.cs 5. dio*

Postoji SaveData() funkcija koja poziva SavePlayer() funkciju iz SaveSystem skripte te sačuva napredak igrača u igrici. Nakon toga se definira LoadData() funkcija u kojoj se učitavaju podaci i nastavlja se napredak igrice.

### 5.7.2 Sustav Sačuvanja Napretka

Sustav sačuvanja igrice se koristi da bi se mogao spasiti napredak igrice u datoteku u našem računalu. U isječku ispod su prikazane SavePlayer() i LoadPlayer() funkcije u SaveSystem.cs skripti.

```
 public static void SavePlayer(MindsEyeController mindsEyeController,
TheFinalEvent theFinalEvent)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        string path = Application.persistentDataPath + "/player.save";
        FileStream stream = new FileStream(path, FileMode.Create);
        PlayerData data = new PlayerData(mindsEyeController, theFinalEvent);
        formatter.Serialize(stream, data);
        stream.Close();
    }
    public static PlayerData LoadPlayer(){
        string path = Application.persistentDataPath + "/player.save";
        if(File.Exists(path))
        {
            BinaryFormatter formatter = new BinaryFormatter();
            FileStream stream = new FileStream(path, FileMode.Open);
            PlayerData data = formatter.Deserialize(stream) as PlayerData;
            stream.Close();
            return data;
        }
        else
        {
            Debug.LogError("Save file not found in " + path);
            return null;
        }
    }
```

*Isječak Koda 53 Skripta SaveSystem.cs 1. dio*

SavePlayer() funkcija prvo definira binarni formater koji pretvara teks u binarni kod. Nakon tog definiran je put na sustavu gdje će datoteka biti spašena. Napravljena je datoteka i u nju su spašeni podaci iz PlayerData skripte koja će biti u nastavku prikazana. Koristi se formater te se zatvara file stream.

U LoadPlayer() funkciji pronalazi se put do save datoteke i onda se provjerava da li postoji ta datoteka. Ako postoji koristi se formater da se pročita datoteku i deserijalizira, znači pretvara se iz binarnog koda u tekst. Zatvara se file stream i vraćaju podaci. Ako datoteka ne postoji onda se logira error i vraća se null. Isječak koda u nastavku prikazuje kod unutar PlayerData.cs skripte.

```
[System.Serializable]
public class PlayerData{
    public bool hasBell;
    public float[] position;
    public bool holdingChaosCube;
    public PlayerData(MindsEyeController mindsEyeController, TheFinalEvent
theFinalEvent)
    {
        hasBell = mindsEyeController.bellActivatable;
        position = new float[3];
        position[0] = mindsEyeController.transform.position.x;
        position[1] = mindsEyeController.transform.position.y;
        position[2] = mindsEyeController.transform.position.z;
        holdingChaosCube = theFinalEvent.holdingChaosCube;
    }
}
```

*Isječak Koda 54 Skripta PlayerData.cs 1. dio*

U PlayerData skripti postoji klasa koja definira podatke koje želimo spasiti. U konstruktoru se spašava da li igrač drži zvonce, da li igrač drži chaos cube objekt i spašava se pozicija igrača u svijetu.

### 5.7.3 Menadžer Zvuka

AudioManager skripta se koristi kao Singleton da bi se moglo bilo gdje u projektu pristupiti zvukovima koji su spašeni. U isječcima koda u nastavku je prikazana AudioManager.cs skripta.

```
    public AudioSound[] sounds;
    private AudioSound Find(SoundType s)
    {
        return Array.Find(sounds, sound => sound.type == s);
    }
```

*Isječak Koda 55 Skripta AudioManager.cs 1. dio*

U AudioManager skripti prvo su pronađeni svi zvukovi iz AudioSound skripte. Nakon toga su definirane Play(), Stop() i Pause() funkcije koje pauziraju, zaustavljaju i puštaju audio klipove. Horor igrice se snažno oslanjaju na korištenje zvukova da bi se utjecalo na emocije igrača s obzirom da igrači obraćaju pažnju na zvukove u igricama (Gibson, 2016). Naspram vizualnih informacija koji su ograničeni na ekranu, zvukovi omogućavaju stvaranje kontinuiteta i prisutnosti s događajima u naraciji i okruženju.

```
public void Play(SoundType s){
    var playSound = Find(s);
    if (playSound != null)
    {
        playSound.source.Play();
    }
}
```

*Isječak Koda 56  Skripta AudioManager.cs 2. dio*

Stop() i Pause() funkcije izgledaju isto kao i Play() funkcija samo umjesto playSound.source.Play() stavljamo Stop() ili Pause(). U isječku koda ispod je prikazano inicijaliziranje izvora zvukova.

```
private void InitSources(){
    foreach(var sound in sounds){
        var source = gameObject.AddComponent<AudioSource>();
        source.clip = sound.audioClip;
        source.loop = sound.loop;
        source.volume = sound.volume;
        source.pitch = sound.pitch;

        source.outputAudioMixerGroup = sound.outputAudioMixerGroup;
        sound.source = source;
    }}
```

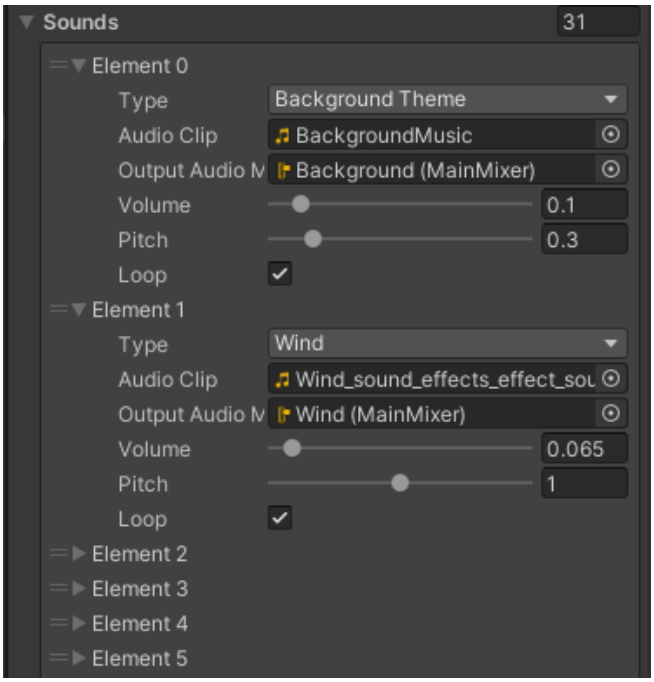
*Isječak Koda 57  Skripta AudioManager.cs 3. dio*

U InitSources() funkciji svakom zvuku se dodaju svojstva na koja se može utjecati u editoru. AudioSound.cs skripta koja je prikazana u idućem isječku koda sadržava definicije svih zvukova koje je potrebno pokrenuti u 2D modu. 2D tipovi zvukova su oni koji nisu vezani za poziciju objekta u odnosu na igrača već ih igrač čuje gdje god da se AudioListener nalazi naspram AudioSource-a. 3D zvukovi se čuju glasnije ako je AudioListener blizu AudioSource-a i obrnuto.

```
public enum SoundType{
    BackgroundTheme,
    Wind,
    BlackSnowBurn,
    . . .
}
[System.Serializable]
public class AudioSound{
    public SoundType type;
    public AudioClip audioClip;
    public AudioMixerGroup outputAudioMixerGroup;
    [Range(0f, 1f)]
    public float volume;
    [Range(0f, 2f)]
    public float pitch;
    public bool loop;
    [HideInInspector]
    public AudioSource source;
}
```

*Isječak Koda 58 Skripta AudioSound.cs 1. dio*

Vidi se da su u AudioSound.cs skripti definirani audioklipovi i napravljene definicije svojstava koji su korišteni za sve zvukove. Na slici ispod su prikazani elementi zvukova dodani u inspektoru/editoru.




*Slika 36 Lista Zvukova od Audio Menadžera*

### 5.7.4 Menadžer Smrti

DeathManager je skripta koja se koristi da se upravlja slučajevima u kojim glavni lik umre i pokrenu se scene koje sadržavaju odgovarajuće videe za svaki tip smrti igrača. U idućim isječcima koda će biti prikazana skripta DeathScreenManager.cs.

```
    private AudioSource[] allAudioSources;
    public VideoPlayer VideoPlayer;
    void Start(){
         StopAllAudio();
         VideoPlayer.loopPointReached += EndReached;
    }
    void StopAllAudio()
    {
     allAudioSources = FindObjectsOfType(typeof(AudioSource)) as AudioSource[];
     foreach( AudioSource audioS in allAudioSources) {
         audioS.Stop();
    }
}
```

*Isječak Koda 59 Skripta DeathScreenManager.cs 1. dio*

Ova skripta se nalazi na GameObjektu u „death scenes“ tako da se Start() funkcija pozove tek kad igrač umre u igrici. Na početku se zaustave svi zvukovi.

```
    void EndReached(UnityEngine.Video.VideoPlayer vp)
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Main Scene");
        StopAllAudio();
        Cursor.lockState = CursorLockMode.None;

        string path = Application.persistentDataPath + "/player.save";
        if(File.Exists(path))
        {
            GameManager.Instance.LoadData();
        }
    }
```

*Isječak Koda 60 Skripta DeathScreenManager.cs 2. dio*

EndReached() funkcija učitava glavnu scenu nakon kraja odgovarajućeg videa smrti. Nakon tog se učitava sačuvan napredak u igrici tako da igrač može nastaviti od zadnje točke sačuvanog napretka. U isječku koda u nastavku, prikazuje se Update() funkcija opisane skripte.

```
    void Update()
    {
        if(Input.GetKey(KeyCode.Space))
        {
            Time.timeScale = 1f;
            SceneManager.LoadScene("Main Scene");
            StopAllAudio();
            Cursor.lockState = CursorLockMode.None;

            string path = Application.persistentDataPath + "/player.save";
            if(File.Exists(path))
            {
                GameManager.Instance.LoadData();
            }
        }
    }
```

*Isječak Koda 61 Skripta DeathScreenManager.cs 3. dio*

U Update() funkciji na svakom frame-u se provjerava da li je igrač pritisnuo gumb „SPACE“. Ako jeste onda se preskoči video smrti i odmah se pređe na glavnu scenu i učita napredak igrača. Ovo omogućava da se preskoči snimak nakon smrti i da igrač ne mora svaki put gledati scenu ako često gubi/umire u igrici.

### 5.7.5 Zadnji Događaj

The Final Event je zadnja sekcija u ovom radu i opisuje zadnji događaj koji se desi u igrici. Nakon što igrač odnese objekt „Chaos Cube“ do postolja kod objekta „The Propagator“, aktivira se zadnji događaj pod imenom „The Final Event“. U nastavku je prikazan kod koji čini logiku upravljanja zadnjim događajem.

```
    public void pickUpChaosCube(){
        chaosCube.SetActive(false);
        hillGate.SetActive(false);
        AudioManager.Instance.Play(SoundType.GateOpen);
        holdingChaosCube = true;
    }
```

*Isječak Koda 62 Skripta TheFinalEvent.cs 1. dio*

U skripti TheFinalEvent.cs se nalazi logika podizanja objekta „ChaosCube“. Nakon ovoga postoji javna funkcija startFinalEvent() koja se poziva u Raycast() skripti nakon što igrač pritisne gumb „E“ dok gleda na postolje kod propagatora i drži „chaos cube“ objekt. U idućem isječku su prikazane skripte startFinalEvent() i stopFinalEvent().

```
public void startFinalEvent()
  {
      holdingChaosCube = false;
      chaosCubeOnThePropagator.SetActive(true);
      chaosCubeOnThePropagatorCollider.enabled = false;
      TheFinalEventCoroutine = StartCoroutine(theFinalEvent());
  }
  public void stopFinalEvent()
  {
      finalEventPostProcessing.SetActive(false);
      currentPostProcessing.SetActive(true);
  }
```

*Isječak Koda 63 Skripta TheFinalEvent.cs 2. dio*

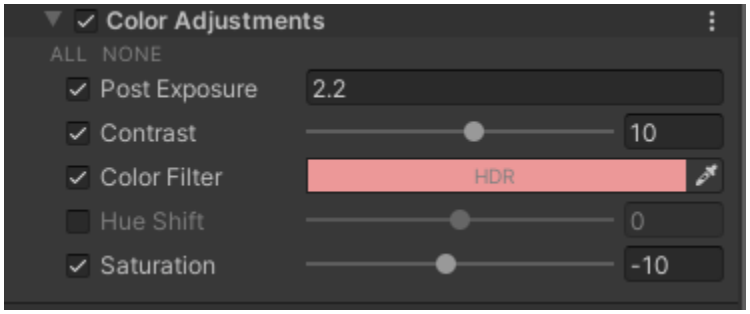
Nakon startFinalEvent() funkcije u kojoj se pokreće theFinalEvent() korutina koja je prikazana u idućem isječku koda, definirana je funkcija stopFinalEvent(). U njoj su vraćeni efekti naknadne obrade na standardne.

```
IEnumerator theFinalEvent(){
      yield return new WaitForSeconds(2);
      secondaryCameraFinalEventPostProcessing.SetActive(true);

      mindsEyeController.bellActivatable = false;
      mindsEyeController.currentCooldown = 100;
      mindsEyeController.activateEye();

      audioMixer.SetFloat("Volume", -80f);

      yield return new WaitForSeconds(4);
      mindsEyeController.deactivateEye();

      audioMixer.SetFloat("Volume", 0f);
      audioMixer.SetFloat("BackgroundVolume", -80f);

      currentPostProcessing.SetActive(false);
      finalEventPostProcessing.SetActive(true);
      bellHand.SetActive(false);
      bellHandSlider.SetActive(false);

      StartCoroutine(cameraShake.Shake(300f, .6f, false));

      AudioManager.Instance.Play(SoundType.ThePropagatorNoise);
      AudioManager.Instance.Play(SoundType.GeigerCounterStrong);
      AudioManager.Instance.Play(SoundType.HellHornInstant);
```

```
        foreach(GameObject gate in finalPathGate)
            gate.SetActive(false);
        yield return new WaitForSeconds(5);
        AudioManager.Instance.Play(SoundType.HellHorn1);
    }
```

*Isječak Koda 64 Skripta TheFinalEvent.cs 3.dio*

U theFinalEvent() korutini prvo se čekaju dvije sekunde. Nakon toga se uključe efekti naknadne obrade koji su specifični za zadnji događaj. Glavna razlika kod ovog post-processing profila je promijenjena boja. U nastavku su prikazane promjene boje u postavkama naknadne obrade te prikaz scene s ovim efektima primijenjenim.




*Slika 37 Profil Naknadne Obrade Zadnjeg Događaja*

*Slika 38 Naknadna Obrada Zadnjeg Događaja*

U nastavku je aktivirana Mind's Eye sposobnost, utišan zvuk nakon čega se čekaju četiri sekunde. Nakon toga je deaktivirana sposobnost te je pojačan zvuk. Pušteni su odgovarajući zvučni efekti, pokrenuta je funkcija za protresanje kamere i simuliran je zadnji prepad u igrici.

Nakon ovoga igrač hoda prema nuklearnoj eksploziji, kad dođe do određenog collidera aktivira se posljednja animacija u igrici. Ovo je prikazano u isječku koda ispod.

```
if(collision != null && collision.gameObject.CompareTag("FinalEventCamera"))
        {
            finalCutsceneCamera.SetActive(true);
            mainCamera.SetActive(false);
            fadeInCanvas.SetActive(true);
            crosshair.SetActive(false);
            finalCameraCutsceneAnimator.enabled = true;
            finalCameraCutsceneAnimator.SetTrigger("ActivateFinalCutscene");

            if(WaitForSecondsThenDoCoroutine != null)
                StopCoroutine(WaitForSecondsThenDoCoroutine);
          WaitForSecondsThenDoCoroutine = StartCoroutine(waitForSecondsThenDo());
        }
```

*Isječak Koda 65 Skripta PlayerCollision.cs, dio povezan s zadnjim događajem 1. dio*

Zadnja animacija pokreće kameru polako prema nebu dok gleda prema dolje. Igrač vidi dva tijela na podu i jedno od njih drži Mind's Eye Zvonce. Pri završetku animacije scena se promijeni na „Main Menu Scene“ i igrica se završava.

# 6 Zaključak

Kroz ovaj rad, obrađeni su svi elementi koji su prisutni u atmosferskoj psihološkoj horor igrici Kuroi Yuki. Opisani su svi prisutni koncepti i elementi, te su čitatelji stekli znanje potrebno za razvoj igrice istog ili srodnog žanra. Rad je pokrio sustav čestica (eng. particle system), naknadnu obradu (eng. post processing), raycasting, okruženje igrice, skripte, animacije, i brojne druge elemente prisutne u igrici.

Prođeno je kroz priču igrice te su opisani likovi, tema i način pripovijedanja u igrici. Opisani su UI elementi, shaderi (eng. shaders) i graf shader-a (eng. shader graph). Kolizije i okidači su objašnjeni te je prikazana njihova uporaba u Kuroi Yuki. Prikazani su unikatni elementi prisutni u igrici kao Mind's Eye sposobnost, funkcionalnost The Smoldering Gazer neprijatelja. Isto tako je opisan sustav sačuvanja napretka u igrici i učitavanja sačuvanog stanja te su prikazani svi menadžeri korišteni u igrici kao GameManager, AudioManager, DeathSceneManager i ostali. Korutine predstavljaju velik dio programiranja igrica u Unity-u te je koncept i korištenje korutina objašnjen i prikazan.

Mogućnosti proširivanja igrice su moguće i neki primjeri toga bi bili proširivanje sustava sačuvanja napretka igrice i dodavanje NavMesh-a i NavMesh Agent neprijatelja, koji bi imao mogućnost pokretanja po sceni. U razvoju Kuroi Yuki, izabran je manje tradicionalan način borbe s neprijateljem i njegovog izgleda te je više fokusirano na koncept okruženja kao neprijatelja. Isto tako bi mapa mogla biti proširena, te bi se moglo dodati što više bilješki u okruženje.

Dodatni mogući način proširenja i poboljšanja igrice bi bio dodavanje više zvučnih i vizualnih efekata i popunjavanje alternativnog svijeta s više objekata i interaktivnih elemenata. U igrici Kuroi Yuki, glasovni glumci čitaju većinu bilješki koje igrač pronađe u okruženju, odličan način produbljivanja igrice bi bio povećavanje broja glasovnih glumaca i dodavanje animacija na što više objekata u igrici.

Teško je napraviti horor igricu zaista strašnom i često su standardni načini postizanja toga previše jednostavni i nisu dovoljno kreativni ali velika je šansa da će čitatelji ovog rada dobiti dosta različitih ideja i načina da postignu taj cilj.

**Sažetak** Ključne riječi: unity, c#, horor, videoigra, programiranje, razvoj videoigara.

*Cilj ovog diplomskog rada je prikaz, objašnjenje i opis procesa izrade i implementacije elemenata u atmosferskoj psihološkoj horor igrici Kuroi Yuki. Prikazani su svi elementi korišteni pri razvoju ove igrice te su objašnjene sve skripte i komponente povezane s funkcionalnosti i izgledom igrice. Objašnjen je način na koji su teme i atmosfera/ambient u igrici ovakvog žanra pojačani i zaključen je optimalan način za učiniti isto. U Kuroi Yuki postoje unikatni elementi kao Mind's Eye sposobnost gledanja u alternativnu dimenziju i rijetko viđen način napadanja od neprijatelja Smoldering Gazer koji uključuje povlačenje kamere od igrača prema neprijateljevom oku. Način na koji je napravljen Crni Snijeg je objašnjen te su dodatno opisani sustav čestica, naknadna obrada, animacije, UI elementi, skripte s kojima utječemo na elemente u igrici, pokretanje igrača, i tako dalje. Čitatelji ovog rada će na kraju rada naučiti puno stvari o razvijanju trodimenzionalnih igrica iz perspektive prvog lica i unaprijedit će svoje znanje o razvijanju atmosferske psihološke horor igrice.*

**Abstract** Key words: unity, c#, horror, videogame, programming, game development.

*The purpose of this master's thesis is the display, explanation and description of the process of development and the implementation of the elements of the atmospheric psychological horror game Kuroi Yuki. All of the elements used in the development of this videogame are shown and all of the scripts and components connected to the functionality and design are explained. Furthermore, the way that the themes and the atmosphere / ambiance in a videogame of this genre are amplified is explained and the optimal way to implement such a concept is concluded. In Kuroi Yuki, there exist unique elements such as the Mind's Eye ability to peer into an alternative dimension and the rarely seen way of attacking seen in the Smoldering Gazer enemy, which includes dragging the players camera towards the Gazer's Eye. The way in which the Black Snow is made is explained along with the particle system, post processing, animation, UI elements, scripts with which we control the elements in the game, player movement and many more concepts within Kuroi Yuki. The readers of this thesis will ultimately learn many different approaches and concepts about the development of three-dimensional first-person videogames and will greatly further their knowledge of making atmospheric psychological horror videogames.*

# Literatura:


Allén, A. E. (March 2021). Dohvaćeno iz https://www.theseus.fi/bitstream/handle/10024/495036/Game%20Design%20Thesis%20-%20Antti%20Allen.pdf?sequence=2&isAllowed=y

Amilin, J. (8. May 2021). OnCollisionEnter vs. OnTriggerEnter and when to use them.

Cammarata, N. (Spring 2016). Dohvaćeno iz https://academics.depaul.edu/honors/curriculum/Documents/2016%20Senior%20Theses/Cammarata,%20Nina%20Senior%20Thesis%20SQ15-16.pdf

ÇAMÖNÜ, İ. (19. February 2020). Vertex Manipulation using Shader Graph in Unity3D.

Carrillo, U. (30. June 2022). Using Coroutines in Unity.

French, J. (13. July 2021). How to Rotate in Unity.

French, J. (18. June 2021). Raycasts in Unity, made easy.

Fuentes, L. (28. July 2021). Blender Mesh: All You Need to Know.

Gibson, J. (April 2016). Dohvaćeno iz https://www.researchgate.net/publication/315818806_Fear_-_The_Impact_of_Virtual_Reality_in_Horror_Games#pfe

Griffin, M. (May 2019). Dohvaćeno iz https://core.ac.uk/download/pdf/199235889.pdf

Haas, J. K. (2014). *A history of the unity game engine.* Dohvaćeno iz https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf

Lilett, D. (20. May 2021). How To Use Every Node in Unity Shader Graph.

Mobsby, N. (12. November 2021). Blender: How to Add a Vertex.

Nielsen, D. L. (Spring 2013). Dohvaćeno iz https://projekter.aau.dk/projekter/files/76674331/In_the_mood_for_horror.pdf#page=21&zoom=100,72,621

Oyedele, T. (5. September 2022). Exploring post-processing in Unity.

Stegner, B. (26. November 2020). First-Person Games vs. Third-Person Games: What Are the Differences?

Unity. (2021). The Unity Engine Documentation.


# Kuroi Yuki GitHub: https://github.com/mseparov/Kuroi-Yuki

# Popis slika:

# Popis isječaka koda: