Izrada videoigre za više igrača žanra Boss Rush u programskom alatu Unity

Rožić, Luka

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:591811

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Rožić

IZRADA VIDEOIGRE ZA VIŠE IGRAČA ŽANRA BOSS RUSH U PROGRAMSKOM ALATU UNITY

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Rožić

JMBAG: 0016129859

Studij: Informacijsko i programsko inženjerstvo

IZRADA VIDEOIGRE ZA VIŠE IGRAČA ŽANRA BOSS RUSH U PROGRAMSKOM ALATU UNITY

DIPLOMSKI RAD

Mentor/Mentorica:

Doc. dr. sc. Mladen Konecki

Varaždin, srpanj 2022.

Luka Rožić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad temelji se na izradi video igre žanra *Boss Rush*. Sama videoigra podržava, uz glavnog igrača, priključenje još jednog, dodatnog igrača. Prije nego što će se govoriti o izradi videoigre, nužno je prvo reći općenito što je *Boss Rush*, kada i kako je nastao te koji elementi videoigara su prisutni u tom žanru. Navest će se najbolji primjeri videoigara koji su tog žanra ili imaju njegove elemente igrivosti (engl. *gameplay*). Reći će se nešto i o programskom alatu *Unity* koji se koristio za razvoj videoigre, kao i o Microsoft Visual Studiju koji je služio za razvoj logike videoigre, odnosno skripata. Osim prethodno spomenutih alata, za spremanje podataka u bazu koristila se Googleova platforma *Firebase* te će se nešto i o njemu reći. Tada slijedi opis, točnije koncept izrađene videoigre, a nakon toga će se opisati proces izrade videoigre, što i kako se sve razvilo, pri čemu će naglasak biti najviše na programskom kodu. Na kraju rada slijedi zaključak.

Ključne riječi: videoigre, *Boss Rush*, podrška za više igrača, *Unity*, *Visual Studio*, *Firebase*, C#, dizajn i razvoj

Sadržaj

1. Uvod

Videoigre više nisu relativno nova pojava. One dovoljno dugo postoje kao jedan od oblika zabave, stoga su se razvili razni načini igranja i prepričavanja priče *videoigre*, kao i raznovrsni žanrovi i podžanrovi *videoigara*. Porastom sveukupnog broja igrača porastao je i broj razvijača (engl. *developers*) videoigara, što je zapravo uzrok sve većeg broja izdavanja videoigara na tržištu, bile one inovativno i kvalitetno izrađene ili ne. Ono što je sigurno jest da se s vremena na vrijeme uvijek izradi takva igra koja će se u bližoj ili daljoj budućnosti koristiti kao referenca za najbolji primjer žanra kojem pripada ili kao primjer za neki element koji u svojoj vrsnosti prednjači u usporedbi s nekim drugim videoigrama, što nije jednostavno za postići u današnjem vremenu kada je mnogo toga već viđeno. Ono što jest nova pojava, novi oblik umjetničkog izražavanja *developera* preko videoigara jest *Boss Rush*.

Boss Rush kao pojam nije nešto novo igračima jer je prisutan još i od prije kao jedan od dijelova starih videoigara koje igrač mora proći. Vrlo je važno objasniti što je točno *Boss Rush* kako bi se ostatak rada mogao razumjeti. Stoga će se u radu prvo ukratko prepričati povijest *Boss Rusha*, navesti i objasniti njegove glavne karakteristike i predstavnici. Poslije toga će se nešto kratko reći i o *Unityju*, programskom alatu koji se koristio za izradu videoigre na kojoj se ovaj rad temelji. Uz *Unity* će se opisati i *Microsoft Visual Studio* korišten za izradu skripata videoigre te Googleov *Firebase* koji se koristio za pohranu postignutih rezultata od igrača. Isto tako, navest će se nekoliko glavnih pojmova koji će se nekoliko puta spomenuti tijekom rada. Imajući tada ideju što je *Boss Rush* i što se sve od alata koristilo za izradu videoigre, slijedi objašnjenje koncepta video igre, odnosno što je glavni cilj video igre, koja je priča te na što će se sve staviti naglasak. S obzirom na to da se videoigra najviše temelji na borbi protiv snažnih neprijatelja, velik dio rada bit će usmjeren upravo prema njima, kako programski dio, tako i prezentacijski. Reći će se nešto i o podršci videoigre za više, točnije dva igrača i način na koji se to realiziralo. Uz sve navedeno govorit će se i o nekim ostalim elementima, poput animiranja likova, stvaranju raznih specijalnih efekata i o stvaranju samih scena videoigre. Za kraj će se donijeti zaključak o ovom praktičnom radu.

Glavni, veliki neprijatelji kod videoigara oduvijek su mi bili posebna iskustva jer su do samih granica znali testirali moje igrače vještine: strpljivost, brzo reagiranje, taktiku i mudrost. Osobno volim velike izazove kod videoigara, stoga sam odabrao da tema ovoga rada bude posvećena upravo njima. *Unity* sam odabrao jer sam od prije upoznat s njim, kao i s *Visual Studiom*.

2. Metode i tehnike rada

Za izradu videoigre, kao i za izradu ovoga diplomskoga rada, koristile su se različite vrste izvora. Ponajprije su se odigrale mnoge videoigre kako bi se bolje shvatio pojam *Boss Rusha* i kakvu ulogu on ima u svijetu *videoigara*. Pogledali su se razni videozapisi u kojima se objašnjavao *Boss Rush* i videoigre s podrškom za više igrača te pročitali razni forumi u kojima su ljudi iznosili svoja mišljenja o takvoj vrsti videoigre. Potrebno je bilo čitati i službenu dokumentaciju *Unityja* za razne metode koje su se koristile prilikom izrade programskog koda, kao i službenu dokumentaciju Microsofta za mnoge biblioteke koje *Visual Studio* nudi. Knjige se nisu koristile za proučavanje žanrova videoigara, nego za primjenjivanje najboljih praksa za pisanje efikasnog programskog koda.

3. *Boss Rush* videoigre

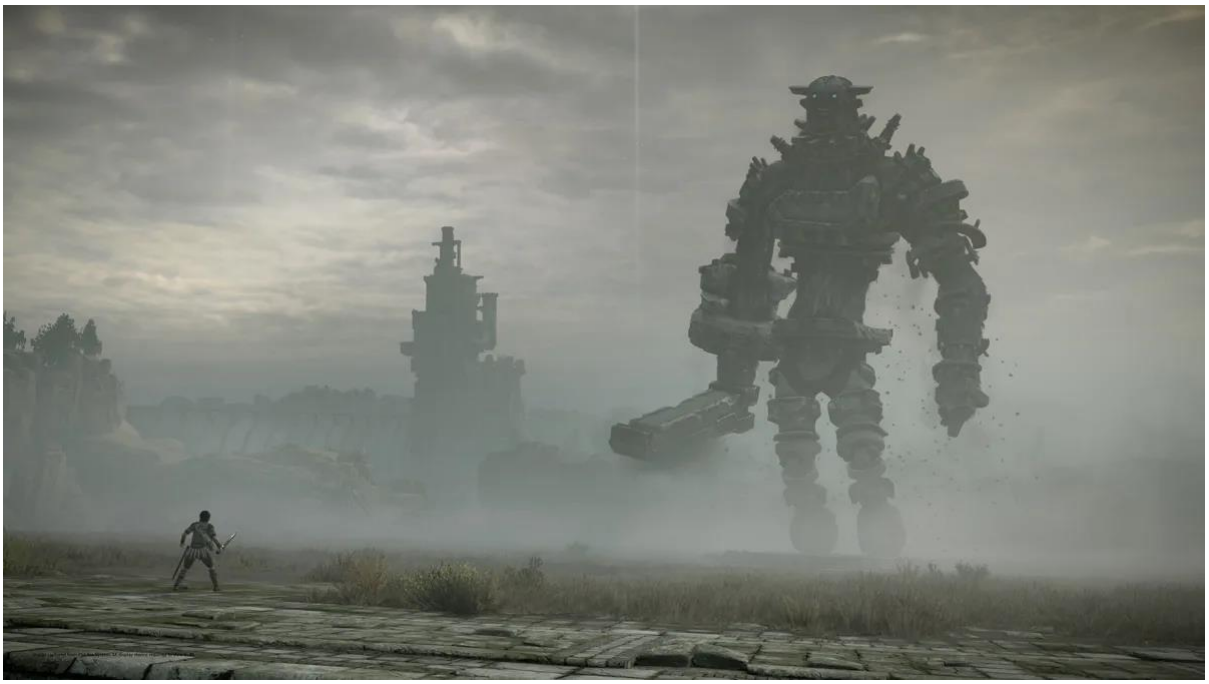
Igračima su poznati žanrovi videoigara poput platformera, pucačina, akcijskih avantura i tako dalje, no *Boss Rush* se donedavno još nije smatrao podžanrom, a kamoli žanrom. Razlog tomu jest da je *Boss Rush* zapravo jedan manji dio cijele videoigre koji su igrači morali nadvladati kako bi mogli dalje prolaziti kroz glavnu priču, no u novije vrijeme to se promijenilo. Kako se točno *Boss Rush* razvio iz jednog manjeg dijela *gameplaya* videoigre u zaseban (pod)žanr, reći će se u sljedećem poglavlju.

3.1. Povijest *Boss Rusha*

Glavni neprijatelj (engl. *boss*) je takva vrsta neprijatelja u videoigrama koji predstavljaju veću prepreku igračima negoli mini glavni i obični neprijatelji. Podrijetlo takvog nazivanja zadnjeg većeg neprijatelja nije posve poznato, no nagađa se da je taj pojam usko povezan s pojmom glavni šef kriminalne organizacije (engl. *crime boss*) [1].

Prva igra koja je upotrijebila takvog većeg i složenijeg neprijatelja u obliku zlatnoga zmaja (engl. *Golden Dragon*) jest *dnd* za PLATO *system* iz 1975. godine, što je tekstualna avantura inspirirana stolnom društvenom igrom *Dungeons & Dragons* [2]. Nadalje se tijekom 80-ih godina 20. stoljeća počelo uvelike koristiti takve vrste neprijatelja, najčešće pri kraju svake razine/tamnice u videoigri. Dobri primjeri za to su *The Legend of Zelda* iz 1986. godine koju je razvila tvrtka Nintendo za kućnu konzolu *Nintendo Entertainment System* (NES) [3], *Contra* iz 1987. godine koju je razvio Konami [4], dok je u istoj godini Capcom razvio i pustio u tržište *Mega Man* [5]. Naglasak će se staviti na *Mega Man* zato što kod kraja te igre igrači moraju ponovno pobijediti sve glavne neprijatelje s kojima su već prije borili, što je zapravo glavna karakteristika *Boss Rush* videoigara. Primjeri videoigara iz 8-bitnog i 16-bitnog razdoblja koje su koristile takav koncept su *ActRaiser* (1991) [6], *Kirby's Dream Land* (1992) [7], *Castlevania: Bloodlines* [8] i tako dalje.

Videoigra koja je zbog svoje popularnosti i vrhunskom *gameplayu* veliki korak naprijed napravila za *Boss Rush* jest *Shadow of the Colossus* (2005) [9]. Ta videoigra, za razliku od prije spomenutih primjera, temelji se prvenstveno na borbi protiv glavnih neprijatelja, što znači da ne postoji prolaz razina igre u kojem postoje obični neprijatelji i prepreke, nego se sastoji od kratkog putovanja do arena i od borbi protiv velikih kolosa koji se nalaze u tim arenama, što je zapravo srž *Boss Rush* videoigara.



Slika 1. *Shadow of the Colossus*: borba protiv jednog od mnogih kolosa [10]

U novije vrijeme nezavisni razvijači videoigara (engl. *indie developers*) su ponovno oni koji populariziraju ne toliko znane i popularne žanrove. Primjeri videoigara koje se prvenstveno temelje na borbi protiv glavnih neprijatelja jesu: *Titan Souls* [11], *Jotun* [12] i poznati *Cuphead* [13] koji, iako sadrži nekoliko razina koji su tipični za platformerske videoigre, većinu vremena se igrači bore protiv velikih neprijatelja [14].

4. Elementi *Boss Rusha*

Svi elementi koje *Boss Rush* videoigra ima su zapravo elementi prisutni i u platformerima, akcijskim avanturama, akcijskim i borilačkim videoigrama. Drugim riječima, Boss Rush je „posudio“ elemente koji su karakteristični za druge žanrove videoigara. Kako bi se rečeno bolje shvatilo, opisat će se u sljedećim poglavljima:

- *gameplay* (cilj, izazovi, kontrole, podrška za više igrača)
- prezentacija (umjetnički stil i dizajn, glazba, atmosfera i priča)

4.1. Gameplay

Gameplay Boss Rush videoigara je izravan i jednostavan. Kod takvih igara obično se ne istražuje svijet, nije naglasak na sakupljanju raznih stvari i moći te prolazak razina, iako navedene stvari znaju biti prisutne u poznatim *Boss Rush* videoigrama poput *Cupheada*, no svejedno ne predstavljaju neki značajan dio igre. Kod nekih postoji mijenjanje oružja i moći glavnog protagonista, dok se kod drugih igrač oslanja samo na jedno oružje, bio to mač, strijela ili nešto drugo.

4.1.1. Cilj i izazovi

Cilj svake *Boss Rush* videoigre je isti, a to je pobijediti sve glavne neprijatelje koji čekaju igrača. Ti glavni neprijatelji, odnosno *bossevi* razlog su zašto se te igre i zovu *Boss Rush* jer su bitke protiv njih temelj žanra. Upravo te bitke s glavnim neprijateljima predstavljaju velike izazove za svakog igrača, stoga je teško reći koliko bi otprilike neka *Boss Rush* videoigra trajala jer to ovisi o svakom igraču te o njegovim vještinama i brzini reagiranja.

Kako bi Boss Rush videoigra bila kvalitetna, važno je kod nje učiniti svaku bitku s glavnim neprijateljem izazovnom, ali i pravednom, što znači da svaki put kada igrač izgubi svoj život u videoigri, to mora biti njegova krivica, a ne zato što videoigra stvori prepreke ili napade od glavnog neprijatelja koje ponekad jednostavno nije moguće izbjeći. Isto tako, mnogim igračima je znak da je neka *Boss Rush* igra kvalitetna to da iako u videoigri često umiru, svaki put nakon toga su bolji, odnosno vide svoj napredak u obliku lakšeg izbjegavanja napada od glavnog neprijatelja. Isto tako, vrlo je poželjno učiniti svakog glavnog neprijatelja nezaboravnim, što se može postići vizualnim dizajnom, glazbom i atmosferom, ali ponajviše zabavnošću i/ili izazovnošću bitke protiv njega. Osjećaj uspjeha nakon velikog broja pokušaja također je poželjan element kod glavnih neprijatelja.

4.1.2. Podrška za više igrača

Prije nego što se počne govoriti o podršci takvih igara za više igrača, mora se objasniti kakve vrste podrška postoje. Općenito je podjela igara koje podržavaju više igrača sljedeća:

- videoigre za više igrača lokalno (engl. *local multiplayer games*)
- videoigre za više igrača *online* (engl. *online multiplayer games*)

Lokalne *multiplayer*-videoigre su one za koje je potreban samo jedan uređaj (računalo, kućna ili prijenosna konzola) ili samo jedna mreža kako bi se videoigra igrala, dok je za *online* tipično da više igrača igra videoigru na svojim uređajima koristeći internet.

Za *Boss Rush* videoigre karakteristično je da su namijenjene samo jednom igraču, ponekad lokalno dvama, no u posljednje vrijeme dolazi do izražaja i mogućnost igranja takvih igara s ne/poznatim ljudima *online*. Da bi se igru učinilo još zanimljivijom, potrebno ju je dodatno otežati svaki put kada se dodatno priključi još jedan igrač, za što je dobar primjer *Cuphead* koji tada glavnim neprijateljima poveća maksimalan broj životnih bodova (engl. *health points*, HP).

Slika 2. *Cuphead*: primjer borbe dvaju igrača lokalno protiv glavnog neprijatelja [15]

4.1.3. Kontrole

Kontrole kod ovakvih videoigara moraju nužno biti kvalitetno odrađene. To se najviše odnosi na responzivnost uređaja (tipkovnica, miševa, kontrolera) koje igrači koriste prilikom igranja. Responzivnost označava vrijeme koje je potrebno da videoigra dobije obavijest da je igrač pritisnuo tipku na tipkovnici ili gumb na kontroleru. Kada je takvo vrijeme kraće,

responzivnost je veća, što dalje označava i glađe igranje videoigre. Osim toga, danas je čest slučaj da *developeri* daju mogućnost igračima da sve ili gotovo sve akcije protagonista igre poput skakanja i napadanja mogu smjestiti pod one tipke i gumbe koji njima najviše odgovaraju, tzv. mapiranje kontrola.

4.2. Prezentacija

Spomenuto je da je poželjno učiniti svakog glavnog neprijatelja nezaboravnim te da se to može postići putem *gameplaya*. No, ponekad igračima glavni neprijatelji nisu urezani u pamćenje zbog *gameplaya*, nego zbog drugih elemenata, vizualnih ili auditivnih.

4.2.1. Umjetnički stil i dizajn

Glavni vizualni element videoigara predstavlja odabrani umjetnički stil, bilo da se radi o pikselastoj umjetnosti (engl. *pixel art*), umjetnosti za koju se koristila vektorska grafika, 3D modeliranje ili se izgled videoigre dobio klasičnim načinom – crtanjem svijeta i likova videoigara rukom.

Odabirom umjetničkog stila slijedi dizajniranje svijeta i likova pri čemu se vrijeme posebno posvećuje ne samo dizajniranju glavnog protagonista, nego i dizajniranju glavnih neprijatelja. Čest je slučaj da su takvi neprijatelji u odnosu na glavnog protagonista veliki, a nerijetko po izgledu i groteskni, što je jedan od učinkovitih načina da igračima ostanu po nečemu zapamćeni nakon što odigraju videoigru. Mogu biti i manji, primjerice veličine protagonista, no tada se to *gameplayom* uravnoteži na način da su brzi ili imaju veći arsenal napada. U svakom slučaju, dizajn glavnih neprijatelja igračima ne smije biti monoton, nego se po nečemu moraju isticati kada bi ih se išlo međusobno uspoređivati.

4.2.2. Glazba, atmosfera i priča

Glazba *Boss Rush* videoigara je raznolika. Ona može biti energična tako da igrače dodatno uživi u videoigru, može biti tragična, strašna ili epska. Sve spomenuto ovisi o načinu na koji igra želi predstaviti svijet, a na kraju krajeva i o kakvom se svijetu radi i koja priča stoji iza toga svijeta. Ponekad je glazba rijetko prisutna u igri, a primjer je putovanje po svijetu *Shadow of the Colossus* čime se postiže jedna vrsta atmosfere u videoigri.

Priče kod ovakvih videoigara nisu toliko bitne, no one postoje. Prepričavaju se preko teksta koji se igračima pojavi u početku i tijekom igre ili preko animiranih scenskih dijelova. Općenito zna biti slučaj da videoigre svoju priču indirektno ispričaju upravo preko glazbe i atmosfere, a *Boss Rush videoigre* u tome nisu iznimke.

5. Unity, Visual Studio i Firebase

Danas je više nego ikada prije javnosti dostupan velik broj programskih alata za razvoj videoigara, poznat pod nazivom *engines* (hrv. motori). Drugim riječima, više nije samo slučaj da svaki novi tim mora razviti vlastiti *engine*, nego postoje organizacije koje razvijaju takve složene alate te ih stavljaju na svoje službene *web*-stranice za preuzimanje. Primjeri tih organizacija su *Epic Games* sa svojim novim *Unreal Engineom* 5 koji se primarno koristi za izradu 3D video igara koristeći C++ objektno orijentirani jezik [16] i *Unity Technologies* koji nudi programski alat *Unity* čije skripte *developeri* pišu programskim jezikom C# najčešće za izradu 2D videoigara [17] [18]. Druge programske alate koje vrijedi spomenuti su potpuno besplatan Godot čiji izvorni kod je otvoren za izmjene (engl. *open source*) [19], *GameMaker* koji daje mogućnost kreiranja samo 2D videoigara [20], *CryEngine* čija se snaga često uspoređuje s *Unreal Engineom* [21] i mnogi drugi.

Osim *enginea*, potrebno je imati i nešto pomoću čega će se pisati programski kod, za što upravo služe integrirana razvojna okruženja (engl. *integrated development environment*). *JetBrains Rider* [22] se koristi najviše za *Unreal Engine*, dok je *Visual Studio* preferiran za Unity [23].

Ako se radi o igri koja mora spremati podatke u servere, onda je potrebno odabrati prikladan program za upravljanje bazom podataka. Primjeri takvih programa, odnosno baza podataka su sljedeći: MySQL, SQLite, *Firebase*, MongoDB [24] [25] [26] [27].

Za izradu videoigre ovoga rada se upotrijebio *engine Unity* zajedno s *Visual Studijem* i *Firebaseom*. Zato će se ukratko nešto reći i o njima.

5.1. Unity

Unity kao programski alat daje razne mogućnosti korisnicima da svoju kreativnost oblikuju u nešto što se može vidjeti i iskusiti. Pomoću njega nije moguće samo kreirati 2D videoigre nego i 3D. Isto tako, daje korisnicima mogućnost izrade videoigara virtualne stvarnosti (engl. *Virtual Reality*, VR), proširene stvarnosti (engl. *Augmented Reality*, AR) i programe za miješanu stvarnost (engl. *mixed reality*, MR), što sve zajedno čini proširenu stvarnost XR (engl. *extended reality*) [28]. XR *Unity Technologies* kao posebnu značajku alata predstavlja vlastima svijeta koje ju koriste za razne simulacije i treninge. Drugim riječima, *Unity* se više ne koristi samo za izradu videoigara, animacija, filmova, u automobilskoj industriji i kod arhitekture, nego je došao do razine da se koristi na velikim projektima korištenih od vlade [29].

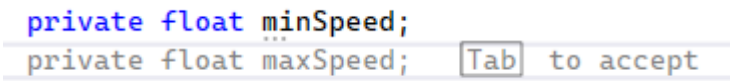
U početku *developerima Unityja* nije bio cilj izraditi takav *engine* koji bi se koristio po cijelome svijetu. Nicholas Francis, Joachim Ante i David Helgason su 2005. godine prvo imali na umu samo razviti videoigru *GooBall* [30, str. 4,7] , a da bi to uspjeli postići, morali su razviti *engine* za to. Na kraju su uvidjeli da su programski alat kvalitetno izradili te su ga odlučili dodatno proširiti raznim funkcionalnostima. U međuvremenu je osnovana tvrtka Unity Technologies koja često u javnost pušta nove verzije *Unity enginea*, bilo da se radi zbog popravaka pogrešaka koje *engine* ima ili jednostavno zbog novih dodataka koje *developeri* kasnije koriste. Upravo se jedan takav novi dodatak koristio i opisao u ovom radu - *Input manager*, no bit će kasnije opisan uz sve ostale korištene mogućnosti koje *Unity* nudi.

5.2. Microsoft Visual Studio

Nužno je imati razvijenu logiku napisanu za videoigru koja bi se na temelju toga smisleno pokretala na način osmišljen od strane *developera*. Stoga je bilo potrebno odabrati razvojno programsko okruženje koje će poslužiti za izradu takve logike (skripte), što je upravo *Microsoftov Visual Studio*.

Visual Studio je odabran zbog svoje bogate ponude biblioteka koje korisnici mogu implementirati i koristiti, podrške za različite programske jezike (C, C#, C++, Python) [31], velike zajednice prisutne na internetu koja je spremna odgovoriti na razna postavljena pitanja i zbog kvalitetne dokumentacije svega onog što *Visual Studio* nudi. Tako je jedan od dodataka *Visual Studija* podrška za rad s *Unityjem*, stoga se ga odmah i instaliralo.

Vrijedi spomenuti jedan koristan dodatak *Visual Studija* koji je prisutan u verziji 2022, a to je *IntelliCode*. *IntelliCode* radi na način da *developerima* nudi prijedloge što da sljedeće napišu usred njihovog pisanja. Svoje prijedloge daje na temelju dosad napisanog koda u nekoj klasi ili ih dohvaća iz raznih projekata objavljenih na *GitHubu* [32]. Stoga dok *IntelliCode* ponudi neku liniju koda, rijetko kada ponudi nešto što potpuno nema smisla, zbog čega se i smatra boljim i inteligentnijim od *IntelliSensea* koji na sličnom principu radi [33]. Zna davati rješenja za nešto i prije negoli *developer* počinje o rješenju razmišljati. Jednostavan primjer *IntelliCodea* može se vidjeti na sljedećoj slici (Slika 3.):



Slika 3. Primjer prijedloga *IntelliCodea* [autorski rad]

5.3. Firebase

U ovom radu će se morati koristiti i baza podataka u koju će se pohranjivati podaci o igračima. Zato je potrebno imati složen i spreman server, a u ovom je slučaju za server odabran *Firebase*.

Firebase je tzv. „Backend as a service*"* platforma. Drugim riječima, to je servis koji nudi različite proizvode i usluge za korištenje u razvoju *web* i mobilnih aplikacija [34] [35]. Prvo ju je razvila istoimena tvrtka 2011. godine, no 2014. ju je kupio Google [36]. Neke od stvari što *Firebase* nudi su [35]:

- ***Realtime Database*** – baza podataka koja sve akcije i zadatke radi u stvarnom vremenu. Podaci se spremaju u obliku JSON-a.
- ***Cloud Firestore*** – NoSQL baza podataka u oblaku (engl. *cloud*). Često se koristi za sinkroniziranje podataka kod aplikacija koje obrađuju podatke u stvarnom vremenu.
- ***Cloud Functions*** – okvir (engl. *framework*) koji nije potkrijepljen serverom. Koriste ga *developeri* da pokrenu pozadinski kod koji će odgovarati HTTPS komponentama i *Firebase* zahtjevima.

Za potrebe ovog rada bio je korišten *Realtime Database*, jer iako je *Cloud Firestore* generalno bolje rješenje s obzirom da ne radi s relacijskim bazama podataka (zbog čega i stoji prefiks *No* u NoSQL nazivu, kao Ne SQL) te ga je jednostavno koristiti, u bazu će se spremati za svakog glavnog neprijatelja samo ime igrača i njegovo najbrže vrijeme u pobjeđivanju tog glavnog neprijatelja, za što je sasvim dovoljan *Realtime Database*. Da bi se spremale vrijednosti za više različitih atributa, tada bi se odabrao *Cloud Firestore*.

5.4. Pojmovi unutar Unitya

Za završni rad autora ovog diplomskog rada također se razvijala video igra koristeći *Unity* te su se tamo opisali glavni pojmovi kako bi se rad mogao lakše pratiti i razumjeti [37]. Potrebno je i ovoga puta objasniti pojmove koji će se često puta spominjati tijekom rada.

5.4.1. GameObject

Glavni objekti oko kojih velika većina stvari ovisi su poznati pod nazivom „GameObjects". Oni sami po sebi ne rade ništa, no važni su jer im se mogu dodijeliti komponente [38].

5.4.2. GameObject dijete i GameObject roditelj

Unutar *Unity* alata obično se može vidjeti s lijeve strane veliki prozor „Hijerarhija“ koja prikazuje *gameobjecte*. Tamo (ili preko skripte) je moguće preko načina povuci i ispusti (engl. *drag and drop*) podrediti *gameobject* nekom drugom *gameobjectu*, pri čemu taj podređeni postaje dijete (engl. *child*), dok nadređen postaje roditelj (engl. *parent*). To je jedan od načina kako se može jednostavnije dohvatiti komponenta roditelja unutar skripte djeteta.

5.4.3. Components

Komponente su te koje se dodaju *gameobjectima* i implementiraju stvarnu funkcionalnost. Najčešća komponenta koja se koristi unutar *Unityja* i stvori se svaki put kada se kreira novi *gameobject* jest „Transform“ komponenta. Unutar nje može se definirati pozicija, rotacija i veličina *gameobjecta* [39]. Primjeri ostalih komponenata su glavna kamera, *rigidbody* koji služi za dodavanje zakona fizike *gameobjectu*, *collideri*, skripta, vizualni i audio efekti itd.

5.4.4. Collider

Kao što piše unutar *Unity* dokumentacije, „Collider“ komponenta je ta koja definira oblik *gameobjecta* u svrhu izvođenja fizičkih kolizija [40]. Oni služe kako bi dva *gameobjecta* registrirala jedan drugog ukoliko se njihovi dodijeljeni *collideri* dotaknu, pri čemu se dogodi ona akcija koja je napisana u skripti. Primjer za to je oduzmi životne bodove igraču koji je sa svojim *colliderom* dotaknuo *collider* neprijatelja. Postoje 2D i 3D *collideri*, no koristit će se samo 2D *collideri* od kojih su najpoznatiji:

- *Polygon Collider* 2D – *collider* koji se sastoji od raznih spojenih točaka i crta koji tako zajedno čine zatvoreni oblik
- *Edge Collider* 2D – *collider* u obliku jedne crte na kojoj je moguće stvoriti razne točke i tako mijenjati smjerove i duljinu crte između tih točaka
- *Box Collider* 2D – *collider* u obliku kvadrata/pravokutnika
- *Circle Collider* 2D – *collider* u obliku kruga
- *Capsule Collider* 2D – *collider* u obliku valjka

5.4.5. Prefabs

Vrlo često je unutar videoigara potrebno koristiti jedan te isti *gameobject* za dupliciranje jer se primjerice želi više neprijatelja istoga tipa kreirati i smjestiti po različitim dijelovima svijeta videoigre. No, da bi se to postiglo, morao bi se cijelo vrijeme duplicirati taj prvi *gameobject* manualno pomoću „Copy“ naredbe. S druge strane, postoji učinkovitije rješenje za to, a to je da se taj prvi *gameobject* koji predstavlja tip neprijatelja pretvori u „prefab“ koji se tada unutar

skripte može koristiti za stvaranje (instanciranje) *gameobjecta* neprijatelja istih vrijednosti i postavki kao i kod originalnog *gameobjecta* [41]. Može se reći da originalni *gameobject* služi kao predložak za sve ostale *gameobjecte*. Isto tako, razlog zašto su prefabovi vrlo korisni jest to što se mogu koristiti kod svih drugih scena projekta.

6. Igrivost i glavne ideje videoigre – „Ruined Essence of the Divine“

Izrađena *videoigra* zove se *Ruined Essence of the Divine*, što bi prevedeno na hrvatski značilo *Slomljena Srž Božanstvenih*. Priča videoigre jest da postoje božanstvena bića koja su po svojoj srži dobronamjerna, no kako je čovječanstvo njihovo zrcalo i kako se čovjek s vremenom sve više u tom svijetu fantazije iskvario, tako su se i oni iskvarili, pritom uzrokujući veliku neravnotežu svemira s obzirom na to da gospodare velikim moćima. Stoga je na dvojici glavnim protagonistima palatinima (herojskim vitezovima), koji su unatoč svemu ostali čestiti, da zaustave ta bića prije nego u potpunosti svojom prisutnošću unište cijeli svemir.

Igrač kontrolira spomenutog glavnog lika palatina (engl. *paladin*), dok drugi igrač također kontrolira palatina druge boje. Palatini se brzo kreću te mogu jednom skočiti sa zemlje, dok drugi put mogu još jednom u zraku skočiti. Isto tako, palatini mogu skakati sa zidova i na taj način se uspinjati po vertikalnim zidovima. Velika prednost palatina u borbi protiv glavnih neprijatelja jest njihovo brzo kotrljanje pomoću kojega su besmrtni te im gotovo ništa ne može naškoditi, dok njihov štit odbija većinu projektila, uzvraćajući tako projektil neprijatelju koji ga je ispalio i oduzimajući mu dio života. Svejedno, svaki igrač mora paziti na svoja 4 životna boda, 4 srca koja mogu vidjeti u gornjem lijevom i u gornjem desnom dijelu ekrana, zajedno sa svojom energijom prikazanom zelenom bojom koja se svaki puta potroši kada se štite ili kotrljaju. Kotrljanje oduzima više energije negoli štićenje, a ako je energije premalo ili je nema, ne mogu se štititi niti kotrljati. U tom slučaju mora proći jedno kratko vrijeme da im se energija ponovno vrati.

Kad pokrenu videoigru, igrači vide naslovnu scenu. Prvi igrač u početku mora napisati svoje korisničko ime koje se lokalno spremi u računalo. Nakon toga mogu početi igrati videoigru ili pogledati najbrža vremena za pobjeđivanje glavnih neprijatelja koje su postavili ostali igrači. Preko naslovne stranice mogu započeti igrati videoigru koja ih stavlja u mali svijet koji se sastoji od vrata u koja mogu ući i boriti se protiv glavnih neprijatelja. U početku su 4 od 5 vrata otključana, a nakon što pobjede sva 4 glavna neprijatelja, otključaju se vrata za posljednjeg, najtežeg neprijatelja cijele videoigre.

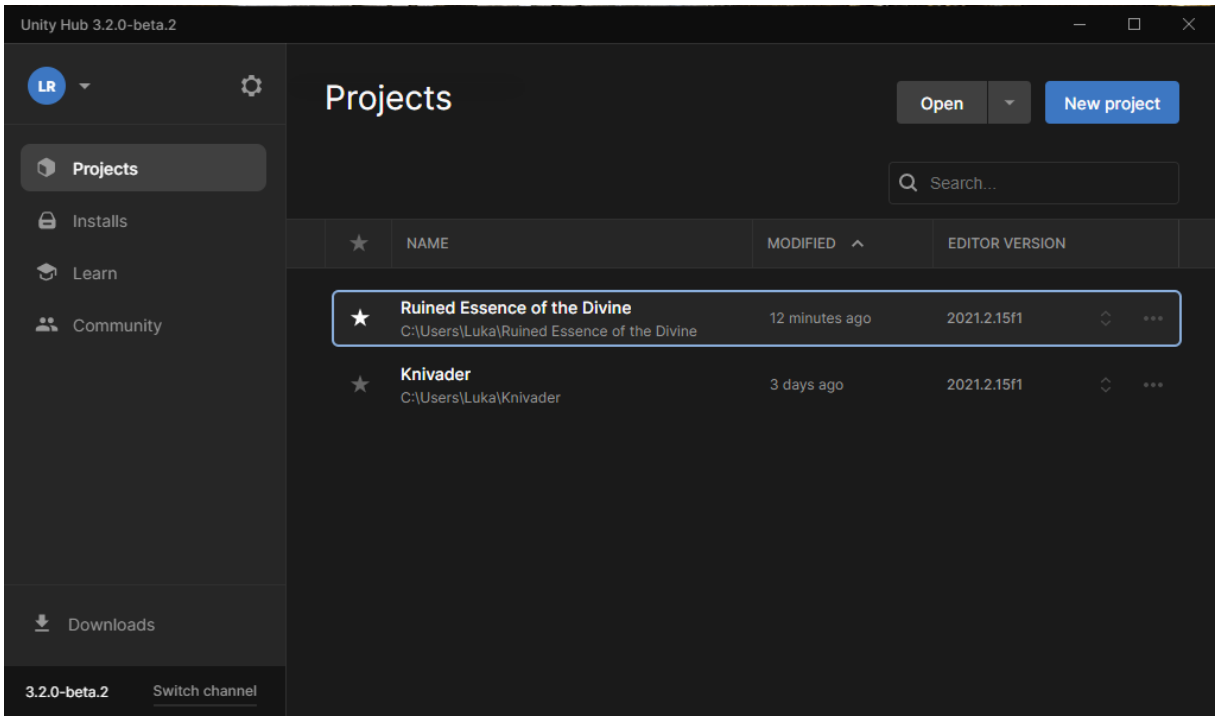
Svaki glavni neprijatelj je jedinstven te igrači moraju razviti najbolju strategiju kada napasti, a kada ne. Moraju naučiti koje sve napade glavni neprijatelji koriste kako bi ih mogli na kraju krajeva i pobijediti. Ono što će igrači primijetiti jest da prvi glavni neprijatelj nije toliko inteligentan, no svaki drugi će predstavljati sve veći izazov i biti složeniji. Kada igrači pobjede posljednjeg glavnog neprijatelja, igra je završena.

7. Izrada videoigre

Slijedi glavni dio rada, a to je izrada videoigre koristeći razne programske alate. Reći će se prvo o početnim procesima prilikom izrade, onda će se rad osvrnuti na programske kodove glavnog protagonista, glavne neprijatelje i na ostale važne skripte. Nakon toga će se govoriti o spremanju i dohvatu podataka iz baze, procesima animacije, dodavanju zvuka i na kraju o scenama unutar *Unityja*.

7.1. Priprema projekta

Prvo se otvorio *Unity Hub* verzija 3.2.0-beta.2 u kojemu se mogu vidjeti svi projekti koje razvija ulogirani korisnik (Slika 4.). Klikne se na „New project“ te se za ovaj slučaj odabrao 2D projekt zajedno s kamerom. Tada se otvorio prozor *Unity enginea* verzije 2021.2.15f1.

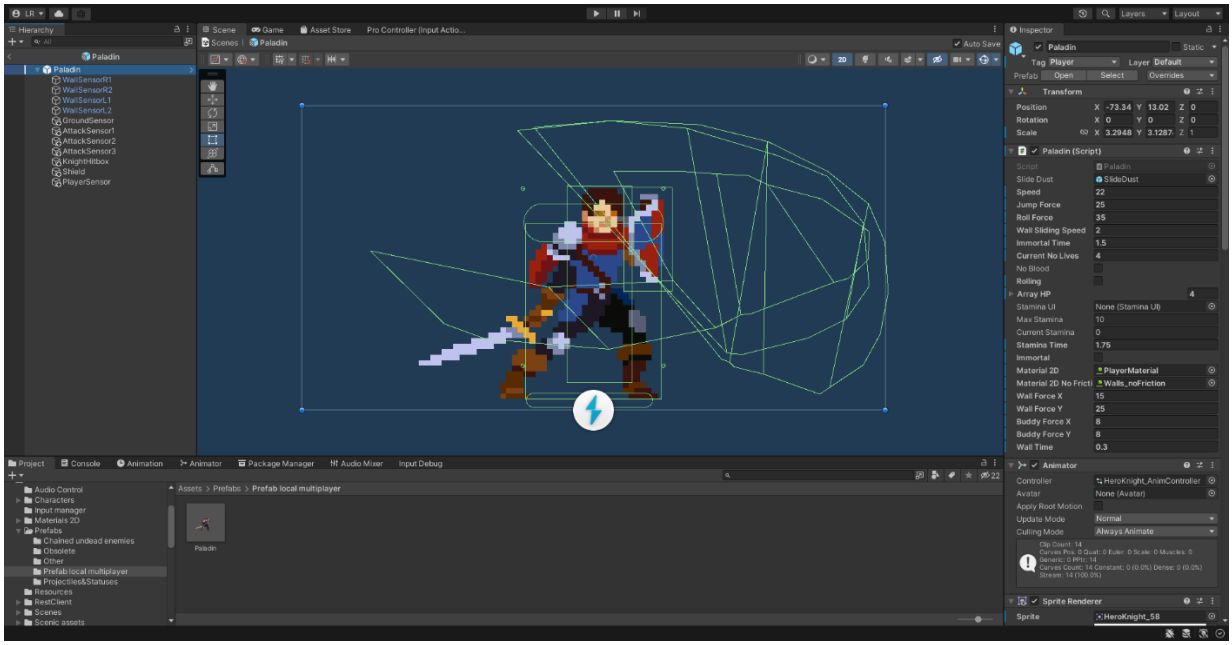


Slika 4. Izgled *Unity Huba* (verzija 3.2.0-beta.2) [autorski rad]

U početku jedini *gameobject* koji se kreirao jest glavna kamera, stoga je bilo potrebno dodavati nove *gameobjecte* s raznim komponentama, ali za to je prvo potrebno pribaviti slike za *gameobjecte*, zbog čega se otišlo na *web*-stranicu *Unity Asset Storea*. Prije se moglo tamo pristupiti preko prozora unutar samog *Unityja*, no ukinuli su mogućnost korištenja tog prozora zbog toga što je previše usporavao izvođenje procesa samog *enginea*. Preuzele su se različite slike i animacije neprijatelja, krajolika i efekata, dok se za glavnog protagonista preuzela i

skripta. Nakon što su se preuzele slike i uvezle u projekt, izradila se testna soba za testiranje mehanika palatina. Za to je bilo potrebno postaviti 2D *edge collidere* na tlo po kojemu palatini mogu hodati. Dapače, moralo se i njima dodati 2D (*box*) *collidere* kako bi mogli hodati po *edge colliderima*.

Sljedeće što je bilo potrebno napraviti jest postaviti unutar *Unityja* glavni vanjski alat, odnosno novu verziju *Visual Studija*, za što je bilo potrebno otići na *Edit -> Preferences -> External tools* i odabrati *Microsoft Visual Studio* 2022. Tada se otvorio program *Visual Studio* i prva skripta u koju se išlo pisati programski kod jest postojeća skripta palatina iz *Asset Storea*, no bilo je nužno gotovo sve obrisati i ispočetka pisati jer skripta palatina nije odgovarala potrebama zamišljene videoigre te nije radila ispravno kada su se različite komponente i mehanike dodavale. Drugim riječima, mali dio originalnog koda je ostao, točnije onaj dio koji se odnosi na senzore viteza na zidove i tlo, kao i kod za pokretanje određenih animacija, iako je velik dio i toga obrisan i izmijenjen. Zato će se u idućem poglavlju pobliže preko programskog koda objasniti koje su se sve mehanike razvile za palatine, a koje za glavne neprijatelje i za ostale elemente videoigre.



Slika 5. Izgled Unity sučelja koji prikazuje palatina sa svim svojim colliderima [autorski rad]

7.2. Skripte palatina

Programski kod palatina će se najjednostavnije objasniti na način da se sve njegove glavne akcije podijeli na potpoglavlja. Tako će se govoriti o kretanju palatina lijevo-desno, onda o skakanju i skakanju po zidovima, napadanju, štićenju, kotrljanju i o ostalim mehanizmima. Glavna skripta, odnosno klasa koja sadrži većinu metoda koje realiziraju sve navedeno zove

se „Paladin.cs.“ Neke metode koje nisu prisutne samo kod skripte palatina, nego su općenito tipične za sve klase koje deriviraju, odnosno nasljeđuju svojstva iz bazne klase „MonoBehaviour“ [42] su:

- **Start()** koja se pokrene izvršavati prije prvog okvira (engl. *frame*), odnosno prije prvog ažuriranja ekrana videoigre [43]
- **Awake()** koja se poziva kada se skripta prvi put učita i prije Start() metode. Koristi se kada se želi inicijalizirati varijable prije nego se aplikacija pokrene [44]
- **Update()** koja se za svaki *frame* poziva i koristi za mjerenje vremena ili ispitivanje nečega tijekom igre (npr. je li dovoljno vremena prošlo da igrač može drugi put mačem zamahnuti) [45]
- **FixedUpdate()** koji najviše služi za implementiranje zakona fizike *gameobjectu* zajedno s *rigidbodyjem* [46]

Posljednje što vrijedi spomenuti jest da biblioteke koje Paladin.cs (kao i većina ostalih) koristi su „System“ i „UnityEngine“, dok se biblioteka „Cinemachine“ koja sadrži metode vezane uz kameru koristi samo kod palatin skripte jer većinu vremena kamera prati samo igrače [47]. Imajući na umu sve rečeno, može se krenuti na opis kreiranih metoda za palatine.

7.2.1. Kretanje palatina po X-osi

Palatinu su se dodijelili mnogi *gameobjecti* s *colliderima* koji služe kako za detektiranje neprijatelja i projektila poput strijela, tako i za detektiranje zidova i tla. *Gameobject* pod nazivom „GroundSensor“ smješten ispod nogu palatina govori nalazi li se palatin na tlu, što je vrlo važan podatak jer se tada zna koje animacije se moraju pokrenuti te može li još jednom skočiti u zrak. Što se tiče samog kretanja lijevo i desno, izrađena je metoda „PaladinMovement()“ koju FixedUpdate() kontinuirano poziva:

```
[SerializeField] float m_speed;
/// <summary>
/// FixedUpdate - useful when dealing with physics
/// </summary>
private void FixedUpdate()
{
  float inputX = moveInput.x;
  PaladinMovement(inputX);
}

/// <summary>
/// Movement based on rigidbody velocity. Also checks if paladin is
/// rolling or blocking
/// </summary>
/// <param name="inputX">Input direction from player</param>
private void PaladinMovement(float inputX)
{
```

```
        // Move
        if (!m_rolling
            && !m_isBlocking
            && !isWallJumping)
        {
            m_body2d.velocity = new Vector2(inputX * m_speed,
               _body2d.velocity.y);
        }

        // If blocking, then paladin can't move
        if (m_isBlocking)
        {
            m_body2d.velocity = new Vector2(0, m_body2d.velocity.y);
        }
    }
```

Prvo što se vidi unutar FixedUpdate() metode jest to da se kreirala lokalna varijabla „inputX“ tipa „float“ u koju se stavila vrijednost moveInput.x, što je varijabla tipa Vector2 te se uzima samo vrijednost za X-os. Inače se moveInput-u cijelo vrijeme dodjeljuje nova vrijednost na temelju tipke koju igrač pritisne i/ili drži za kretanje lijevo ili desno, no detaljno će se o tome kasnije govoriti. Nadalje, poziva se metoda „PaladinMovement()“ te joj se prosljeđuje kao argument inputX.

Metoda u početku ispituje je li palatin trenutno u pokretu kotrljanja, koristi li štit te skače li po zidovima. Ako su vrijednosti svih varijabli unutar „if“ selekcije „false“, odnosno da nisu istinite, tada se postavlja vrijednost brzine rigidbody2d, odnosno „m_body2d.velocity“ da je jednak novom 2D vektoru („new Vector2()“). U tom vektoru brzina po X-osi ne ovisi samo o inputX-u, nego i o varijabli m_speed čija se vrijednost definira unutar samog *Unityja*, što se postiže na način da se za tu *float* varijablu u početku postavi „[SerializeField]“, što *Unityju* govori da takve privatne varijable također mora serijalizirati, odnosno preoblikovati na način da te podatke može lokalno u računalo pohraniti te bilo kada kasnije rekreirati. Takav proces je inače tipičan dok su varijable deklarirane kao *public*, odnosno dok se radi o javnim varijablama [48] koje druge klase mogu vidjeti i koristiti.
Ono što se također u toj metodi provjerava jest koristi li palatin štit. Ako je odgovor da, ne smije se kretati po X-osi, dok po Y-osi smije samo padati, pri čemu pad realizira gravitacija *rigidbodyja*.

7.2.2. Kretanje palatina po Y-osi

Palatin se po Y-osi kreće na temelju padanja, skakanja i skakanja po zidu. Prvo će se objasniti obično skakanje s tla i u zraku, a onda skakanje po zidu.

Za skakanje po zidu prvo se kreirala metoda koja sadrži mnogo provjera o tome smije li palatin skočiti. Ona se svaki puta poziva kada se pritisne odgovarajuća tipka za skakanje. Slijedi dio metode koji se odnosi na obično skakanje iz tla ili u zraku:

```
/// <summary>
/// Method based on jump movement.
/// Checks if paladin is allowed to jump.
/// </summary>
public void JumpAction()
{
    // Ordinary jump
    if (!m_isBlocking
        && m_noJumps > 0
        && !m_rolling
        && !m_isWallSliding
        && Time.timeScale != 0f)
    {
        SoundAnimationJump();

        m_grounded = false;
              m_body2d.velocity = new Vector2(m_body2d.velocity.x,
              m_jumpForce);
        m_noJumps--;
    }
}
```

Kako bi palatin mogao skočiti, mora se zadovoljiti nekoliko uvjeta, a to su redom da ne koristi štit, trenutačni broj skokova mu mora biti veći od nule, ne smije se kotrljati, ne smije skakati po zidovima te „Time.timescale“ mora biti veći od nule. Od spomenutog vrijedi reći o Time.timescale-u, što je zapravo mjera koja označava brzinu izvođenja videoigre, da ako je timeScale 1.0, vrijeme prolazi u videoigri kao i u stvarnom vremenu, no dok je 0.5, vrijeme prolazi dvostruko sporije. Koristi se i tijekom zaustavljanja (engl. *pause*) videoigre, pri čemu se zaustavlja izvođenje funkcija koje su ovisne o okvirnoj stopi (engl. *frame rate*).

Ako je sve zadovoljeno, pokreće se metoda „SoundAnimationJump()“ koja pokrene zvuk i animaciju skakanja. Ono što je važno jest da se postavi vrijednost varijable „m_grounded“ koja je tipa *bool* na vrijednost *false*, što tada znači da palatin više ne stoji na tlu. Postavi se i nova brzina *rigidbodyja*, pri čemu brzina po X-osi ostaje ista, dok se brzina po Y-osi promijenila za onu vrijednost koja je dodijeljena varijabli „m_jumpForce“.

Što se tiče skakanja po zidu, potrebno je podsjetiti se da palatin sadrži razne senzore, od kojih 4 senzora služi za detektiranje zida. Oni su smješteni na rubnim dijelovima pravokutnika *gameobjecta* palatina te je svakom senzoru dodijeljena skripta „SensorPaladin.cs“. Tamo je potrebno spomenuti par metoda:

```
/// <summary>
/// Add number by one if sensor detects collider with tag Ground
/// </summary>
/// <param name="other">Collider from another gameobject</param>
void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag("Ground"))
    {
```

```
            m_ColCount++;
        }
    }

    /// <summary>
    /// Get current state from the sensor
    /// </summary>
    /// <returns>True if sensor is detecting ground</returns>
    public bool State()
    {
        return m_ColCount > 0;
    }
```

„OnTriggerEnter2D(Collider2D)“ je metoda koja se poziva svaki put kada *gameobject*, kojemu je dodijeljena skripta s tom metodom, dolazi sa svojim *colliderom* u dodir s *colliderom* nekog drugog *gameobjecta* [49]. U ovom se slučaju u toj metodi ispituje je li *collider* s kojim se došlo u dodir označen kao „Ground“. Ako je odgovor da, tada se povećava vrijednost varijable „m_ColCount“ za 1, no kod izlaska iz takve kolizije se kod metode „OnTriggerExit(Collider2D)“ smanjuje m_ColCount za 1. Ako se želi ispitati trenutno stanje senzora, poziva se „State()“ koja vraća true ako je m_ColCount > 0, tj. ako senzor detektira tlo, u protivnom vraća *false*.

Sada kada je rečeno što se točno nalazi u skripti senzora, potrebno je vratiti se na glavnu skriptu, Paladin.cs. Za skakanje po zidu ponovno se poziva ista metoda kao i kod običnog skakanja, JumpAction(), no ovoga puta provjeravaju se neki drugi uvjeti. Te provjere ispituju vrijednost varijable „m_isWallSliding“, koja se unutar metode „CheckIfWallSliding()“ postavlja na *true* ako se palatin kliže po zidu, dok *false* postavlja ako to nije slučaj. Slijedi isječak koda iz metode CheckIfWallSliding():

```
    /// <summary>
    /// Set m_isWallSliding to true if both wall sensors on one side detect
    /// wall and set bool for animation.
    /// Change velocity of paladin if player is holding a button towards
    /// wall.
    /// </summary>
    /// <param name="inputX">Input direction from player</param>
    private void CheckIfWallSliding(float inputX)
    {
        if ((m_wallSensorR1.State()
                && m_wallSensorR2.State()
                && !gameObject.GetComponent<SpriteRenderer>().flipX)
            || (m_wallSensorL1.State()
                && m_wallSensorL2.State()
                && gameObject.GetComponent<SpriteRenderer>().flipX))
        {
            m_isWallSliding = true;
            m_animator.SetBool("WallSlide", m_isWallSliding);
        }
        else
        {
            m_isWallSliding = false;
            m_animator.SetBool("WallSlide", m_isWallSliding);
```

```
        }

        if ((m_wallSensorR1.State()
                && m_wallSensorR2.State()
                && inputX > 0)
            || m_wallSensorL1.State()
            && m_wallSensorL2.State()
            && inputX < 0
            && !m_grounded)
        {
            m_noJumps = 0;
            AE_SlideDust();
            m_body2d.velocity = new Vector2(
                m_body2d.velocity.x,
                Mathf.Clamp(
                   m_body2d.velocity.y,
                    -m_wallSlidingSpeed,
                    float.MaxValue));
        }
    }
```

Metodu „CheckIfWallSliding()“ poziva Update() te se u njoj provjeravaju stanja senzora. Ukoliko „State()“ metode jednog i drugog senzora s lijeve ili desne strane vraćaju *true*, znači da se pokraj palatina nalazi zid. Bilo je potrebno napraviti dodatnu provjeru, a to je da se provjeri prema kojem zidu je palatin usmjeren, u protivnom bi mogao skakati po zidu koji se nalazi iza njega, dok je željeno ponašanje videoigre da skače samo po zidu koji se ispred njega nalazi. Ako je uvjet zadovoljen, postavlja se m_isWallSliding na *true*, što znači da se kliže po zidu. Isto tako se odmah i za animaciju naredbom „SetBool()“ postavi varijabla WallSlide na *true*, što okine animaciju za klizanje po zidu.

Druga *if* selekcija također ispituje detektiranje zida, no ovoga puta se još ispituje i držanje gumba za kretanje prema tom zidu (inputX) te nalazi li se palatin u zraku. Ako je to slučaj, postavlja se vrijednost skakanja na 0, što onemogućuje dodatan skok nakon skoka sa zida, poziva metodu „AE_SlideDust()“ za kreiranje efekta prašine ispred igrača te smanjuje brzinu padanja palatina po zidu. Kod tog postavljanja brzine klizanja po zidu koristi se „Mathf.Clamp()“, unutar koje se postavila vrijednost brzine po Y-osi i definirale su se vrijednosti minimalne i maksimalne brzine klizanja po zidu, a u slučaju da bi brzina *rigidbodyja* po Y-osi bila manja od minimalne brzine koju se definiralo unutar *Unityja*, tada bi se ta brzina vratila na tu minimalnu brzinu. Isto vrijedi i za prekoračenje maksimalne brzine, pri čemu se u tom slučaju vraća na maksimalnu brzinu, no vjerojatnost da se to dogodi je mala jer *float* Max.Value označava golem broj [50].

7.2.3. Napad palatina

Napad palatina mačem postiže se na način da se prilikom klika na određenu tipku privremeno aktivira 2D *polygon collider* čiji oblik odgovara sličicama animacije za zamah

mačem. Palatin može napraviti tri različita poteza mačem, od kojih se prvi pokrene klikom na gumb za napad, dok se drugi i treći pokrenu ako se nedugo nakon prijašnjeg klika ponovno klikne na gumb za napad, u protivnom se pokrene animacija i 2D *polygon collider* za prvi napad. Metoda koja je odgovorna za pokretanje napada mačem zove se „AttackAction()“:

```
/// <summary>
/// Method for activating attack collider and animation.
/// There are three attack combos.
/// Loops back to the first combo after the third combo attack
/// or if the measured interval of time since the last attack is too
/// large.
/// </summary>
private void AttackAction()
{
    if (!m_isBlocking
        && m_timeSinceAttack > 0.25f
        && !m_rolling
        && !m_isWallSliding)
    {
        m_currentAttack++;

        if (m_currentAttack > 3
           || m_timeSinceAttack > 1.0f)
        {
            m_currentAttack = 1;
        }

        m_animator.SetTrigger("Attack" + m_currentAttack);
        SingletonSFX.Instance.PlaySFX("SFX" + (6 + m_currentAttack)
           .ToString() + "_sword_combo_" + m_currentAttack);

        StartCoroutine(CanAttackAgain(m_currentAttack));

        m_timeSinceAttack = 0.0f;
    }
}
```

Kao i kod prijašnjih opisanih metoda, ovdje je također potrebno imati na početku provjere, u ovom slučaju za napad. Palatin ne smije koristiti štit, mora proći kratko vrijeme nakon posljednjeg napada mačem (bar četvrtina sekunde), ne smije se kotrljati niti skakati po zidovima. Kada je sve zadovoljeno, povećava se vrijednost varijable „m_currentAttack“ (trenutni napad) za 1, koji je inače u svim ostalim slučajevima jednak 0. Nadalje, provjerava se nije li m_currentAttack veći od 3, odnosno nije li treći potez mačem već napravljen te nije li previše vremena prošlo nakon posljednjeg napada. Ako je jedan od navedenih uvjeta zadovoljen, m_currentAttacku se postavlja vrijednost na 1. Postavi se okidač (engl. *trigger*) koristeći naredbu „SetTrigger()“ te se time pokrene ona animacija koja odgovara vrijednosti varijable m_currentAttack. Isto tako se odsvira onaj zvučni efekt koji odgovara trenutnom potezu mača, za što se koristi izrađena klasa „SingletonSFX.cs“ na koju se primijenio uzorak dizajna (engl. *design pattern*) singleton, no o uzorcima dizajna će se nešto kasnije govoriti.

Nakon toga se poziva metoda „CanAttackAgain()“ pomoću „StartCoroutine()“. StartCoroutine() se često koristi unutar ovog projekta, najviše kod skripata neprijatelja jer se pomoću nje može izvođenje neke akcije zaustaviti tako dugo dok se određeni uvjet ne zadovolji [51]. Metoda koja se pomoću StartCoroutine() zove mora bit tipa IEnumerator. Slijedi kod metode CanAttackAgain():

```
/// <summary>
/// Method which sets certain 2D polygon collider active for a short period
/// of time based on attack combo
/// </summary>
/// <param name="attackNo">Current number from 3 different attacks in a
/// row</param>
/// <returns></returns>
private IEnumerator CanAttackAgain(int attackNo)
{
    attacking = true;
    float timeForDisable = 0.15f;

    if (attackNo == 3)
    {
        timeForDisable = 0.25f;
    }

    GameObject child = transform.GetChild(5 + attackNo).gameObject;
    yield return new WaitForSeconds(0.1f);

    child.SetActive(true);
    yield return new WaitForSeconds(timeForDisable);

    child.SetActive(false);
    attacking = false;
}
```

Postavi se vrijednost varijable „attacking“ na *true* te se postavi vrijednost varijable „timeForDisable“ na 0.15f, odnosno na četvrtinu sekunde ako se radi o trećem napadu. Tada skripta koja je dodijeljena glavnom *gameobjectu* palatina pomoću naredbe „transform.GetChild()“ dohvaća ono dijete čiji *collider* odgovara trenutnoj animaciji za napad. Zatim se izvrši naredba „yield return new WaitForSeconds()“ koja zaustavlja daljnje izvođenje te metode tako dugo dok ne prođe vrijeme koje se definiralo u „WaitForSeconds()“, u ovom slučaju desetina sekunde. Nakon što prođe kratko vrijeme, dijete koje sadrži u sebi 2D *polygon collider* se aktivira te svakom neprijatelju koji dodirne taj *collider* oduzima se dio životnih bodova ukoliko nije besmrtan. No, to dijete ne ostane dugo aktivno, nego se ponovno deaktivira nakon što prođe vrijeme koje se dodijelilo varijabli „timeForDisable“, čija vrijednost ovisi o duljini pojedine animacije za napad. Na kraju metode se *attacking* postavlja na *false*, dok se kod prijašnje metode AttackAction()vrijednost varijable „m_timeSinceAttack“ postavlja na 0, što znači da ponovno mora proći kratko vrijeme kako bi palatin mogao napasti. Samoj varijabli m_timeSinceAttack vrijednosti se dodaju u obliku Time.deltaTime unutar Update()

metode. Time.deltaTime je zapravo interval između prijašnjeg okvira i trenutnog koji se izvodi izražen u sekundama [52].

7.2.4. Obrana palatina

Palatin uz mač ima i magičan štit koji odbija većinu projektila. S obzirom na to da on i brzo trči te ne postoji vrijeme ubrzanja nakon mirovanja nego odmah dostigne maksimalnu brzinu trčanja, realizirana je još i mehanika kotrljanja, čime u kratkom vremenu postane besmrtan i tako izbjegne sve napade neprijatelja. No, postoji ograničenje za korištenje štita i kotrljanja, a to je da za njih u trenutku izvedbe mora imati dovoljno snage (engl. *stamina*).

S obzirom na to da se štićenje i kotrljanje palatina u programskom kodu realiziralo ponovno putem različitih provjera te aktiviranjem i deaktiviranjem raznih *collidera*, dok će se kod za projektile koje štit odbija objasniti u poglavlju prvog glavnog neprijatelja, reći će se samo kako se postignulo ponašanje snage tijekom igre. Za nju su se također koristile razne provjere, no ipak ih je važno spomenuti kako bi se mogla odmah dobiti i ideja na koji način rade i spomenute dvije akcije obrane.

Palatinova snaga u videoigri izražena je brojevima od 0 do 10, pri čemu 0 označava da nema više snagu, dok 10 označava da je pri punoj snazi. Njegova snaga se smanji za 3 ukoliko koristi štit, dok se za 4 smanji ako se otkotrlja. Takva oduzimanja se događaju unutar metoda za te akcije, dok se obnova snage realizirala pomoću dvije metode: „CheckStamina()“ i „StaminaRegeneration()“:

```
/// <summary>
/// Method for counting time for stamina to be regenerated.
/// Starts coroutine for stamina regeneration.
/// </summary>
private void CheckStamina()
{
    if (!m_isBlocking
        && currentStaminaTime < staminaTime)
    {
        currentStaminaTime += Time.deltaTime;
    }

    if (!staminaMethodCalled
        && currentStamina < 10)
    {
        staminaMethodCalled = true;
        StartCoroutine(StaminaRegeneration());
    }
}


/// <summary>
/// Method for stamina regeneration. Starts after staminaTime while
/// stamina is below 10 units
/// </summary>
private IEnumerator StaminaRegeneration()
```

```
{
    while (currentStamina < 10)
    {
        yield return new WaitForSeconds(0.5f);

        if (currentStaminaTime > staminaTime)
        {
            currentStamina += 1;
        }
    }

    staminaMethodCalled = false;
}
```

Te dvije metode su usko povezane, stoga će ih se zajedno objasniti. Prva metoda, CheckStamina() provjerava je li štićenje još u tijeku izvođenja te je li currentStaminaTime manji od staminaTime, pri čemu staminaTime označava vrijeme koje je potrebno da prođe kako bi se snaga mogla obnoviti nakon što se pokrene jedna od akcija obrane koji postavljaju currentStaminaTime na 0. Ako se palatin ne štiti te je currentStaminaTime manji od staminaTime, dodaje se vrijeme currentStaminaTime. Nadalje, ako se metoda za obnovu snage, tj. StaminaRegeneration() nije pozvala te ako je trenutačna snaga manja od 10, potrebno je staviti na *true* da se metoda pozove i pozvati StaminaRegeneration().

U metodi StaminaRegeneration() se „while()“ petljom provjerava je li trenutačna snaga manja od 10 jer palatin usred obnove snage može koristiti jednu od akcija obrane i tako ponovno smanjiti ukupan broj snage koju je tada u tom istom izvođenju petlje potrebno obnoviti. Snaga se obnavlja za 1 nakon svakih pola sekunde, no prije nego se obnovi potrebno je preispitati je li došlo do akcije obrane. To se realiziralo na način da se preispita je li currentStaminaTime veći od staminaTime, jer ako nije, znači da je palatin koristio štit ili se kotrljao. Kada je snaga u potpunosti obnovljena, staminaMethodCalled se postavlja na *false*.

7.2.5. Ostali mehanizmi

Skripta palatina sadrži još mnogo ostalih metoda koje se izvršavaju kako bi se realizirale sve potrebne mehanike palatina. Tako je i oduzimanje života palatinu primjer za to, pri čemu se svaki put poziva metoda „DecreaseHealth()“ nakon što ga neprijatelj izravno ili neizravno dodirne projektilom. U tom slučaju pokrene se prikladna animacija za to, odsvira zvučni efekt te palatin postane jedno kratko vrijeme besmrtan kako bi mogao pobjeći neprijatelju.

Valja još spomenuti da se unutar Awake() metode inicijaliziraju i pripremaju različite stvari. Primjerice, kada se učita neka scena i kada se u toj sceni klikom na bilo koji gumb akcije stvori palatin, tada će glavna kamera početi pratiti njega, osim ako je u pitanju borba s četvrtim i petim glavnim neprijateljem. Naredba za kameru jest sljedeća:

```
GameObject.Find("VirtualCamera").GetComponent<CinemachineVirtualCamera>()
    .Follow = transform;
```

„GameObject.Find()“ metoda koristi se kako bi se pronašao *gameobject* unutar *Unityja* s nazivom koji je napisan unutar zagrada, u ovom slučaju „VirtualCamera“. Kada se pronađe takav *gameobject*, dohvaća se njegova komponenta pomoću GetComponent<>() i pritom se unutar <> zagrada napiše komponenta koja se želi dohvatiti. Nakon toga se dohvaćenoj kameri naredi da slijedi palatina dodjeljujući joj *transform* komponentu palatina. Nužno je reći da se GameObject.Find() rijetko kada koristi u cijelom projektu jer je dobra praksa da se izbjegava njegovo korištenje s obzirom na to da se često može dogoditi da taj *gameobject* koji se želi dohvatiti ne postoji zato što se njegovo ime krivo napisalo u skripti/*Unityju*, može doći do promjene njegovog naziva ili jednostavno više ne postoji.

Ostalo što skripta Paladin.cs sadrži neće se objasniti jer je ona velika te su se najbitnije stvari već spomenule. Jedino što će se još detaljnije objasniti u toj skripti jest korištenje „Input Actiona“, što u prijevodu znači „Akcije Unosa“.

7.3. Sistem unosa (engl. Input System)

Interakcija između igrača i videoigre postiže se na način da uređaji koje igrači koriste (tipkovnica, miš, kontroler, glas) šalju podatke određenom *engineu* koji ih prikuplja i dalje nešto radi s njima. U ovom slučaju *Unity* te podatke poput klika na gumb šalje *Input Systemu*. Trenutno *Unity* sadrži klasičan *Input System*, odnosno „UnityEngine.Input“ čije naredbe u skripti za dohvat klika gumba od strane igrača u principu izgledaju kao sljedeća reprezentativna naredba (dohvat tipke slova Y):

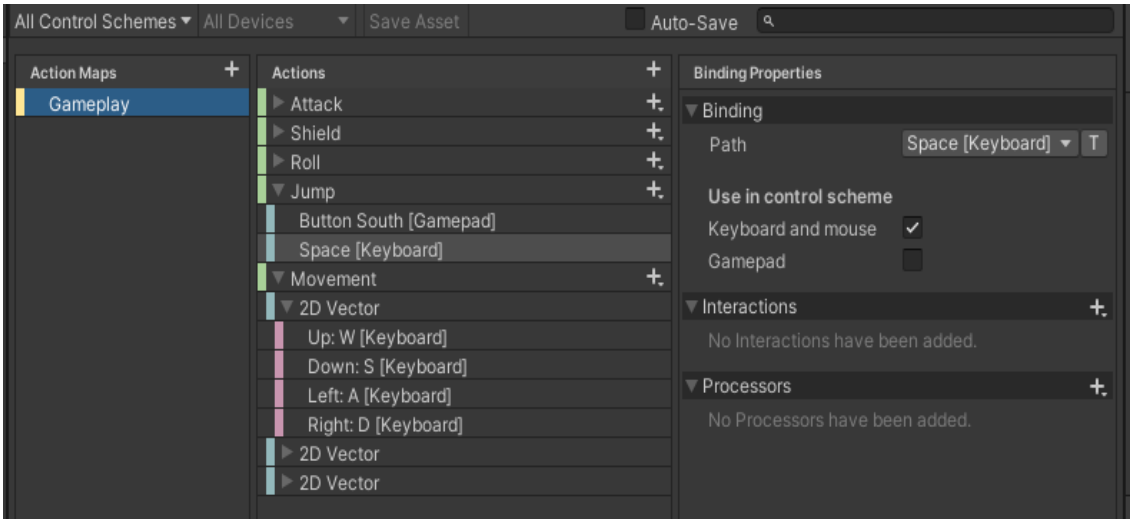
```
Input.GetKeyDown(KeyCode.Y)
```

Prva službena, stabilna verzija novog *Input Systema* je za javnost izašla 2019. godine, no *developeri* tada još nisu voljeli koristiti novi *Input System* jer je često neispravno radio i bio pun pogrešaka. U međuvremenu se novi *Input System* sa svakom novom verzijom uvelike poboljšao te se zbog toga odabrao za ovaj rad zbog svojih naprednih mogućnosti koje mnoge stvari olakšavaju [53].

Za korištenje novog *Input Systema* bilo je potrebno preuzeti njegovu najnoviju verziju i uvesti ga u *engine*. Kako bi se vidjelo koje sve uređaje spojene s računalom *Unity* prepoznaje s razlogom što će se kasnije tijekom testiranja moći vidjeti koji uređaji rade ispravno, a koji ne, potrebno je bilo otići na *Window -> Analysis -> Input Debugger*. Tada se unutar mape *Assets* gdje se nalaze sve skripte, slike, animacije, materijali, zvučni efekti i slično stvorila nova mapa pod nazivom *Input manager*. Unutar nje se desnim klikom na miš pod *Create* odabralo

kreiranje novog *Input Actiona*. Tada se otvori *Input Action* gdje se moralo kreirati dvije kontrolne sheme (engl. *control schemes*), jedan za kontrolere pod nazivom „Gamepad“, a drugi za miš i tipkovnicu pod nazivom „Keyboard and mouse“. Jedan i drugi će koristiti sve iste akcije, stoga je bilo potrebno kreirati samo jedan „Actions map“ koji se popunio sa svim akcijama koje palatin izvodi (napad mačem, štićenje, kretanje itd.) [54].

Za svaki *Action* bilo je potrebno dati naziv te kod Postavke Akcija (engl. *Action Properties*) definirati tip akcije (engl. *Action Type*) koji može biti: *Button* (gumb), *Value* (vrijednost) i *Pass Through* (prođi kroz). Primjerice, za akciju napadanja odabrao se tip akcije gumb jer je za napad potrebno samo kliknuti gumb, dok se za kretanje po X-osi odabrao *Value* kao tip akcije za koji se još morao definirati i tip kontrole (engl. *Control Type*) Vector2, jer će se palatin kretati po smjerovima lijevo-desno. Za svaku postavku akcije bilo je potrebno definirati i vezivnu postavku (engl. *binding properties*). Tamo se zapravo definira gumb koji će pokrenuti akciju. Slijedi slika (Slika 6.) koja prikazuje prozor *Input Actiona* pod imenom „Pro Controller“ te je u toj slici označena vezna postavka kojoj je postavljena tipka *Space* s tipkovnice i time će ona svakim pritiskom pokrenuti akciju „Jump“, odnosno skoči:



Slika 6. Input Action prozor [autorski rad]

Nakon što se definiraju svi gumbi i tipke za sve akcije, potrebno je bilo označiti *Pro Controller* kao *Input Action* i kliknuti na potvrdni okvir (engl. *checkbox*) „Generate C# Class“ kako bi se generirala C# klasa koja sadrži sve definirane postavke akcija. Nakon toga se kliknulo na gumb „Save Asset“ kako bi se kreirala skripta ili ažurirala ako već od prije postoji. Sljedeće što se moralo napraviti jest kreirati komponentu „Player Input“ unutar glavnog *gameobjecta* palatina i tamo smjestiti *Pro Controller Input Action*. Važno je tamo definirati „Behaviour“, odnosno ponašanje dodijeljenog *Input Actiona*, a za ovu se videoigru definiralo da se svaki put pozovu događaji definirani unutar *Unityja* (engl. *Invoke Unity events*).

Prije nego što se popuni lista tih *Unity* događaja, potrebno je vratiti se u Paladin.cs skriptu i tamo definirati metode za primanje akcije gumba. One nakon primanja dalje zovu odgovarajuću metodu akcije palatina. Kako bi bilo jasnije, slijedi isječak iz koda:

```
// Method for getting input from various
// devices and Player Input component -  Attack
public void AttackInput(InputAction.CallbackContext ctx)
{
    if (ctx.performed)
    {
        AttackAction();
    }
}
```

Metoda „AttackInput()“ ima definiran parametar tipa zapečaćene (engl. *sealed*) klase *InputAction*, gdje je *sealed* klasa takva klasa koju niti jedna druga klasa ne može naslijediti [55]. Taj parametar sadrži podatak o tomu je li se gumb kliknuo, što se provjerava u *if* selekciji s „ctx.performed“. Ako je to slučaj, potrebno je zvati metodu napadanja mačem. S druge strane vrijedi objasniti što ako se pod *Action Properties* definirao *Action Type* kao *Value*:

```
// Method for getting input from various devices
// and Player Input component - Movement
public void MovementInput(InputAction.CallbackContext ctx)
{
    if (ctx.performed)
    {
        moveInput = ctx.ReadValue<Vector2>() * movementDirection;
    }
    else if (ctx.canceled)
    {
        moveInput = Vector2.zero;
    }
}
```

Ovdje ukoliko je ctx.performed istinit, onda treba pročitati vrijednost tipa Vector2 iz koje se dalje može izvući vrijednost za kretanje po X-osi. Ona je jednaka 1 ukoliko se drži gumb za kretanje desno, -1 ako je za lijevo [56, 8:20]. Ona se množi s varijablom „movementDirection“ koja je u normalnom slučaju jednaka 1, no po redu drugi glavni neprijatelj ima moć preokretanja kontrole igrača. Unutar *else ifa* ispituje se je li kontekst prekinut, odnosno ctx.canceled. Ako je to slučaj, znači da je igrač prestao držati gumb za kretanje lijevo-desno i tada se postavljaju vrijednosti *moveInputa* na nule pomoću Vector2.zero.

Nakon što su metode za primanje InputAction.CallBackContext raspisane, potrebno je popuniti listu *Unity* događaja. Unutar *Player Input* komponente se pod *Events -> Gameplay* za svaku navedenu akciju iz prozora *Input Actions* definira da se metode nakon klika pozivaju

tijekom igranja videoigre, a ne tijekom igranja i uređivanja videoigre. Također se definira skripta i metodu iz te skripte koja će se pokrenuti nakon što se klikne određeni gumb.

Zadnje što je još preostalo za napraviti jest otići u Unityju na *Edit -> Project settings -> Player -> Other Settings -> Configuration* i tamo za „Active Input Handling“ postaviti da se koristi novi *Input System*. Ako se to ne napravi, *Unity* neće prepoznati unose od igrača i raditi odgovarajuće akcije poput hodanja, nego će očekivati da se sve realiziralo koristeći stari *Input system*.

Tako na kraju videoigra podržava igranje koristeći miš zajedno s tipkovnicom i kontroler, a za ovaj je rad od kontrolera upotrijebljen *Nintendo Switch Pro Controller*. Za njega se moralo još preuzeti program reWASD [57] kako bi ga *Unity* mogao prepoznati kao običnog kontrolera i raditi s njim bez poteškoća. Osim što videoigra podržava igranje preko navedenih uređaja, odmah se može izraditi i podrška za više igrača koristeći kreiran *Input Action*.

7.4. Podrška za više igrača

Videoigra je zamišljena da podržava igranje više igrača lokalno na istom računalu. To se postiglo na način da se kreirao „Player Input Manager“. Tamo je bilo nužno pod „Joining“, što znači pridruživanje, postaviti „Join Behaviour“ na “Join Players When Button Is Pressed“, što sve zajedno zapravo znači da se postavilo ponašanje pridruživanja na „Pridruži igrače kada je gumb pritisnut“ [56, 9:50].

Bilo je nužno preoblikovati glavnog *gameobjecta* palatina u prefab kako bi se ga dalje stavilo pod „Player Prefab“ u *Player Input Manager* jer na taj način ta komponenta zna koji *gameobject* treba svaki put stvoriti kada se pritisne gumb na onim uređajima koji su definirani pod *Input Actions* (miš i tipkovnica, kontroler). Također je važno staviti maksimalan broj igrača na 2.

Ono što *Player Input Manager* automatski radi jest da kada igrač pritisne tipku na tipkovnici, on će moći upravljati samo sa svojim palatinom preko tipkovnice i miša. Drugim riječima, igrač neće moći upravljati i drugim palatinom dok se drugi igrač pridruži u igru iako dijele isti *Input Actions* i skriptu.

8. Glavni neprijatelji

Glavni neprijatelji najvažniji su dio cijele videoigre. Sveukupno ih ima pet te svaki ima svoju vrstu napada, arenu, glazbu i zvučne efekte i sve što je potrebno kako bi se postiglo da su bitke s njima jedinstvene i da se time izbjegne monotonija tijekom igranja. Za svakog glavnog neprijatelja će se prvo objasniti kako izgleda scenarij bitke protiv njega, koje posebnosti ta bitka ima koje ju čine drugačijom od ostalih bitaka, koje napade koristi glavni neprijatelj te bitnije stvari prikazati u programskom kodu.

8.1. Prvi glavni neprijatelj – Fern Behemoth

Glavni neprijatelj kojega igrači prvoga susreću se zove *Fern Behemoth*, golemo zeleno čudovište koje na velikoj stijeni u crnom kanjonu bezumno bježi lijevo-desno. Kada mu se igrači prvi puta približe, pozdravi ih glasnim krikom te ih počne napadati.

Prvi glavni neprijatelj nije toliko inteligentan. On ne prati igrače na način da dođe do njih pa da im onda naškodi, nego bježi s jednog kraja stijene na drugi. Uvijek nakon što se na kraju stijene ispred provalije okrene, počinje nasumično koristiti jedan od svojih napada. Nakon što mu igrači ukupan broj životnih bodova (engl. *health points*, HP) smanje, dođe na sredinu arene, ponovno ispusti glasan urlik te prizva zelene vatre koje se postepeno i brzo stvaraju od mjesta gdje se on nalazi sve do oba kraja arene. Posljedica tog napada jest požar na drvima koje rastu na toj stijeni, što su dodatne prepreke za igrače koji tada moraju bolje paziti kuda se kretati.

Potrebno je nabrojati i objasniti sve vrste napada koje glavni neprijatelj Behemot koristi. Napadi su sljedeći:

1. bježanje s jednog kraja arene na drugi kraj koje igrači mogu izbjeći samo ako se kotrljaju
2. riganje nekoliko zelenih vatrenih kugla koje igrači također mogu izbjeći kotrljanjem, no mogu i svojim štitom odbiti te kugle natrag prema Behemotu i tako mu oduzeti više života negoli mu se oduzima mačem
3. hodanje te pritom ostavljanje vatre iza sebe na tlu, pri čemu negdje na sredini arene u kratkom vremenu ne ispusti vatru, što igrači mogu iskoristiti tako da se otkotrljaju na tom dijelu i da tamo budu tako dugo dok vatra oko njih ne nestane

Slika 7. Palatini bježe od Behemota [autorski rad]

Slika 8. Behemot riga vatru prema igraču s podignutim štitom tijekom druge faze borbe [autorski rad]

Sve navedene napade dodatno otežava drugi dio borbe zbog toga što tada posušena drveća gore. Samog Behemota u početku nije moguće preskočiti nego ga izbjeći kotrljanjem, no dok drveća gore to nije lako izvesti, stoga se Behemot svaki put veličinom umanji nakon što mu se oduzmu određeni brojevi života, što omogućuje igračima da ga tada mogu preskočiti. Behemot se prestaje smanjivati dok započne druga faza borbe. Isto tako, Behemot ima veći broj života ako ulazi u borbu protiv dva igrača, dok protiv jednog ima manje kako bi se videoigra uravnotežila što se tiče težine za proći.

Nakon što igrači mačem i odbijenim vatrenim kuglama smanje ukupan broj života Behemota na nulu ili manje, Behemot ispusti posljednji veliki krik te polako nestaje u zemlju. Vrijeme koje se mjerilo od početka bitke sprema se u bazu podataka zajedno s igračevim imenom koje je definirao u naslovnoj sceni.

Za Behemota je izrađena jedna skripta, „FernBehemoth.cs“ te će se ona u sljedećem poglavlju objasniti. Prije nje objasnit će se skripta „AbstractBoss.cs“ koju nasljeđuju skripte svih glavnih neprijatelja videoigre. Poslije toga slijedi objašnjenje skripte „Projectile.cs“ koja je dodijeljena svim *gameobjectima* koje palatin može svojim štitom odbiti.

8.1.1. Zajednička skripta glavnih neprijatelja

Klasa AbstractBoss.cs je, kao što stoji u samom nazivu, apstraktna. Prema Joyce Farrellu [58, str. 456], apstraktna klasa je ona iz koje se ne mogu izraditi konkretni objekti, nego se iz nje mogu naslijediti svojstva. Za to je potrebno staviti ključnu riječ (engl. *keyword*) „abstract“ ispred ključne riječi „class“ unutar skripte.

Unutar apstraktne klase definirale su se varijable koje imaju svi glavni neprijatelji videoigre, od kojih su dvije najvažnije životi, odnosno „health“ čije vrijednosti mogu biti samo cijeli brojevi (integer, int) i brzina kretanja glavnog neprijatelja koja se deklarirala pod nazivom „velocity“ tipa „float“ s razlogom što takvim tipom varijabla može primati decimalne brojeve. Postoje i druge varijable koje se deklariraju, kao što je gameobject „timerCountDown“ koji služi za pokretanje mjerenja vremena na početku igre i za zaustavljanje i spremanje u bazu podataka na kraju bitke.

Metode unutar apstraktne klase mogu biti označene kao „abstract“ ili kao „virtual“ te im ne smije biti dodijeljen modifikator pristupa (engl. *access modifier*) „private“, nego „protected“ ili „public“ kako bi mogle biti naslijeđene [58, str. 442]. One označene kao *abstract* nisu ispunjene kodom, nego potklasa, odnosno klasa koja nasljeđuje apstraktnu klasu, mora nadjačati tu metodu (engl. *override*) koristeći ključnu riječ „override“ i staviti u nju svoju implementaciju (kod) [58, str. 457]. Virtualne metode s druge strane nije potrebno nadjačati te se u njima može napisati željeni kod, no programerima je dana mogućnost da dijete svejedno može nadjačati tu metodu, svoju implementaciju staviti, ali se isto tako može i dalje koristiti kod iz bazne (apstraktne) klase koristeći riječ „base“ i uz nju staviti naziv metode [58, str. 442]. Sljedeći isječak iz klase AbstractBoss.cs prikazuje primjer jedne apstraktne metode, dok će se kasnije prikazati i primjer jedne virtualne:

```
/// <summary>
/// Decrease health from a boss
/// </summary>
/// <param name="layer">Layer 16 = minor damage, Layer 19 = major
```

```
/// damage</param>
public abstract void MinusHealth(int layer);
```

Kao što se vidi, metodi „MinusHealth()“dodala se ključna riječ *abstract* te ju je potrebno nadjačati. Trenutno je skripta FernBehemoth.cs prikladan primjer tog nadjačavanja, a odmah se može i objasniti što se događa u toj metodi. Stoga je dan kod istoimene metode:

```
/// <summary>
/// Decreases Fern Behemoth's health
/// </summary>
public override void MinusHealth(int layer)
{
    if (!invincible)
    {
        if (layer == 16)
        {
            health -= 2;
            SingletonSFX.Instance.PlaySFX("SFX15_blood_splash");
        }
        else if (layer == 19)
        {
            health -= 3;
            SingletonSFX.Instance.PlaySFX("SFX23_fire_burn");
            SingletonSFX.Instance.PlaySFX("SFX21_neo_ridley_second_scream
            ");
        }

        if (health <= 0)
        {
            minAttack = 0;
            maxAttack = 0;

            Death();
            return;
        }
        // Code goes on...
    }
}
```

Prvo što se može vidjeti jest spomenuta ključna riječ *override* nakon koje se metoda popunila kodom. U ovom slučaju ispituje se besmrtnost Behemota jer ukoliko više nije kratkoročno besmrtan nakon posljednjeg smanjenja života, dalje se provjerava što mu želi smanjiti život, odnosno ukoliko je *collider* glavnog neprijatelja došao u doticaj s *colliderom gameobjecta* kojemu je dodijeljen sloj (engl. *layer*) mača označen brojem 16, taj broj će se ovoj metodi proslijediti te će se glavnom neprijatelju smanjiti broj života za 2 i odsvirati zvučni efekt. Ukoliko se radi o projektilu čiji sloj je označen brojem 19, glavnom neprijatelju život će se smanjiti za 3. Nakon toga slijedi provjera o trenutnom broju života Behemota. Ukoliko je jednak ili manji od nule, postavljaju se svi dozvoljeni napadi na 0, što znači da se više neće odabrati sljedeći nasumičan napad Behemota kod metode „PickAttackAction()“ o kojoj će se uskoro govoriti. Isto tako, na kraju se poziva metoda „Death()“ koja je primjer virtualne metode. Prvo

slijedi programski kod iz skripte AbstractBoss.cs, a onda nadjačavanje u skripti FernBehemoth.cs:

```
/// <summary>
/// Stop the boss from attacking.
/// Save measured time in Firebase.
/// </summary>
protected virtual void Death()
{
    timerCountDown.GetComponent<TimeCountDown>().countTime = false;
    float countedTime =
          timerCountDown.GetComponent<TimeCountDown>().GetTime();
    HighScores.PrepareForDatabase(GetType().Name, „Player1“,
          countedTime);

    rigidBody2d.velocity = Vector2.zero;
}
```

Unutar Death() metode zaustavi se mjerenje vremena iz skripte „TimeCountDown.cs“, dohvati se vrijeme pomoću „GetTime()“ metode te se poziva statička metoda „PrepareForDatabase()“ iz klase „PlayerData.cs“ koja tada sprema u bazu izmjereno vrijeme i ime igrača. Također se postavi brzina glavnih neprijatelja na 0. Nadjačavanje upravo opisane metode u FernBehemoth.cs skripti radi se na sljedeći način:

```
/// <summary>
/// Calls a virtual method from the abstract class
/// and then uses specialized code for death.
/// </summary>
protected override void Death()
{
    base.Death();

    audioSourceStepSFX.enabled = false;

    spriteRenderer.color = new Color(0.2924528f, 0.2924528f, 0.2924528f);

    GameObject.Find("HitboxWalk").SetActive(false);
    GameObject.Find("HitboxTurn").SetActive(false);

    Destroy(gameObject, 5f);
    StartCoroutine(DeathScreams());
}
```

Iako se koristila ključna riječ *override*, svejedno se žele izvršiti dijelovi koda istoimene metode koji su napisani u roditeljskoj klasi, što se postignulo s „base.Death()“. Linije koje su dalje napisane odnose se na deaktiviranje *collidera* Behemota tako da više ne može ozlijediti igrača, niti igrač njega. Također se deaktiviraju i ostali *collideri* vezani uz okretanje Behemota na drugu stranu dok dođe na kraj litice. Na kraju se koristi „Destroy()“ koji obriše *gameobject* (Behemota) nakon što prođe 5 sekundi i pozove metoda za pokretanje zvuka umirućeg Behemota.

8.1.2. Skripta prvog glavnog neprijatelja

Mnoge varijable definiraju se i inicijaliziraju prilikom pokretanja skripte Behemota, što se događa najviše unutar Awake() i Start() metode. Najčešće se koristila za glavne neprijatelje Start() metoda za to, pri čemu je Start() metoda Behemota reprezentativna:

```
// Start is called before the first frame update
void Start()
{
    rigidBody2d = GetComponent<Rigidbody2D>();
    animator = GetComponent<Animator>();
    spriteRenderer = GetComponent<SpriteRenderer>();
    audioSourceStepSFX = GetComponent<AudioSource>();

    rigidBody2d.velocity = Vector2.zero;

    minAttack = 1;
    maxAttack = 4;

    // Etc...
}
```

Kao što se može primijetiti, unutar Start() metode vrši se dohvaćanje komponenata koje su dodijeljene Behemotu kao *gameobjectu*, pritom se brzina, odnosno rigidBody2d.velocity postavlja na nulu. Ono što je važno za reći su minAttack i maxAttack koji su zajedno tipa int te određuju koje napade Behemot smije koristiti. Odabir napada događa se u sljedećoj metodi:

```
/// <summary>
/// Randomizing numbers to decide which attack should be used next.
/// If the current chosen attack is the same as the previous one,
/// method will call itself (Recursive method).
/// </summary>
private void PickAttackAction()
{
    velocityAttacking = facingLeft ? velocity * (-1) : velocity;
    int randomMove = Random.Range(minAttack, maxAttack);

    if (!isAttacking)
    {
        if (alreadyUsedAttackMove == randomMove)
        {
            PickAttackAction();
            return;
        }

       isAttacking = true;

        switch (randomMove)
        {
            case 1:
                StartCoroutine(LeaveGreenFire());
                break;
            case 2:
                StartCoroutine(AttackRun());
                break;
```

```
                case 3:
                    StartCoroutine(BreatheFire());
                    break;
                case 4:
                    StartCoroutine(StartFireOnTrees());
                    break;
                default:
                    StartCoroutine(BreatheFire());
                    break;
            }

            alreadyUsedAttackMove = randomMove;
        }
    }
```

Prvo se u metodi određuje brzina Behemota ispitivanjem *bool* vrijednosti varijable „facingLeft“ pomoću operatora „?“ i „:“ [59]. Ukoliko je facingLeft, što znači okrenut prema lijevo, istinit, tada pomnoži njegovu brzinu s -1 i dodijeli tu vrijednost varijabli velocityAttacking, odnosno brzini napadanja. U protivnom će brzina biti jednaka pozitivnoj vrijednosti varijable velocity. Nadalje, lokalno kreiranoj varijabli „randomMove“ dodijelit će se nasumična vrijednost, što ovisi o vrijednostima minAttack i maxAttack. Primjerice, ukoliko je minAttack jednak 2, a maxAttack jednak 4, tada će Random moći nasumično odabrati brojeve 2 i 3 te će Behemot smjeti koristiti samo one napade koji su označeni tim brojevima [60].

Nakon što se broj za napad odabere i ako Behemot nije u akciji napadanja, tada se preispituje nije li zadnji iskorišten napad jednak onom koji se nasumično odabrao da se izvrši jer se ne želi postići da Behemot uzastopno koristi jednu te istu vrstu napadanja. Ako je to istina, metoda se ponovno poziva kako bi se ponovno odabrao nasumičan broj te se izvršava ista provjera tako dugo dok ne dođe broj različit od broja koji označava posljednje izvedeni napad. Takve metode koje same sebe pozivaju nazivaju se rekurzivnim metodama [58, str. 498]. Tada se postavi da neprijatelj napada (isAttacking = true;) te slijedi „switch“ izraz kojemu se dodijeli vrijednost varijable randomMove kako bi se pokrenuo „case“ koji odgovara toj vrijednosti. Ukoliko se radi o vrijednosti 1, pokrenut će se napad „LeaveGreenFireOnGround()“, što u prijevodu znači „ostavljaj zelenu vatru na tlu“. Na kraju metode se varijabli „alreadyUsedAttackMove“ dodijeli vrijednost napada koji se nasumično odabrao i izvršio.

Od svih navedenih metoda napadanja u *switch* izrazu, detaljnije će se reći o „BreatheFire()“ pomoću koje Behemot riga vatru. Metoda je sama po sebi jedna od većih, no potrebno ju je objasniti jer se za neke napade kod ostalih glavnih neprijatelja koristi ista shema kao i kod nje. Kod za to je sljedeći:

```
    /// <summary>
    /// Breathes green fire at the player
```

```
/// </summary>
/// <returns></returns>
private IEnumerator BreatheFire()
{
    rigidBody2d.velocity = Vector2.zero;

    audioSourceStepSFX.enabled = false;
    animator.SetBool("BeIdle", true);

    int lowerNoFireballs = halfway == true ? 4 : 0;
    int noFireballs = Random.Range(7 - lowerNoFireballs, 10 -
        lowerNoFireballs);

    for (int i = 0; i < noFireballs; i++)
    {
        if (health <= 0)
        {
            yield break;
        }

        GameObject clonedGreenFire = Instantiate(
               greenFireball,
               new Vector3(
                   mouthPosition.transform.position.x,
                   mouthPosition.transform.position.y,
                   transform.position.z),
                   Quaternion.identity);

        PlaceClonedGreenFire(clonedGreenFire, i);
        SingletonSFX.Instance.PlaySFX("SFX20_floor_fire");

        Destroy(clonedGreenFire, 4f);
        yield return new WaitForSeconds(0.4f);
    }

    animator.SetBool("BeIdle", false);
    rigidBody2d.velocity = new Vector2(velocityAttacking + (facingLeft ?
        -8 : 8), 0f);
    audioSourceStepSFX.enabled = true;
}
```

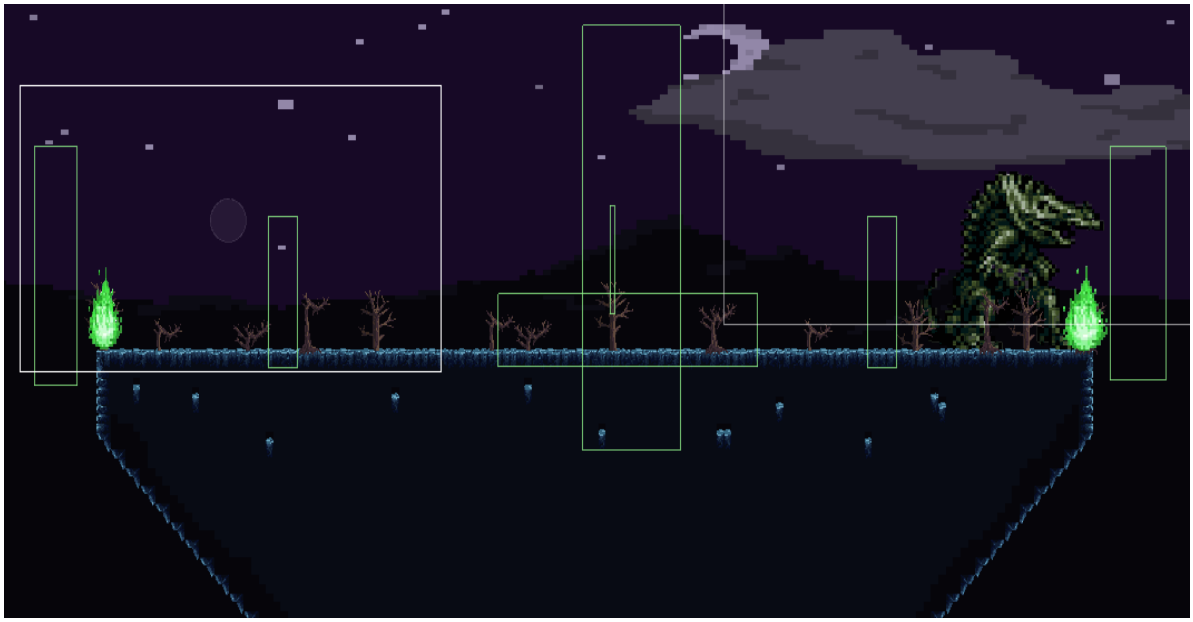
Prvo se postavi brzina neprijatelja na 0, zaustavi se sviranje zvučnog efekta hodanja koji je većinom vremena pokrenut, pokrene se animacija za mirovanje, odredi se broj za varijablu „noFireballs“ koji će smanjiti sveukupan broj vatrenih kugla za instanciranje, što zapravo ovisi o trenutnoj fazi borbe jer se želi postići da se manji broj vatrenih kugla instancira tijekom druge faze. Tada se nasumično odredi sveukupan broj vatrenih kugla za instanciranje na što dodatno utječe *noFireballs*. Zatim se pokrene izvođenje „for“ petlje koja će se vrtjeti onoliko puta kolika je vrijednost *noFireballs*. Unutar nje se prvo ispituje nije li kojim slučajem u međuvremenu Behemot poražen. Ako nije, instancira se klon „clonedGreenFire“ objekta „greenFireball“. To se postiže naredbom „Instantiate()“ u koju je potrebno prvo postaviti *gameobject* koji se želi klonirati, u ovom slučaju prefab *greenFireball* koji sadrži u sebi skriptu Projectile.cs. Zatim se mora odrediti mjesto gdje se želi stvoriti klon pomoću „new Vector3()“ pozicije, a potom je dalje

potrebno odrediti x, y i z poziciju mjesta stvaranja, što su u ovom slučaju usta Behemota. Zadnje za Instantiate se pomoću „Quaternion.identity“ [61] odredi može li se novostvoreni objekt u videoigri rotirati.

Novokreiranom projektilu potrebno je naštimati brzinu i rotaciju, što se sve događa u metodi „PlaceClonedGreenFire()“. Brzina se tako korigira „set“ metodom za varijable MultipliedVelocityX i MutlipliedVelocityY koje se množe na postojećoj brzini *rigidbodyja*, dok se izmjena rotacije postiže pomoću „Quaternion.Euler()“ [62]. Na kraju for petlje definira se Destroy() koji će obrisati projektil za 4 sekunde te se koristi „yield return new WaitForSeconds()„ kako se ne bi stvorila nova vatrena kugla odmah nakon trenutne. Na kraju metode se vrati prvotna vrijednost brzine Behemota, kao animacija i zvuk hodanja.

Drugi napadi Behemota realizirali su se na sličan način, a to je korištenje funkcije „Instantiate()“, definiranje gdje će se stvoriti vatra, koja će mu biti brzina te koliko dugo će trajati. Metoda koja je malo različitija jest „AttackRun()“ u kojem se poveća vrijednost brzine *rigidbodyja* nakon kratkog stajanja te se vrati na normalnu brzinu nakon što dođe do kraja litice.

Vrijedi spomenuti da se na svakom kraju litice nalaze *collideri* koje ukoliko ih Behemot dodirne, okrene se i bježi prema drugom kraju velike stijene. Ti *collideri* se nakratko deaktiviraju kako se Behemot ne bi na jednom mjestu cijelo vrijeme okretao. U neposrednim blizinama litica nalaze se *collideri* zbog kojih Behemot počinje koristiti svoje napade nakon dodira s njima. Izgled scene sa svim *colliderima* označenima zelenim crtama za tu arenu može se vidjeti na sljedećoj slici (Slika 9.):



Slika 9. Fern Behemoth arena: *collideri* arene za Behemota [autorski rad]

8.1.3. Skripta projektila

Svi projektili u *Unityju* su pohranjeni kao prefabi te svaka od njih sadrži kao komponentu skriptu „Projectile.cs“. Već je spomenuto da ta klasa sadrži članove MultipliedVelocityX i MultipliedVelocityY za koje su definirane „get“ i „set“ metode, odnosno *geteri* i *seteri*. Najvažnija metoda u toj skripti jest OnTriggerEnter2D(). Unutar nje se prvo *if* selekcijom ispituje sljedeće:

```
if (((collision.transform.rotation.eulerAngles.y == 180
        && transformParent.rotation.eulerAngles.y == 180)
        || (collision.transform.rotation.eulerAngles.y == 0
        && transformParent.rotation.eulerAngles.y == 0))
    && collision.gameObject.layer == 17
    && parried == false)
```

Ona ispituje je li palatin sa svojim aktivnim štitom koji ima zaseban *boxcollider* okrenut prema projektilu koji dolazi prema njemu. To je važno jer se u protivnom projektil koji dolazi s lijeve strane može odbiti od štita koji je okrenut prema desnoj strani, dok je željeni scenarij da je štit okrenut prema nadolazećem projektilu. Navedeno se ispituje rotacijama tih *gameobjecta* koje se dobiva „transform.rotation.eulerAngles.y“, s time da je ova skripta dodijeljena djetetu glavnog *gameobjecta* projektila, stoga se uzima rotacija roditelja pomoću „transformParent“. Nadalje, mora se zadovoljiti da kolizija dolazi od *gameobjecta* čiji je *layer* označen brojem 17, što je zapravo *layer* štita.

Zadnje mora biti zadovoljen uvjet da projektil ne smije biti već odbijen. Nakon toga se varijabli „parried“ dodjeljuje vrijednost *true*, *layer* postane jednak 19, što je važno jer kada dođe u koliziju s glavnim neprijateljem, život glavnog neprijatelja će se uvelike smanjiti. Na kraju se promijeni vrijednost *rigidbody* projektila.

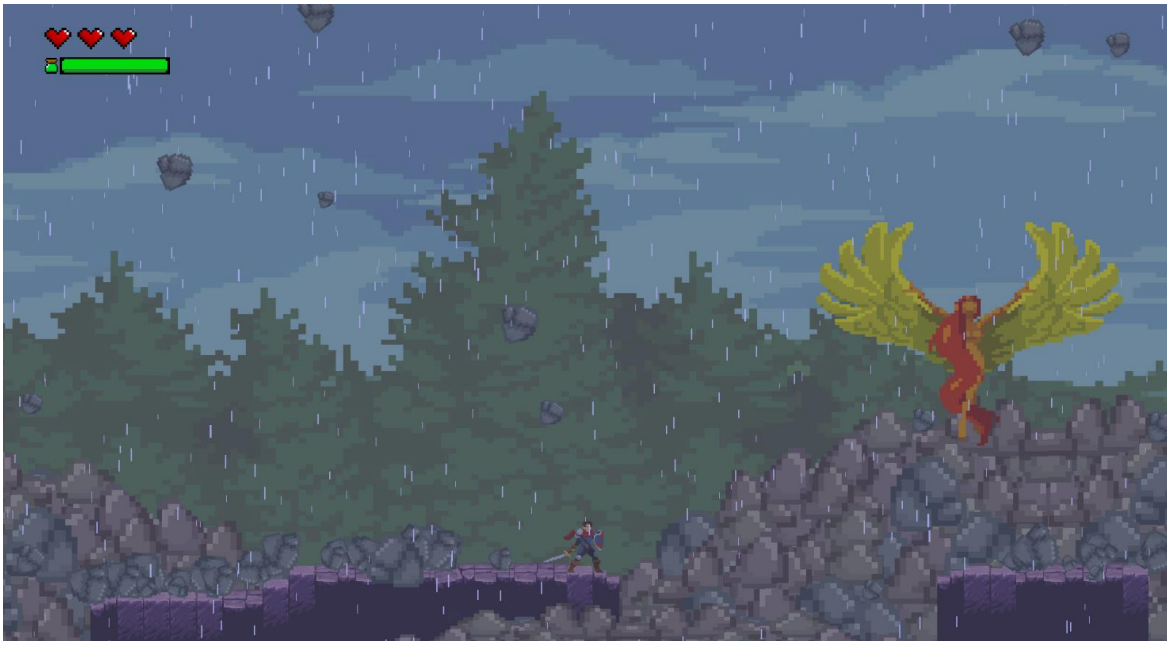
8.2. Drugi glavni neprijatelj – Psychic Psycho

Drugi glavni neprijatelj nazvan je *Psychic Psycho*, što znači „telepatski psihopat“, a u videoigri je to pali ženski anđeo koji svoje napade temelji na telepatskim moćima. Njezini napadi su:

1. zbunjivanje palatina na način da se pritiskom na gumb za kretanje ulijevo palatin kreće prema desno
2. naizmjenično teleportiranje u neposrednoj blizini igrača te letenje prema njemu
3. rotiranje kamere naopako
4. prisiljavanje palatina da u kratkom vremenu visoko skače bez prestanka
5. napad oštrim perima koji se stvaraju lijevo i desno od nepomičnog palatina koji se od njih može obraniti jedino štitom
6. stvaranje magičnog kruga oko nje prilikom početka svakog napada koji ozljeđuje palatina ako ga dodirne.

Sama borba s njom odvija se unutar katedrale (Slika 10.), no kada joj igrači oduzmu polovicu sveukupnog života, naljuti se, postane brža u napadanju i uništi cijelu katedralu (Slika 11.). Stoga se druga polovica borbe odvija vani u prirodi na ruševnima tijekom kišnoga dana. U početku rada opisalo se da se *Boss Rush* igre temelje najviše na *gameplayu* i na prezentaciji, gdje se upravo takva izmjena mjesta borbe odnosi na prezentacijski dio i tako učini glavnog neprijatelja vrijednim spomena.

Slika 10. Borba protiv anđela u katedrali – kamera je naopako okrenuta te oštra pera napadaju palatina [autorski rad]



Slika 11. Borba protiv anđela na porušenoj katedrali [autorski rad]

Za ovog glavnog neprijatelja, kao i za nadolazeće, u kodu će se govoriti samo bitnije i specifične funkcije. To su zbunjivanje palatina i okretanje kamere, dok su se ostali napadi izveli slično kao i kod Behemota.

8.2.1. Skripta drugog glavnog neprijatelja

Anđeo svoje napade temelji na trenutnim pozicijama igrača. Behemot to sa svojim napadima nije radio, no ovaj i idući glavni neprijatelji moraju cijelo vrijeme znati trenutne pozicije igrača te ponekad pristupiti vrijednostima varijabala njihovih skripata. Zbog toga se na samom početku skripte „PsychicPsycho.cs“ izradila nova lista *gameobjecta* pod nazivom „heroes“ te su se toj listi dodali *gameobjecti* igrača koristeći Find(), gdje je naziv *gameobjecta* prvog igrača „PlayerKnight1“ dok je drugog „PlayerKnight2“. Jedan od napada koji mora dohvatiti *gameobject* palatina jest „InvertControlsAttack()“ čiji poziv se događa unutar *switch* izraza.

Kada se pozove metoda InvertControlsAttack(), odmah se pozove, kao i kod svake druge metode anđelovih napadanja, stvaranje magičnog kruga. Postizanje izvrtanja kontrole igrača događa se na način da se dohvati skripta palatina te se prvo pozove metoda „OnDisable()“, koja dalje dohvaća skriptu PlayerInput.cs i poziva metodu DeactivateInput(). Razlog zašto se na kratko želi onemogućiti igračima unos novih naredbi preko njihovih kontrolera jest što u protivnom igrači mogu i dalje držati tipku za kretanje, dok se preokretanje kontrola primjenjuje samo onda dok se ne drži tipka. Poziva se zato iz skripte palatina „InvertMovementDirection()“

koja varijabli movementDirection promjeni trenutni predznak na suprotnu vrijednost. Metoda MovementInput() je ta koja koristi tu varijablu, no ona je već bila opisana u poglavlju „*Sistem unosa*“. Nakon kratkog vremena, ponovno se poziva metoda InvertMovementDirection() za vraćanje u normalno stanje.

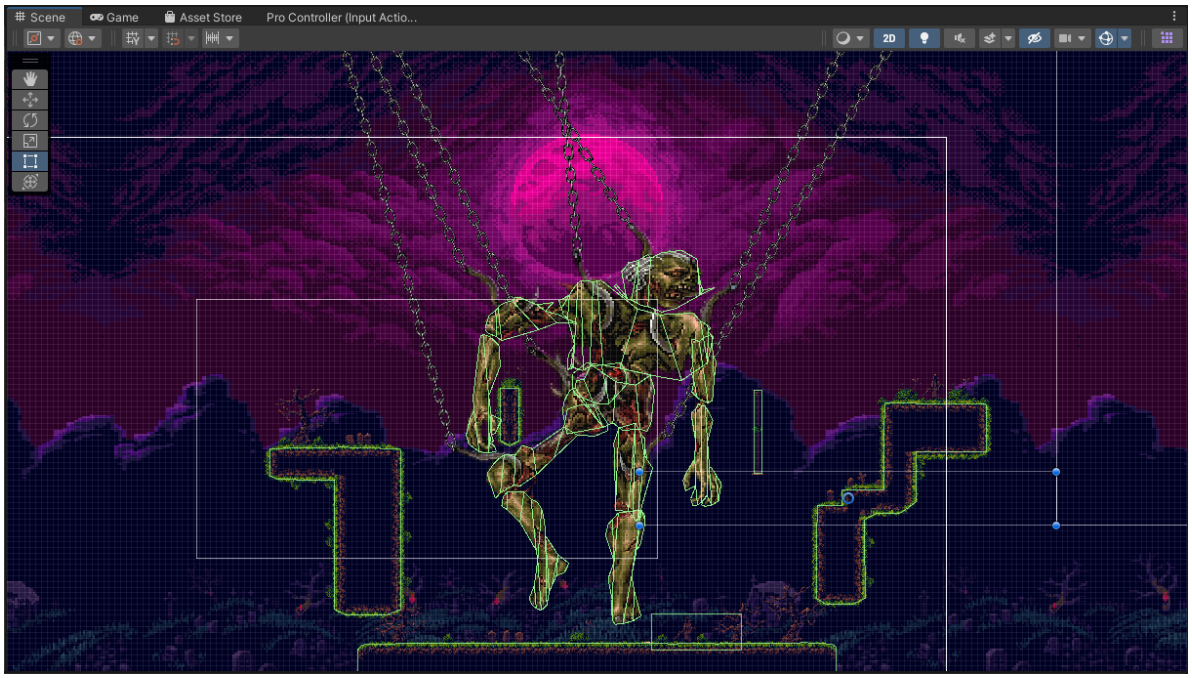
Što se anđela tiče, vrijedi spomenuti još jednu metodu, a to je „RotateCameraAttack()“. Slijedi isječak iz te metode:

```
for (int i = 0; i <= 180; i++)
{
        virtualCamera.GetComponent<CinemachineVirtualCamera>()
                .m_Lens.Dutch = i;
        yield return new WaitForSecondsRealtime(0.01f);
}
```

Gameobject „virtualCamera“ sadrži komponentu koju se naknadno moralo u *Unityju* preuzeti i uvesti, a to je *Cinemachine*. *Cinemachine* su moduli za napredno upravljanje kamerama, stoga *developeri* ne trebaju trošiti svoje vrijeme na razvijanje svoje skripte za postojeću glavnu kameru [63]. Uglavnom, unutar for petlje dohvati se ta komponenta „CinemachineVirtualCamera“ te se preko „,m_Lens“, odnosno leća kamere dohvati *Dutch* i postavi trenutna vrijednost varijable „i“ koja se tijekom for petlje povećava sve do vrijednosti 180. Na taj način se kamera polako rotira sve do 180 stupnjeva. Za vraćanje na normalnu poziciju, ponovno se koristi for petlja, ali je tada početna vrijednost „i“ jednaka 180 te se svakom iteracijom smanji za 1.

8.3. Treći glavni neprijatelj – Chained Undead

Nakon pobjede protiv anđela, igrači ulaze u treću arenu i vide groteskan prizor. Taj prizor je zapravo treći glavni neprijatelj *Chained Undead* (Slika 12.), što u prijevodu znači „lancem okovani živi mrtvac“.



Slika 12. Treći glavni neprijatelj u svojoj areni [autorski rad]

Taj se glavni neprijatelj ne može micati zbog tih lanaca, nego se oslanja na pozivanju svojih sljedbenika: dva kostura i velike muhe. Kosturi se nalaze na dnu arene te oni slijede igrače po X-osi. Igrači ih mogu uništiti, ali nakon nekoliko sekundi ponovno ožive. S druge strane postoje 3 vrste muha. Obična muha samo slijedi igrače po X i Y-osi kao i ostale vrste muha. Žuta jurišna muha se u blizini igrača nakratko zaustavi te velikom brzinom poleti do posljednjeg mjesta gdje se igrač nalazio prije negoli se ona zaustavila. Otrovna zelena muha riga kuglu otrova prema igračima.

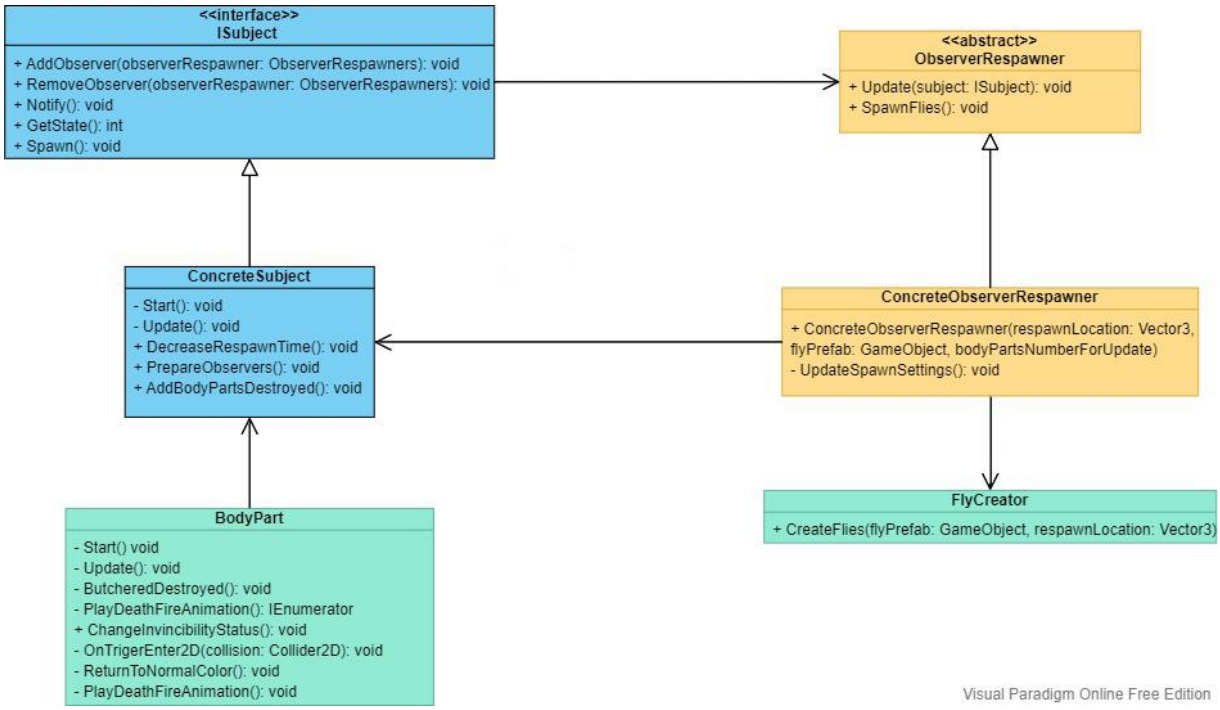
Mrtvaca se može uništiti samo ako se njegovo srce uništi, no srce se neće pokazati igračima tako dugo dok se svi ostali dijelovi tijela osim ramena i glave ne unište. Mora se napomenuti da se određeni dijelovi tijela ne mogu uništiti ako se ne unište neki prijašnji. Primjerice, gornjem dijelu noge šteta se može nanijeti samo ako je donji dio noge uništen.

S obzirom na to da je glavni neprijatelj statičan, manji neprijatelji su ti koji igrače napadaju, no u idućem poglavlju neće se objašnjavati skripte njih niti glavnog neprijatelja, nego će se preko dijagrama objasniti procese stvaranja muha koje se događaju cijelo vrijeme tijekom bitke. To se najučinkovitije postiglo primjenjivanjem uzorka dizajna „Observer“.

8.3.1. Skripte za stvaranje sljedbenika trećeg glavnog neprijatelja

Bilo je rečeno da postoje 3 vrste muha, no one se zajedno u početku ne počinju stvarati. Obična muha se stvori nakon što se unište donji dijelovi noga, pri čemu se tada leteći otoci arene (koji su u početku bitke nevidljivi) također stvore da se igrači mogu popeti na njih i onda ostale dijelove uništiti. Nakon što se uništi još jedan dio tijela, počinje se stvarati jurišna muha, a nakon četvrtog uništenog i otrovne muhe. Isto tako, one se stvaraju na različitim mjestima arene.

Za sve rečeno prepoznalo se da bi se problem najučinkovitije riješio primjenom uzorka dizajna *observer*. *Observer* je objektni uzorak ponašanja (engl. *Behavioral Design Pattern*) u kojemu između klasa postoji veza jedan naprema više [64, str. 162] [65, slajd 51]. Ta veza se postiže tako da jedna klasa koja sadrži najnovije promjene podataka obavijesti ostale klase koji promatraju tu klasu da je došlo do promjene. Ti promatrači nazivaju se *observeri*. Oni tada traže od te prve klase da im daje najnovije stanje podataka te nakon toga dalje izvode ostale procese za koje su namijenjeni. Na sljedećoj slici (Slika 13.) može se vidjeti izrađeni UML dijagram klasa koji služi za prikazivanje klasa i komunikaciju između njih, a u ovom slučaju prikazuje klase koje su dio *observer* uzorka dizajna [66].



Slika 13. UML dijagram klasa uzorka dizajna *observer* za stvaranje neprijatelja [autorski rad]

Kao što se vidi na slici 13, ovaj uzorak dizajna sastoji se od jednog sučelja (engl. *interface*) „ISubject.cs“ koje zahtjeva da klasa koja ju implementira, „ConcreteSubject.cs“ mora obavezno nadjačati njegove metode. ConcreteSubject.cs nasljeđuje i klasu MonoBehaviour

koju još nasljeđuju i klase „BodyPart.cs“ i „FlyCreator.cs“, dok ostale u dijagramu to ne rade. Nadalje, metode koje *ConcreteSubject* implementira su „AddObserver()“ koji dodaje *observere* u svoju listu *observera* i „RemoveObserver()“ koji izbriše danog *observera* iz liste, „Notify()“ koji obavještava *observere* da je došlo do promijene, odnosno da su igrači još jedan dio tijela uništili, „GetState()“ koji vraća trenutni broj uništenih dijelova tijela i „Spawn()“ koji javlja *observerima* da je vrijeme da kreiraju novu muhu. Uz navedene metode, on još sadrži i neke svoje koje je moguće vidjeti na dijagramu.

Što se *observera* tiče, kreirala se apstraktna klasa „ObserverRespawner.cs“ koju nasljeđuje klasa „ConcreteObserverRespawner()“, što se može vidjeti na dijagramu da crta s bijelim ispunjenjem usmjerena na apstraktnu klasu označava nasljeđivanje, odnosno generalizaciju [67]. Metode koje mora nadjačati su „Update()“ koji prima objekt tipa „ISubject“ te ga koristi tako da dohvati trenutnu vrijednost broja uništenih dijelova tijela. Ne smije se pomiješati ta Update() metoda s onom koja je definirana u klasi MonoBehaviour. Također nadjačava metodu „SpawnFlies()“ koja javlja klasi „FlyCreator()“ da mora kreirati novu muhu određenog tipa u areni na definiranoj poziciji.

Cijeli scenarij *observera* kao uzorka dizajna radi na način da kada igrači unište *gameobject* koji predstavlja dio tijela, skripta „BodyPart.cs“ dodijeljena tom *gameobjectu* metodom „ButcheredDestroyed()“ javlja skripti ConcreteSubject.cs *gameobjecta* „RespawnController“ (u prijevodu znači upravitelj stvaranja) da je još jedan dio tijela uništen. Konkretni subjekt doda broj 1 u svojoj varijabli „destroyedBodyParts“ koji je tipa int te poziva metodu Notify(). Taj subjekt općenito ima spremljenu listu svih *observera* koji prate te događaje te kod svakog poziva metode „observer[i].Update()“ prosljeđuje sebe s „this“. *Observer* unutar svoje metode Update() traži od konkretnog subjekta podatak o broju uništenih dijelova tijela. Konkretni subjekt mu vrati taj podatak te *observer* kod sebe ažurira maksimalan broj muha određene vrste koji smiju biti u areni, bilo da se prvo radilo o nuli ili se broj dodatno povećao jer su igrači gotovo sve dijelove tijela uništili.

S obzirom na to da konkretni subjekt sadrži listu svih *observera*, odmah se iskoristilo to da svakih nekoliko sekundi konkretni subjekt javlja *observerima* da je vrijeme da stvore nove muhe. Izgled opisanih muha može se vidjeti na sljedećoj slici (Slika 14.):

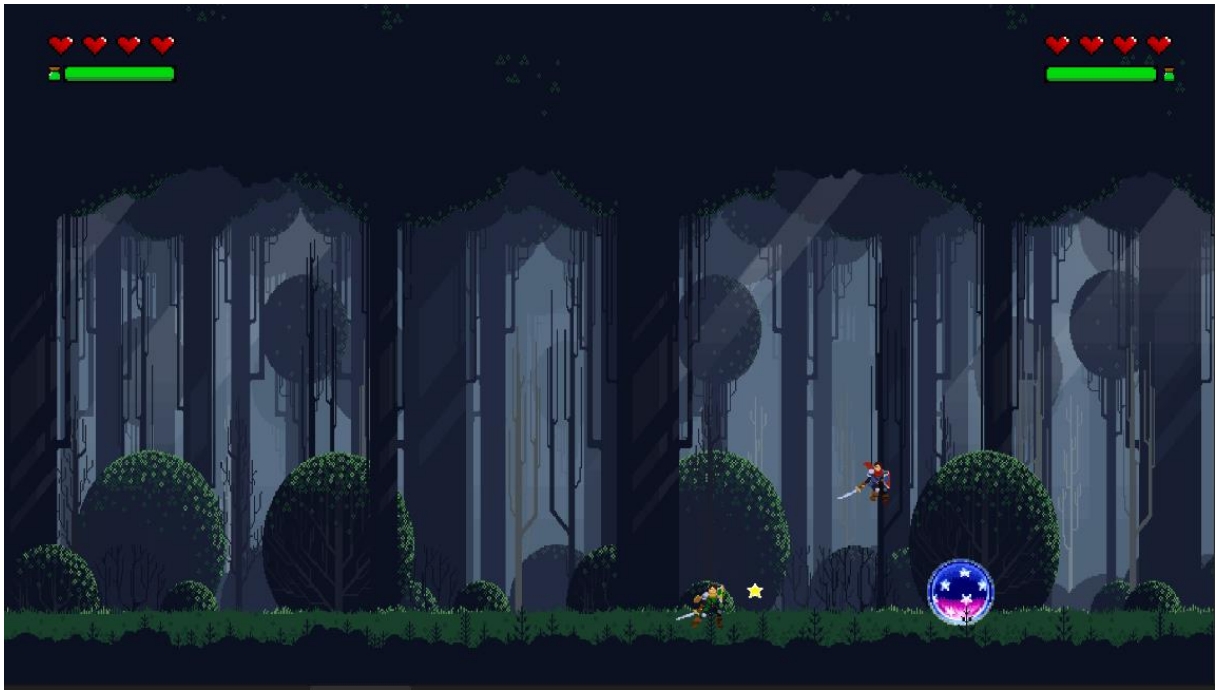
Slika 14. Muhe redom s lijeva na desno: obična, jurišna, otrovna s mjehurićem [autorski rad]

8.4. Četvrti glavni neprijatelj – Sidus Istar

Predzadnji glavni neprijatelj videoigre je čarobnjak pod imenom *Sidus Istar*, što na latinskom „Sidus“ znači zvijezda, dok „Istar“ na izmišljenom jeziku pisca J.R.R. Tolkiena znači čarobnjak [68]. Ovaj čarobnjak ima dva oblika, od kojih je prvi zapravo oblik kugle, dok pravi izgled čarobnjaka igrači susreću tek na kraju borbe.

Sama borba osmišljena je tako da igrači ne vode bitku protiv čarobnjaka samo na jednom mjestu. Nakon prve kratke bitke protiv kugle, kamera se počinje pomicati vertikalno prema nebu pa sve do svemira, dok za to vrijeme igrači moraju savladati prepreke raznih vrsta. Scenarij cijele borbe uz opise i slike je idući:

1. u početnoj borbi kugla ispaljuje žute zvijezde koje igrači zaustave štitom te zamahom mača lansiraju natrag prema čarobnjaku (Slika 15.)



Slika 15. Palatini se bore protiv čarobnjaka [autorski rad]

2. kamera se počinje kretati prema nebu te se igrači moraju uspinjati po planini, pritom moraju paziti na oštre bodlje koje se nalaze na raznim dijelovima planine, kotrljajuće velike stijene čije kretanje je izazvao čarobnjak, velike plave zvijezde koje čarobnjak ispaljuje na onom dijelu arene gdje igrači moraju dugo skakati po zidovima, skakati na oblake koji se kreću različitim brzinama i pritom paziti da ne padnu te izbjegavati okrugle tamne tvari koje čarobnjak ispaljuje dok se igrači uspinju po oblacima

Slika 16. Palatin preskače velike kamene [autorski rad]

3. borba protiv pravog izgleda čarobnjaka odvija se u svemiru gdje je sila gravitacije mala, stoga palatini visoko skaču te moraju izbjegavati razne zvjezdane napade čarobnjaka

Slika 17. Borba protiv pravog oblika čarobnjaka [autorski rad]

Može se zaključiti da postoje razni događaji kod ovog glavnog neprijatelja. Ne samo da se mnogo vremena izdvojilo za izradu skripata za njihovo ostvarenje, već je ovoga puta veću ulogu same borbe imao sam *Unity engine*. Ukratko će se nešto reći i o tome.

8.4.1. Događaji četvrtog glavnog neprijatelja

Do sada je bio slučaj da kamera slijedi igrače koristeći *Cinemachine*. Kamera ovoga puta nikoga ne slijedi nego je u početku statična. Nakon što igrači sa zvijezdama nanesu dovoljno ozljeda čarobnjaku, on pobjegne prema nebu, što je i uzrok polaganog kretanja kamere vertikalno prema gore. To se postiglo na način da se stavila *Rigidbody* 2D komponenta na objekt kamere te da se dodao pozitivan broj za Y-os kada dođe vrijeme za to. Na vrhu *gameobjecta* kamere dodao se *Box Collider* 2D i postavila se njegova postavka „Is Trigger“ na istinu, što znači da se igrači neće sudariti s tim colliderom kao s nekim fizičkim objektom, nego će proći kroz njega. Razlog zašto je „Is Trigger“ jednak *true* jest to što će upravo *gameobject* kamere poslužiti za okidanje novih događaja. Primjer za to je rušenje stijena kada kamera dođe do određenog dijela arene kako bi se stvaralo kamenje koje se kotrlja prema igračima.

Za kamenja se moralo unutar *Unityja* u *Rigibody* 2D Postaviti da je „Body Type“ jednak „Dynamic“, što znači da zakoni fizike koji su implementirani u *engineu* vrijede za ta kamenja. Nadalje, morao se povećati „Gravity Scale“, odnosno sila gravitacije tih kamenja jer će u protivnom visoko odskakati po *collideru* tla. Također se naveliko morao povećati *Mass*, što je masa tih kamenja jer se tijekom testiranja dogodilo da su se igrači mogli sudariti s kamenom te ga s lakoćom pomicati, što nije željeni scenarij.

Nakon dolaska do vrha planine, igrači skaču po oblacima. Za te oblake se uz *Edge Collider* 2D koristio i *Platform Effector* 2D koji daje mogućnost igračima da mogu na oblak skočiti od ispod. Za to se trebalo na *Platform Effector* 2D komponenti postavku „Use one way“ kvačicom označiti da je istinita, što znači da se može doći na *collider* s jednog smjera (u ovom slučaju od ispod) te dok je igrač iznad njega, može stajati na njemu. Na kraju se na *Edge Collider* 2D kvačicom označilo „Used By Effector“ kako bi se primijenio *Platform Effector* 2D.

Poslije oblaka, čarobnjak ispaljuje kugle tamne tvari prema igračima. Te se kugle kreću prema posljednjem mjestu na kojemu se nalazio igrač za što se, kao i kod jurišne muhe iz prošlog glavnog neprijatelja, unutar Update() metode koristio „Vector2.MoveTowards()“. Ta metoda računa poziciju između dvije točke, pri čemu se za prvi argument postavlja prva točka, za drugi argument druga, dok zadnji argument označava najveću udaljenost do koje se smije prijeći u danom korištenju te metode [69]. Ono što je drugačije između jurišne muhe i kugle tamnih tvari jest da se jurišna muha uvijek zaustavlja na onoj točki gdje se nalazio igrač prije jurišanja, što je kod „Vector2.MoveTowards()“ drugi argument, no za kugle tamnih tvari željelo se postići da se one i dalje kreću nakon što dođu do te točke. Problem se riješio korištenjem matematičke teorije vektora. Euklidski vektori su geometrijski objekti koji sadrže duljinu i smjer [70] te se u ovom slučaju želi produljiti duljina vektora prema kojoj se kreću kugle. Kako bi se

to postiglo, morala se izvesti određena matematička formula. Vektor $\vec{v}$ se prikazao u komponentnoj formi, pri čemu se sastoji od dvije komponente, horizontalne v_x i vertikalne v_y:

$$\vec{v} = \langle v_x, v_y \rangle$$

Te komponente označavaju pomake vektora između početne i krajnje točke po X i Y-osi. Neka A i B predstavljaju takve točke sa svojim x i y koordinatama. Potrebno je koordinate krajnje točke (B) oduzeti s koordinatama početne točke (A) za svaku os kako bi se dobile komponente vektora, odnosno:

$$A = (A_x, A_y) \qquad B = (B_x, B_y)$$
$$v_x = B_x - A_x \qquad v_y = B_y - A_y$$

Nadalje, dan je vektor $\vec{z}$ koji ima isti smjer kao i vektor $\vec{v}$, no on je *k* puta veći od vektora $\vec{v}$. Navedeno se iskazuje na sljedeći način:

$$\vec{z} = k\vec{v}$$

Produženi vektor $\vec{z}$ ima istu početnu točku kao i vektor $\vec{v}$. Pomoću toga dobiva se sljedeća formula za koordinate nove krajnje točke C za produljenje vektora [71]:

$$C_x = A_x + kv_x = A_x + kA_x + kB_x = kB_x + A_x(1-k) \qquad C_Y = kB_y + A_y(1-k)$$

Dobivena formula za točku C se koristila unutar skripte „DarkMatter.cs“. Na sljedećoj slici može se vidjeti da kugle tamne tvari zbog navedene formule putuju i dalje nakon što dođu do pozicije igrača koji se tamo nalazio u trenutku stvaranja kugla (Slika 18.):

Slika 18. Čarobnjak ispaljuje kugle tamnih tvari prema igraču [autorski rad]

8.5. Peti glavni neprijatelj – Glacial Overlord

Peti, ujedno i posljednji, glavni neprijatelj videoigre zove se Glacial Overlord, što znači „ledeni gospodar“. Često je kod videoigara slučaj da su zadnji glavni neprijatelji igračima najteži za proći, pri čemu ovaj neprijatelj nije izuzetak. Vrlo je brz, koristi mješavinu napada čarolijom i napada mačem. Ono što je kod njega karakteristično kada se bori protiv dva igrača jest da prije nego što napadne, donese odluku kojega će igrača napasti. Vjerojatnost je veća da napadne onog igrača kojemu je preostalo svega par životnih bodova jer nakon što ga se riješi, olakšat će si borbu s obzirom na to da će mu preostati samo još jedan igrač za pobijediti. Također je vjerojatnije da će napasti onog igrača koji mu se nalazi bliže.

Osim donošenja takvih odluka, odlučuje i o tome hoće li prestati koristiti neke vrste svojih napada ukoliko ih igrači previše puta iskoriste protiv njega. Primjer je odbijanje štitom manjih ledenih siga natrag prema glavnom neprijatelju koji ih je prizvao čarolijom. Umjesto toga će prizivati velike sige koje igrači štitom ne mogu odbiti. Napadi koje koristi su sljedeći:

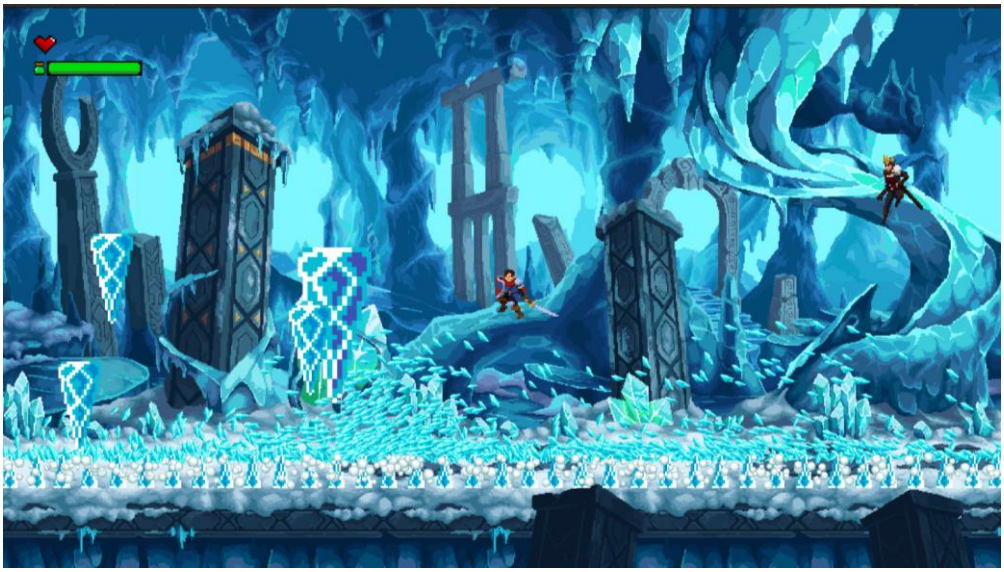
- prizivanje ledenih siga s lijeve i desne strane arene koje nakon kratkog vremena počinju padati te se zabode u tlo, na što igrači također moraju pripaziti jer tamo ostanu određeno vrijeme
- teleportiranje i korištenje velikog mača protiv kojeg se igrači ne mogu zaštititi štitom, nego moraju svoj mač koristiti i tako se mačevati protiv glavnog neprijatelja
- prizivanje ledenih siga sa stropa špilje koje se unište nakon što dodirnu tlo, osim zadnjega na kojega igrači moraju skočiti, u protivnom će dobiti ozljedu jer će iz cijeloga tla najednom izviriti mali oštri stalagmiti
- zaleđivanje igrača koji moraju što više puta u što kraćem vremenu pritisnuti tipku kako bi se mogli odlediti
- prizivanje ledenih zraka koje igrači skakanjem i kotrljanjem mogu izbjeći.

Kao i kod četvrtog glavnog neprijatelja, neće se ulaziti u detalje skripte jer se na iste načine instanciraju objekti i slično izvode napadi. Reći će se samo da se prioritiziranje igrača kojeg će se napasti ostvarilo na način da su se kreirale dvije varijable tipa int. Vrijednost pojedine varijable će se povećati za jedan ukoliko ona bolje zadovoljava određeni uvjet negoli druga. Primjerice, varijabla koja predstavlja prvog igrača će se povećati za jedan ukoliko se taj igrač bliže nalazi glavnom neprijatelju negoli drugi igrač. Prije nego što se pokrene izvršavati jedna od metoda napadanja glavnog neprijatelja, poziva se metoda „DecideWhichPlayerToAttack()“ koja usporedi vrijednosti varijabala prioritiziranja i vrati broj koji reprezentira onog igrača koji je zadovoljio više uvjeta. Izjednačenje se neće dogoditi jer je sveukupan broj uvjeta neparan.

Posljednje što bi se još željelo spomenuti jest „Quaternion.Slerp()“ koji se dosad koristio prvi put. Kako je bilo opisano, jedan od napada glavnog neprijatelja jest da prizove leteće ledene sige koje kratko vrijeme lete i onda padnu na tlo. Za to vrijeme dok padaju, počinju se okretati toliko dugo dok vrh sige nije okrenut prema zemlji, pri čemu se onda zabode u tlo. Zato se koristio Quaternion.Slerp() koji omogućava takvo rotiranje objekta, no prije toga mu je bilo potrebno dodijeliti rotacijske vrijednosti dva Transform objekta i mjeru po kojoj bi se rotirali [72].

Osim realiziranih napada pomoću skripata, nužno je reći da je sama borba popraćena epskom glazbenom skladbom kao i mnogim vizualnim efektima. Ovdje se radi o glavnom neprijatelju cijele videoigre, stoga se igračima želi ostaviti čim bolji dojam o videoigri.

Slika 19. Mačevanje protiv glavnog neprijatelja videoigre [autorski rad]



Slika 20. Glavni neprijatelj videoigre napada palatina svojim moćima [autorski rad]

9. Baza podataka

Spomenulo se korištenje baze podataka *Firebase*, točnije njegova usluga *Realtime Database*. Iako on nudi podršku za Unity videoigre u obliku kompleta za razvoj softvera (engl. *Software Development Kit,* SDK) poznatog kao „Firebase Unity SDK“, ona služi samo za mobilne videoigre iOS i Android uređaja [73]. Zato se koristio *Realtime Database* čiji su se podaci spremali i dohvaćali u obliku JSON formata pomoću REST API-ja. Bitno je objasniti što je API, a što je REST(ful) API. Aplikacijsko programsko sučelje (engl. *Application programming interface*, API) je, kako navodi službena stranica IBM-a: „*set pravila koji definiraju kako se aplikacije ili uređaji spajaju i komuniciraju međusobno*“ [74]. Nadalje, također su definirali REST API, što je: „*API koji se pridržava načela dizajna REST-a, odnosno reprezentacijskog stanja transfera arhitekturalnog dizajna*“ [74]. Opisat će izrada i korištenje baze podataka.

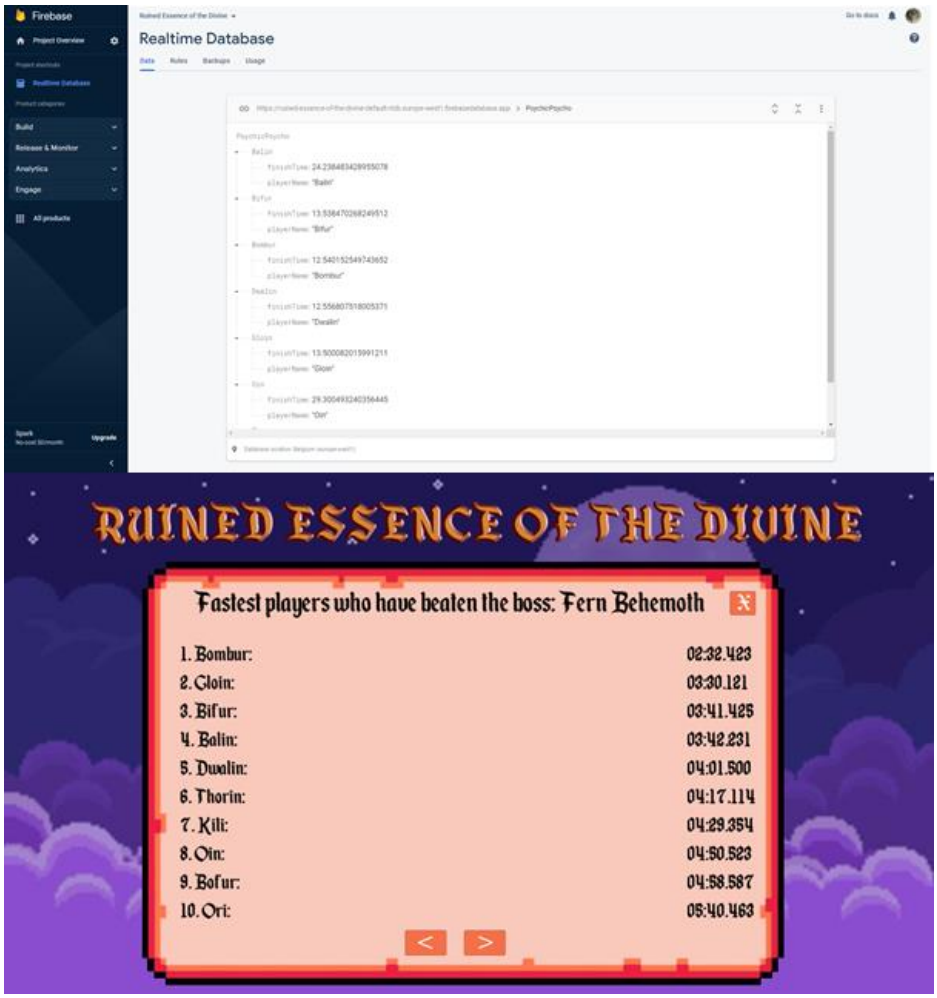
Prvo je bilo potrebno na *web*-stranici *Firebasea* stvoriti novi projekt i dati mu ime. Tada je bilo potrebno definirati *Realtime Database* i odabrati jednog od raspoloživih servera iz liste koji se nalaze na raznim kontinentima svijeta. Nadalje, preuzeo se besplatan REST API iz *Unity Asset Storea* i uvezlo ga se u projekt. Tada se kreirala skripta „Player.cs“ koja sadrži konstruktor u koji je potrebno unijeti ime igrača i izmjereno vrijeme bitke protiv glavnog neprijatelja. Nakon toga se izradila skripta „PlayerData.cs“ koja služi za pohranjivanje podataka u bazu te dohvat istih iz baze. S obzirom na to da se iz baze podataka za svakog glavnog neprijatelja dohvaćaju podaci u obliku JSON formata, potrebno ih je iz takvog formata pretvoriti (engl. *deserialize*) u oblik s kojim je jednostavnije manipulirati unutar skripte. Zato se iz GitHuba preuzeo besplatan kod JSON deserializer. Slijedi dio programskog koda metode „GetDataFromDatabase()“ skripte PlayerData.cs koja služi za dohvat podataka iz baze:

```
RestClient.Get("https://ruined-essence-of-the-divine-default-rtdb.europe-
         west1.firebasedatabase.app/" + boss + ".json").Then(x =>
        {
            fsData playersData = fsJsonParser.Parse(x.Text);
            Dictionary<string, Player> playerDataDictionary
                      = new Dictionary<string, Player>();
            serializer.TryDeserialize(playersData, ref playerDataDictionary);
         //...
        });
```

Statička klasa „RestClient“ je dio koda koji se preuzet s GitHuba. Pomoću njene metode „Get()“ su se dohvatili podaci u JSON formatu, za što je bilo potrebno u metodi definirati poveznicu na bazu. U toj poveznici koristila se i vrijednost varijable „boss“ koja sadrži naziv glavnog neprijatelja za kojega se žele dohvatiti igrače s njihovim najboljim vremenima protiv njega. Tada se koristio programski kod za pretvorbu teksta u oblik koji će se pomoću objekta „serializer“ tipa „fsSerializer“ spremiti u rječnik „playerDataDictionary“. Rječnici su slični listama

(List<>), no oni su takvi tipovi podataka koji u sebi sadrže ključeve i vrijednosti tih ključeva, pri čemu su u ovom slučaju ključevi imena igrača, dok su vrijednosti objekti tipa *Player*. Kod metode „TryDeserialize()“ se za rječnik morala staviti i ključna riječ „ref“ koja govori da se vrijednosti prenose referencom [75] [76]. Ti podaci koriste se za prikaz imena i vremena prvih 10 najboljih igrača za odabranog glavnog neprijatelja, pri čemu se sortiranje na temelju vremena postiglo pomoću LINQ metode „OrderBy()“. Ukratko za LINQ, to je komponenta koja omogućuje .NET jezicima da rade upite za dane podatke [77].

Za unos podataka u bazu koristila se metoda „RestClient.Put()“ unutar koje se morala definirati putanja baze u koju će se spremiti objekt. Prije spremanja, kod provjerava postoji li već igrač unutar baze za određenog glavnog neprijatelja. Ako ne, sprema igrača i njegovo najbolje vrijeme u bazu. Ako da, tada provjerava je li novo izmjereno vrijeme bitke manje, odnosno bolje od onog spremljenog u bazi i ako je to slučaj, sprema u bazu. Sljedeća slika (Slika 21.) prikazuje izgled Firebasea s pohranjenim podacima iz videoigre, kao i primjer korištenja podataka iz baze na ploči naslovne scene video igre.



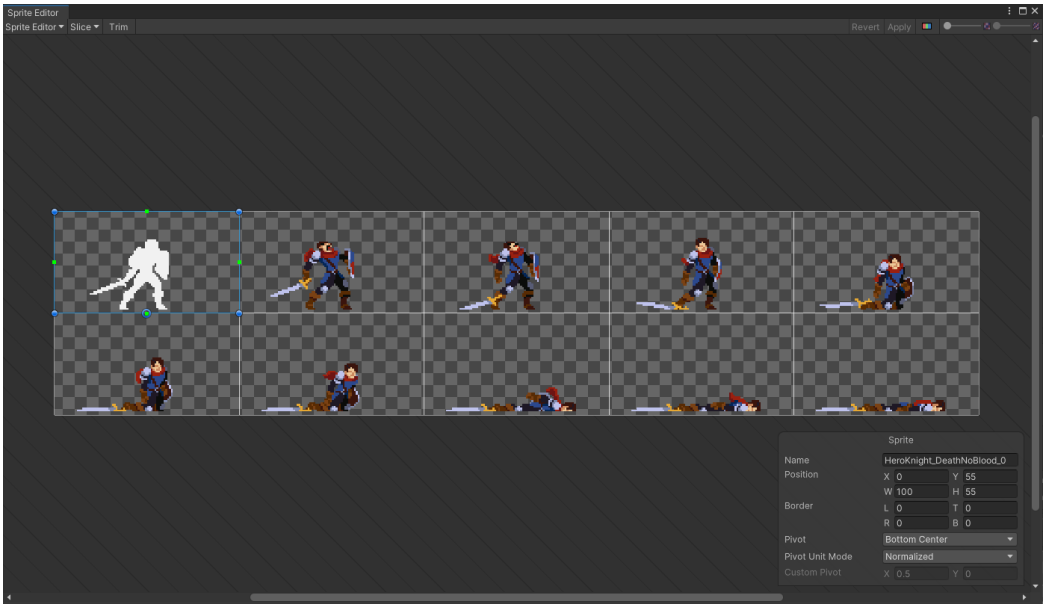
Slika 21. Pohranjeni podaci o igračima u Firebaseu te prikaz spremljenih podataka u videoigri [autorski rad]

10. Važne Unity komponente

Izrada videoigara ne temelji se samo na pisanju programskog koda, nego ona mora imati svoj izgled i zvuk. U sljedećim poglavljima predstavit će se važnije *Unity* komponente koje videoigrama daju identitet, a to su *Animation* i *Animator*, *Tile Palette*, *Particle System* i *Audio.*

10.1. Animation i Animator

Animation odnosno animacija najviše služi za stavljanje više „sprite-ova“, odnosno sličica u jednu vremensku crtu. Tamo se mora definirati vremenski razmak između sličica i tako dobiti pojedinu animaciju, primjerice hodanja palatina. Prije toga potrebno je nacrtati te slike ili ih preuzeti s interneta, najčešće u obliku „sprite sheeta“, što je jedna velika slika koja se sastoji od više manjih sličica. Te je sličice potrebno odvojiti, najčešće na način da se od velike slike napravi više slika sa sličicama koje zajedno prikazuju jednu vrstu animacije. Takve se slike stavlja unutar neke mape *Unitya,* označi ih se, klikne se na gumb „Sprite Editor“ nakon čega se otvori novi prozor za uređivanje sličica (Slika 22.). Na tom prozoru klikne se na „Slice“, odnosno izreži, definira se veličina sličica te se klikom na gumb „Slice“ izrežu sličice i klikne na „Apply“ kako bi se uradak spremio. Nakon toga je moguće ponovno označiti tu sliku iz mape te ju tehnikom povuci i ispusti mišem staviti u scenu igre. Na taj se način automatski kreira animacija i animator za te sličice.

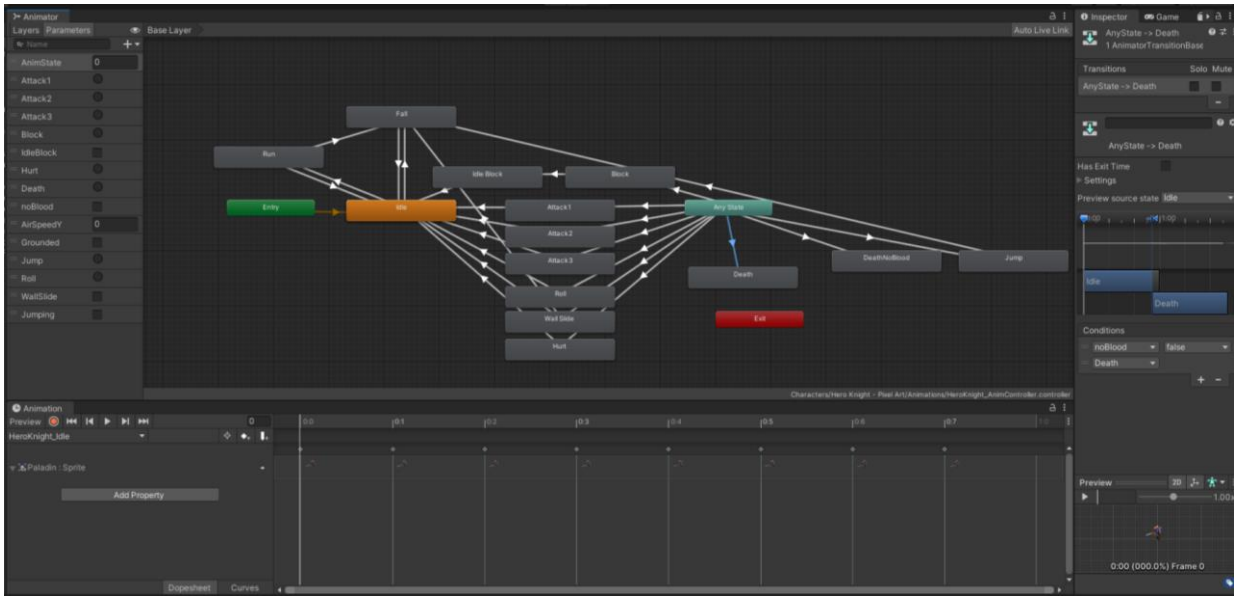


Slika 22. Izrezivanje sličica palatina unutar Sprite Editora [autorski rad]

Drugi način je da se u *Animation* prozoru klikne „Create“ za stvaranje *Animatora* i *Animation Clipa* (isječka animacije), klikne „Add Property“ i odabere „Sprite Renderer“. Unutar

vremenske crte postavljaju se sličice te se klikom na „Play the animation clip“ na glavnoj sceni igre može vidjeti dobivena animacija. Osim *Sprite renderera*, može se kao *property* postaviti i pozicija, rotacija *gameobjecta*, njegov *rigidbody* pa čak i skripta.

Nakon što se kreira dovoljan broj animacija, potrebno je dodati slijed, odnosno logiku iza tih animacija. Drugim riječima, želi se primjerice dobiti da nakon animacije skakanja prema gore slijedi animacija padanja, dok se animacija samog skakanja pokreće nakon što igrač pritisne tipku za skok. Izgled prozora za animiranje i slaganje animatora može se vidjeti na sljedećoj slici (Slika 23.):



Slika 23. Prozori *Animation* (dolje) i *Animator* (gore sredina) [autorski rad]

Kod skripte palatina već se vidjelo korištenje animatora kako bi se pokrenula animacija. Da bi se to postiglo, unutar animatora je bilo potrebno definirati parametre (na slici 23 se parametri vide s lijeve strane) koji služe kao uvjeti ili okidači te se određena animacija pokrene ukoliko su određeni uvjeti za to ispunjeni. Na primjer, da bi se pokrenula animacija prvog zamaha mačem palatina, mora se okinuti *trigger* „Attack1“, za što se u kodu trebala dohvatiti komponenta Animator i postaviti *trigger* „Attack1“:

```
m_animator.SetTrigger("Attack" + m_currentAttack);
```

Još će se objasniti pravokutnici i strelice unutar glavnog dijela animatora. Na slici 23 se vidi zeleni pravokutnik „Entry“ koji označava početak cijelog animatora. Žuti pravokutnik označava zadanu (engl. *default*) animaciju koja će se pokretati nakon zelenog. Crveni pravokutnik označava „Exit“, tj. izlaz te se nakon njega ponovno pokrene „Entry“. Sivi

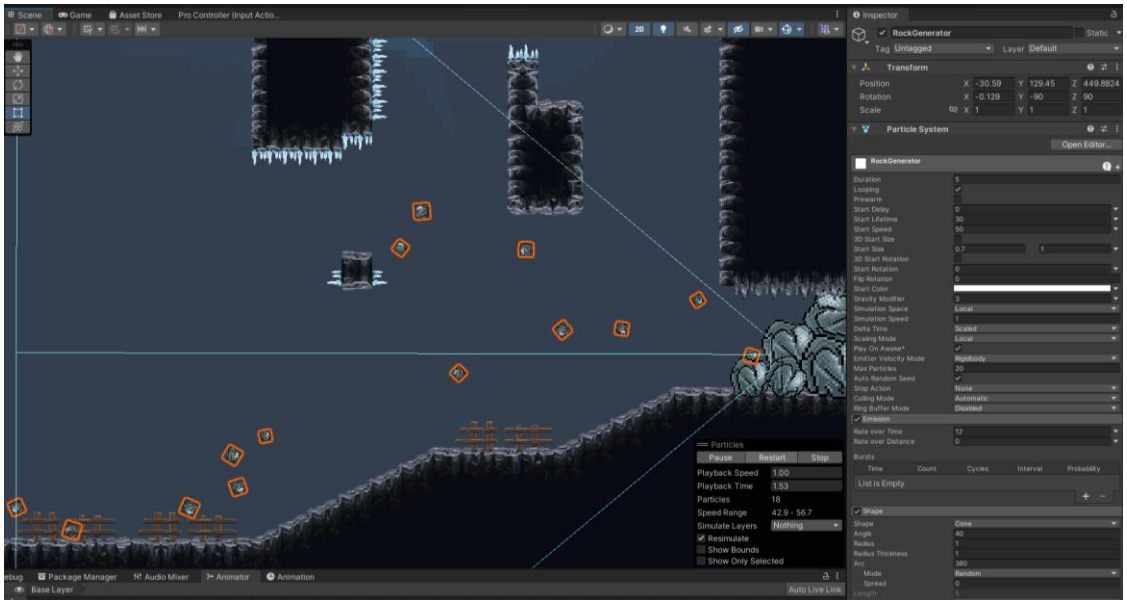
pravokutnici su ostale animacije, dok strelice između njih označavaju prijelaz između jedne animacije u drugu te se tamo postavljaju uvjeti koji moraju biti ispunjeni za prijelaz.

10.2. Paleta pločica (engl. Tile Palette)

Spomenut će se „Tile Palette“, u kojemu se unesu slike te se iste koriste za izgradnju, odnosno za crtanje svijeta videoigre. Pri otvaranju tog prozora stvori se mreža kockica u glavnom „Scene“ prozoru videoigre te se u te kockice mogu dodavati sličice [78].

10.3. Efekt čestica (engl. Particle Effects)

Efekti čestica su zapravo posebni efekti koje korisnici *Unityja* vole primjenjivati radi vizualnog obogaćivanja videoigre. Kreira se novi *gameobject* zajedno s komponentom „Particle System“ te se tamo mogu izraditi efekti različitih oblika i animacija. Glavne postavke efekata su koliko dugo će efekt biti prikazan prije negoli nestane, kolika mu je brzina, koja će mu biti veličina, početna boja i slično. Na mnoge postavke mogu se definirati dvije konstantne vrijednosti, nakon čega se pri stvaranju efekata nasumično odabere broj između tih dviju konstanta i kao vrijednost postavi za određenu postavku. Primjer za to je veličina efekta pri čemu će se tada za svaki efekt nasumično odabrati broj između konstanta i tako će postojati razlika između stvorenih efekata, što se može vidjeti na sljedećoj slici (Slika 24.).



Slika 24. Stvaranje malih kamenčića kojima je definirana kolizija s tlom [autorski rad]

Za efekte se može definirati i broj stvorenih čestica u jednoj sekundi, smanjenje/povećanje brzine, veličine, rotacije tijekom života čestice, teksture, kolizije pa čak i

rigidbody. Drugim riječima, korisnici *Unitya* zbilja mogu biti kreativni i na razne načine postići efekte svakakvih vrsta.

10.4. Zvučni efekti

Zvuk videoigre, bilo da se radi o kratkim zvučnim efektima (engl. *Sound Effects*, SFX) ili o glazbi koja svira tijekom cijele borbe, postiže se koristeći komponentu „Audio Source“, što u prijevodu znači izvor zvuka. U njega se definira AudioClip, odnosno zvučni efekt koji se u nekom trenutku igre želi odsvirati. Kako bi se izbjeglo učestalo dohvaćanje te komponente *gameobjecta* unutar svake skripte, kreirala se jedna skripta pod nazivom „SingletonSFX.cs“. Na tu se skriptu primijenio uzorak dizajna *Singleton*. *Singleton* je objektni uzorak kreiranja koji pruža globalni pristup do klase, onemogućava pristup konstruktoru klase drugim klasama te osigurava da klasa ima samo jednu jedinu instancu [64] [79, slajd 5]. Kreiranje samo jedne instance postiglo se na sljedeći način:

```
public static SingletonSFX Instance { get; private set; }

    // Called before Start() method
    // Destroy a duplicate instance if another one already exists
    private void Awake()
    {
        if (Instance != null
            && Instance != this)
        {
            Destroy(this);
        }
        else {
            Instance = this;
        }
    }
```

Ovdje se ključnim riječima definirala javna statička varijabla „Instance“ tipa SingletonSFX te njezin *geter* i *seter*. Unutar Awake() metode provjerava se postoji li već duplikat instance. Ako je odgovor da, obriše se. Time se osigurava da postoji samo jedna instanca klase. Nadalje, metodu koja ta klasa nudi jest „PlaySFX()“ kojoj se prosljeđuje naziv zvučnog efekta kako bi ju se učitalo iz datoteke „Resources“, a onda jednom odsviralo [80] [81]:

```
    // Plays one time SFX
    public void PlaySFX(string name)
    {
        resourceSFX = Resources.Load<AudioClip>(name);
        sourceOfSFX.PlayOneShot(resourceSFX);
    }
```

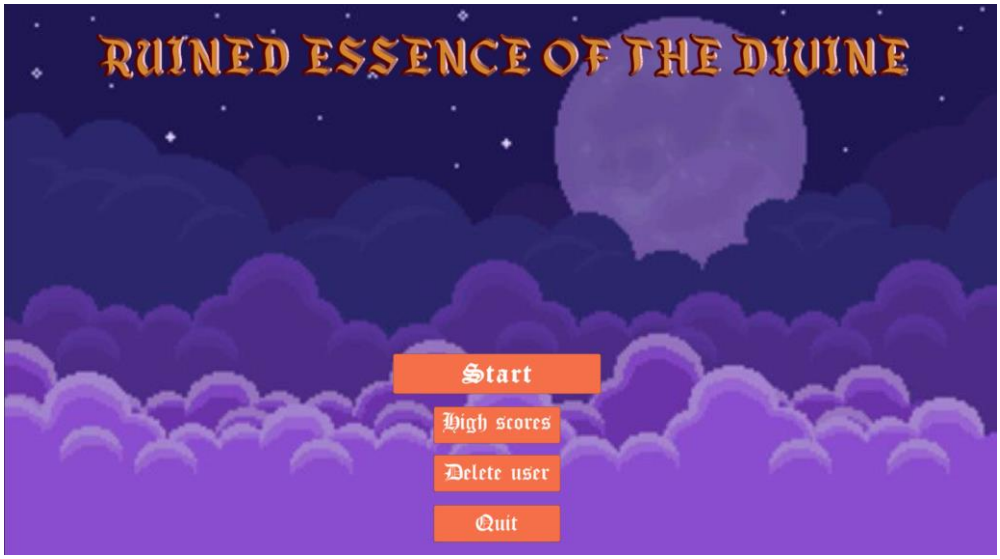
11. Početna scena i svijet za odmaranje

Nakon što pokrene videoigru, igrač mora unijeti novo korisničko ime ako prvi put igra ili je obrisao postojeće korisničko ime. Ime se sprema koristeći klasu „PlayerPrefs“ koja može spremati podatke tipa *string*, *float* i *integer* između sesija igranja [82]. Spremanje je nužno jer se taj podatak koristi u drugim scenama *Unityja*, odnosno za spremanje imena igrača i najboljeg vremena u bazu, stoga je važno pohraniti ga. Taj proces se događa nakon što igrač klikne na gumb „Start“ jer se tada pozivaju sljedeće metode klase „TitleScreen.cs“:

```
public void StartTheGame()
{
        SceneManager.LoadScene("HubWorld");
}

private void OnDisable()
{
       PlayerPrefs.SetString("playerName", playerNameInp.text);
}
```

Metoda „StartTheGame()“ poziva se nakon što se klikne gumb u naslovnoj sceni. Tada se koristi metoda „LoadScene()“ za učitavanje scene igre, u ovom slučaju svijeta za odmaranje [83]. Prilikom učitavanja druge scene, u prvoj sceni *gameobjecti* postanu neaktivni. Prije nego se to dogodi, sprema se u PlayerPrefs unutar varijable „playerName“ tipa *string* vrijednost koju je igrač unio u polje za unos teksta („playerNameInp.text“). To se događa u „OnDisable()“ metodi koja se svaki put poziva prije nego što se prva scena deaktivira, konkretnije prije nego što se klase koje nasljeđuju „MonoBehaviour“ deaktiviraju [84]. Ostale mogućnosti koje igrač ima na naslovnoj sceni osim unosa imena i pokretanja igre su pregled najboljih vremena protiv glavnih neprijatelja, brisanje svojeg korisničkog imena i izlaz iz igre (Slika 25.).



Slika 25. Naslovna scena videoigre: *Ruined Essence of the Divine* [autorski rad]

Nakon što se klikne na gumb „Start“, igrač može vidjeti prvi svijet, odnosno sobu za odmaranje. Tamo se nalaze vrata u koja može ući i boriti se protiv glavnih neprijatelja. U početku nisu sva vrata otključana, nego su zatamnjena te se otključaju nakon što igrač pobijedi određenog glavnog neprijatelja. Stoga su samo prva vrata koja vode do Behemota otključana, što se i može vidjeti na sljedećoj slici (Slika 26.).

Slika 26. Svijet za odmaranje [autorski rad]

Kada igrač uđe u određenu arenu i pobjedi glavnog neprijatelja, dohvaća se spremljeno korisničko ime iz PlayerPrefs za unos u bazu podataka. Nakon što igrač pobjedi svih 5 glavnih neprijatelja, igra je završena.

12. Zaključak

Programski alat *Unity* izabran za ovaj rad nije bio nepoznanica. Upoznalo ga se tijekom izrade videoigre „Knivader“ žanra *Metroidvania* za završni rad. Zato je odlučeno ponovno koristiti *Unity*, no ovoga puta išlo se dublje proučavati alat i njegove razne mogućnosti koje prije nisu korištene. Primjerice, prvi put koristila se baza podataka preko API-ja, upoznalo se i koristilo novi i bolji sistem unosa naredaba iz tipkovnica, miša i kontrolera te se naučilo kako implementirati podršku za više igrača.

Izrada videoigre *Ruined Essence of the Divine* bila je zahtjevna upravo zato što pripada *Boss Rush* žanru. Za izradu kvalitetne igre toga žanra potrebno je osmisliti izgled prostora za svaku bitku, razne napade glavnog neprijatelja, stvaranje ili pronalaženje prikladnih slika i zvučnih efekata, pisanje složenijeg programskog koda za glavne neprijatelje te sve zajedno ukomponirati na način da je konačni proizvod igračima zabavan i zanimljiv za odigrati. Sve je to vrijedilo i za ovu igru, a prilikom testiranja se često dogodilo da nešto nije radilo onako kako je bilo zamišljeno, no na kraju su se uvijek uspješno pronalazila rješenja.

Za ovaj se rad primjenjivalo mnogo toga što se naučilo primjenjivati tijekom obrazovanja na fakultetu i na radnom mjestu. Pisanjem sažetka za svaku metodu čitatelju koda se na bolji način daje do znanja što metoda radi. Nadalje, vodio se račun da svaka metoda ima manje od 30 logičkih linija koda, odnosno linija koje se gledaju kao zasebne izjave te nešto izvršavaju. Isto tako, odlučilo se primijeniti bar par uzoraka dizajna jer se pomoću njih, ako se na dobrom mjestu i pravilno iskoriste, problem rješava na vrlo praktičan način. Također se pripazilo da se piše takav kod čije izvođenje neće biti sporo te da ne zauzima previše memorijskog prostora. Posljedica svega spomenutog je lako čitljiv i učinkovit programski kod.

Istaknulo bi se da je izrada projekata ovakvih vrsta na kraju krajeva uvijek poučna i zabavna. Ovdje se nisu testirale samo vještine pisanja dobrog koda, nego i kreativne strane *developera*. Stoga proces razvoja nije ni u jednom trenutku bio monoton zbog raznih ideja za videoigru koje su dolazile i koje je bilo potrebno ostvariti. Uvijek se prilikom ostvarivanja tih ideja primjenjivalo nešto novo.

Projekt pokazuje na to da videoigre nisu jednostavne za izraditi te da oduzmu mnogo vremena, ali se pritom iz njih može mnogo toga naučiti te je sam proces razvoja obično i zabavan. Stoga bih svakome preporučio da tijekom svoje karijere programiranja izrade bar jednu složeniju videoigru.

13. Popis slika

14. Popis korištenih resursa

14.1. Korišteni programi i programski alati

1. Programski alat: Unity,
   https://unity3d.com/get-unity/download
   Dostupno 27.6.2022.
2. Integrirano razvojno okruženje: Microsoft Visual Studio Professional 2022,
   https://visualstudio.microsoft.com/downloads/
   Dostupno 27.6.2022.
3. Web stranica za preuzimanje YouTube video sadržaja u obliku audio datoteka: tuberipper.com,
   https://tuberipper.com/
   Dostupno: 27.6.2022.
4. Web stranica za skraćivanje audio datoteka - audiotrimmer.org,
   https://audiotrimmer.com/
   Dostupno 27.6.2022.
5. Grafički računalni program: Adobe Photoshop,
   https://www.adobe.com/products/photoshop/free-trial-download.html
   Dostupno 27.6.2022.
6. Program za mogućnost korištenja kontrolera raznih vrsta na računalu i za izmjenu naredba tipaka: reWASD,
   https://www.rewasd.com/
   Dostupno 27.6.2022.
7. Ugradnja Rest Client API-ja za Unity: Unity Package Manager,
   https://assetstore.unity.com/packages/tools/network/rest-client-for-unity-102501
   Dostupno 27.6.2022.
8. Ugradnja cjelovitog JSON deserializera u projekt: GitHub.com,
   https://github.com/jacobdufault/fullserializer,
   Dostupno 27.6.2022.
9. Ugradnja Cinemachine za Unity: Unity Package Manager,
   Dostupno 27.6.2022.
10. Ugradnja TextMeshPro paketa za Unity: Unity Package Manager,
    https://docs.unity3d.com/Packages/com.unity.textmeshpro@3.0/manual/index.html
    Dostupno 27.6.2022.

14.2. Glazba i zvučni efekti

1. Glazba naslovne scene: Mattashi – The Final Battle [Epic 8-bit Orchestral Battle] – ThePrimeCronus,
   https://www.youtube.com/watch?v=ICwnlNitmUw
   Dostupno 27.6.2022.
2. Glazba sobe za odmor i odabir glavnih neprijatelja: Kirby's Return to Dream Land Wii OST – Philip J. Fry II,
   https://www.youtube.com/watch?v=MJAObIDnUtc
   Dostupno 27.6.2022.
3. Glazba arene Fern Behemotha: Super Mario 64 Music - Ultimate Bowser (Final Bowser Fight) – SilverShadowMusic,
   https://www.youtube.com/watch?v=SCrCmlrmHqg&t=81s
   Dostupno 27.6.2022.

4. Glazba arene Psychic Psycho: Metroid Prime 3: Corruption Music - Gandrayda Boss Theme – Metroid Music Channel, https://www.youtube.com/watch?v=_rHnfNmiAcM Dostupno 27.6.2022.
5. Glazba arene Chained Undeada: Legend of Zelda: Twilight Princess - Light and Darkness (Zant's Theme) - YamiMarik1994, https://www.youtube.com/watch?v=wugDsU6g8BA Dostupno 27.6.2022.
6. Glazba arene Sidus Istara: Kirby Super Star Remastered - Vs. Marx – Church of Kondo, https://www.youtube.com/watch?v=ik-DGGuyFj0&t=6s Dostupno 27.6.2022.
7. Glazba Arene Glacial Overlorda: Rundas Theme, Sad Edit. (No long final portion) – Andyblarg's Music, https://www.youtube.com/watch?v=-vP8H6NTJCA Dostupno 27.6.2022.
8. Zvučni efekt: prvi combo mača – freesound.org, https://freesound.org/people/beerbelly38/sounds/362350/ Dostupno 27.6.2022.
9. Zvučni efekt: drugi combo mača, https://freesound.org/people/beerbelly38/sounds/362349/ Dostupno 27.6.2022.
10. Zvučni efekt: treći combo mača, https://freesound.org/people/DragonTrance/sounds/375997/ Dostupno 27.6.2022.
11. Zvučni efekt: kotrljanje palatina po tlu – reddit.com, https://www.reddit.com/r/darksouls/comments/3fskm9/a_reupload_of_some_of_the_more_common_dark_souls/ Dostupno 27.6.2022.
12. Zvučni efekt: podizanje štita – freesound.org, https://freesound.org/people/JapanYoshiTheGamer/sounds/361256/ Dostupno 27.6.2022.
13. Zvučni efekt: palatin je ozlijeđen – freesound.org, https://freesound.org/people/jeckkech/sounds/391668/ Dostupno 27.6.2022.
14. Zvučni efekt: palatin umire – freesound.org, https://freesound.org/people/xtrgamr/sounds/432875/ Dostupno 27.6.2022.
15. Zvučni efekt: zvuk ozljede/krvi – freesound.org, https://freesound.org/people/Pablobd/sounds/511194/ Dostupno 27.6.2022.
16. Zvučni efekt: palatin skače – freesound.org, https://freesound.org/people/deleted_user_10023915/sounds/478051/ Dostupno 27.6.2022.
17. Zvučni efekt: krikovi Fern Behemotha – Evolution of Ridley's Voice (1986-2018) – Samuel, https://www.youtube.com/watch?v=oyUc3xQ3pcQ Dostupno 27.6.2022.
18. Zvučni efekt: hodanje i trčanje Fern Behemotha - sounds-resource-com, https://www.sounds-resource.com/nintendo_64/mario64/sound/1441/ Dostupno 27.6.2022.
19. Zvučni efekt: riganje vatre Fern Behemotha - sounds-resource-com, https://www.sounds-resource.com/nintendo_64/mario64/sound/1441/ Dostupno 27.6.2022.
20. Zvučni efekt: požar – SFX izvađen iz video igre „Castlevania: Symphony of the Night“

21. Zvučni efekti:
    - ozljeda glavnih neprijatelja projektilom
    - magičan napad Psychic Psycho
    - zbunjivanje palatina
    - teleportacija koju koristi Psychic Psycho
    - rotiranje kamere
    - visoko skakanje palatina
    - oštar napad pera
    - odbijanje projektila štitom
    - kratki zvuk eksplozije
    - dugi zvuk eksplozije
    - vitez ne može skakati
    - vitez ponovno može skakati
    - ozljeda Sidus Istara
    - odbijanje žute zvijezde štitom
    - ispaljivanje žute zvijezde
    - ispaljivanje plave zvijezde
    - ispaljivanje kugle tamne tvari
    - ispaljivanje ledenih kristala
    - pad ledenih siga
    - ledeno teleportiranje
    - liječenje glavnog neprijatelja
    - snježna mećava

    – sounds-resource.com, https://www.sounds-resource.com/wii/kirbyrtdl/sound/1139/ Dostupno 27.6.2022.
22. Zvučni efekt: kiša – freesound.org, https://freesound.org/people/EVRetro/sounds/519071/ Dostupno 27.6.2022.
23. Zvučni efekt: umiranje anđela – sounds-resource.com, https://www.sounds-resource.com/playstation/castlevaniasymphonyofthenight/sound/41894/ Dostupno 27.6.2022.
24. Zvučni efekt: uništen dio tijela Chained Undeada – sounds-resource.com, https://www.sounds-resource.com/playstation/castlevaniasymphonyofthenight/sound/41886/ Dostupno 27.6.2022.
25. Zvučni efekt: ulazak muhe u arenu – freesound.org, https://freesound.org/people/Benboncan/sounds/81970/ Dostupno 27.6.2022.
26. Zvučni efekti:
    - jurišanje
    - eksplozija
    - umiranje muhe
    - napad otrovom

    – dkc.atlas.com, http://www.dkc-atlas.com/forum/viewtopic.php?t=1604 Dostupno 27.6.2022.
27. Zvučni efekt: kostur uništen – freesound.org, https://freesound.org/people/personwhois/sounds/458974/ Dostupno 27.6.2022.
28. Zvučni efekt: uništen otrov muhe – freesound.org, https://freesound.org/people/javapimp/sounds/439185/

Dostupno 27.6.2022.
29. Zvučni efekt: lanci – freesound.org, https://freesound.org/people/ani_music/sounds/167914/ Dostupno 27.6.2022.
30. Zvučni efekt: zvukovi Chained Undeada – Mother Brain Roars-Super Metroid – MAXIMUMGODZILLA45, https://www.youtube.com/watch?v=751mxHpHdlo Dostupno 27.6.2022.
31. Zvučni efekt: kotrljanje velikog kamenja – sounds-resource.com, https://www.sounds-resource.com/snes/legendofzeldaalinktothepast/ Dostupno 27.6.2022.
32. Zvučni efekt: lišće – freesound.org, https://freesound.org/people/nextmaking/sounds/86015/ Dostupno 27.6.2022.
33. Zvučni efekt: mačevanje – freesound.org, https://freesound.org/people/JohnBuhr/sounds/326803/ Dostupno 18.7.2022.
34. Zvučni efekt velike ozljede: Two mask damage sound – Hollow Knight – Link9058 https://www.youtube.com/watch?v=cS7cLq4-ebY Dostupno 18.7.2022.

14.3. Font, slike okoline, protagonist i slike neprijatelja

1. Font: Old London – Dieter Steffmann, https://www.dafont.com/old-london.font Dostupno 27.6.2022.
2. Font: Enchanted Land – Creativework69, https://www.dafont.com/enchanted-land.font Dostupno 27.6.2022.
3. Pozadinska slika naslovne scene: Unity Asset Store, https://assetstore.unity.com/packages/2d/environments/2d-pixel-art-background-10-sky-cloud-87800 Dostupno 27.6.2022.
4. Pravokutnici glavnog menija: runica.itch.io, https://runica.itch.io/pixel-menu-box-buttons Dostupno 27.6.2022.
5. Slike dekoracija i tileseta arene Fern Behemotha: Unity Asset Store, https://assetstore.unity.com/packages/2d/environments/crystal-world-platformer-150016 Dostupno 27.6.2022.
6. Slike dekoracija i tileseta arene i slike glavnog neprijatelja Psychic Psycho: Unity Asset Store, https://assetstore.unity.com/packages/2d/characters/gothicvania-church-pack-147117 Dostupno 27.6.2022.
7. Slike dekoracija i tileseta arene Chained Undeada i slike kostura: Unity Asset Store, https://assetstore.unity.com/packages/2d/characters/gothicvania-cemetery-120509 Dostupno 27.6.2022.
8. Slike šume arene Sidus Istara: Uniy Asset Store, https://assetstore.unity.com/packages/2d/textures-materials/nature/free-pixel-art-forest-133112 Dostupno 27.6.2022.
9. Slike, animacije, skripte glavnog protagonista palatina: Unity Asset Store, https://assetstore.unity.com/packages/2d/characters/hero-knight-pixel-art-165188

Dostupno 27.6.2022.
10. Slike Fern Behemotha: pngwing.com, https://www.pngwing.com/en/free-png-irzao Dostupno 27.6.2022.
11. Slika zelene vatre Fern Behemotha: YouTube, https://i.ytimg.com/vi/dsZNPtiOcRI/hqdefault.jpg Dostupno 27.6.2022.
12. Slika pozadine Fern Behemotha: pixelartmaker.com, https://pixelartmaker-data-78746291193.nyc3.digitaloceanspaces.com/image/37d1b971d1d16d6.png Dostupno 27.6.2022.
13. Slike kamenja: davidepesce.com, https://www.davidepesce.com/2020/03/28/nature-pixel-art-tutorial-2-stones-and-rocks/ Dostupno 27.6.2022.
14. Slika šumske pozadine arene Psychic Psycho: pinterest.com, https://i.pinimg.com/564x/2a/6a/95/2a6a95a42d6b8dbecd16f0d2c8bf8c75.jpg Dostupno 27.6.2022.
15. Slika okrugle telepatijske moći Psychic Psycho: Unity Asset Store, https://assetstore.unity.com/packages/2d/characters/2d-pixel-art-pack-rousette-167698 Dostupno 27.6.2022.
16. Slika oštrog pera: pixelartmaker.com, http://pixelartmaker.com/art/352c1a84f405104 Dostupno 27.6.2022.
17. Slika opruge: dreamstime.com, https://www.dreamstime.com/vector-pixel-art-spring-isolated-cartoon-image162283636 Dostupno 27.6.2022.
18. Slika tornada: svgrepo.com, https://www.svgrepo.com/svg/279712/tornado Dostupno 27.6.2022.
19. Slika oblaka prašine: vectorstock.com, https://www.vectorstock.com/royalty-free-vector/pixel-art-smoke-animation-frames-for-game-vector-8484972 Dostupno 27.6.2022.
20. Slike Chained Undeada, njegovih muha i crva: spriters-resource.com, https://www.spriters-resource.com/playstation/cvsotn/sheet/3438/ Dostupno 27.6.2022.
21. Slika mjehurića: pixelartmaker.com, http://pixelartmaker.com/art/21d85b0e1d00cd5 Dostupno 27.6.2022.
22. Slike Sidus Istara i njegovih zvijezda: spriters-resource.com, https://www.spriters-resource.com/game_boy_advance/kirbynim/sheet/3012/ Dostupno 27.6.2022.
23. Slika grane s lišćem: pixilart.com, https://www.pixilart.com/art/pixel-vine-63f8f96334ca616 Dostupno 27.6.2022.
24. Slika velikog kamena: dreamstime.com, https://www.dreamstime.com/rock-stone-boulder-pixel-art-eight-bit-retro-video-game-style-icon-rock-stone-boulder-pixel-art-eight-bit-game-icon-image195967102 Dostupno 27.6.2022.
25. Slika planine: pinterest.com, https://www.pinterest.com/pin/16677461101191219/ Dostupno 27.6.2022.

26. Slika oblaka: shutterstock.com,
https://www.shutterstock.com/pt/image-vector/clouds-set-pixel-art-style-vector-629858270
Dostupno 27.6.2022.
27. Slika svemira: opensea.io,
https://opensea.io/assets/matic/0x2953399124f0cbb46d2cbacd8a89cf0599974963/103089569026097952035939975308051808844940331763834678870114587691086419853412
Dostupno 27.6.2022.
28. Slika nebule: pinterest.com,
https://www.pinterest.com/pin/480548222709345067/
Dostupno 27.6.2022.
29. Slika ledene špilje: artstation.com,
https://www.artstation.com/artwork/zAyO5D
Dostupno 27.6.2022.
30. Slika leteće ledene sige: pixelworlds.fancom.com,
https://pixelworlds.fandom.com/wiki/Frost_Shard
Dostupno 1.7.2022.
31. Slika padajućih siga: emojisky.com,
https://www.emojisky.com/desc/7832083
Dostupno: 18.7.2022.

15. Literatura

[1] „Why Do We Call The Hardest Video Game Enemies ‚Bosses,' Anyway?", *Kotaku*, 19. veljača 2021. https://kotaku.com/why-do-we-call-the-hardest-video-game-enemies-bosses-a-1846301973 (pristupljeno 20. lipanj 2022.).

[2] T. Lee, „An annotated history of video game boss battles", *Polygon*, 28. rujan 2015. https://www.polygon.com/features/2015/9/28/9333685/annotated-history-boss-battles (pristupljeno 20. lipanj 2022.).

[3] „The Legend of Zelda", *Nintendo of Europe GmbH*. https://www.nintendo.co.uk/Games/NES/The-Legend-of-Zelda-796345.html (pristupljeno 20. lipanj 2022.).

[4] How to play Contra – NES Manual, Konami Inc. 1987. [Na internetu]. http://www.thealmightyguru.com/Wiki/images/5/5a/Contra_-_NES_-_Manual.pdf (pristupljeno 20. lipanj 2022.).

[5] „Mega Man (1987 video game)", Wikipedia. 21. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Mega_Man_(1987_video_game)&oldid=1094283280 (pristupljeno 27. lipanj 2022.).

[6] „25 Best SNES Games of All Time", *Den of Geek*, 14. ožujak 2019. https://www.denofgeek.com/games/best-snes-games/ (pristupljeno 20. lipanj 2022.).

[7] „Kirby's Dream Land for Nintendo 3DS - Nintendo". https://www.nintendo.com/store/products/kirbys-dream-land-3ds/ (pristupljeno 20. lipanj 2022.).

[8] „10 Best Castlevania Games Ever Made", *Den of Geek*, 26. rujan 2019. https://www.denofgeek.com/games/10-best-castlevania-games-ever-made/ (pristupljeno 20. lipanj 2022.).

[9] „Shadow of the Colossus Review", *GameSpot*. https://www.gamespot.com/reviews/shadow-of-the-colossus-review/1900-6135831/ (pristupljeno 20. lipanj 2022.).

[10] Shadow of the Colossus [Slika] (9.2.2018.) Dostupno: https://www.wired.com/story/shadow-of-the-colossus-remastered-review/ (pristupljeno 20.6.2022.).

[11] „Souls on Steam". https://store.steampowered.com/app/297130/Titan_Souls/ (pristupljeno 20. lipanj 2022.).

[12] „Save 80% on Jotun: Valhalla Edition on Steam". https://store.steampowered.com/app/323580/Jotun_Valhalla_Edition/ (pristupljeno 20. lipanj 2022.).

[13] „Cuphead: Don't Deal With The Devil | Available on Xbox One - Windows 10 - Nintendo Switch – PlayStation 4 - Steam - GOG - Mac". https://cupheadgame.com/ (pristupljeno 20. lipanj 2022.).

[14] *Top 10 BEST Boss Rush Indie Games*. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=vMObrVg07BY (pristupljeno 20. lipanj 2022.).

[15] Cuphead multiplayer [Slika] (25.11.2020) Dostupno: https://i0.wp.com/atorredecontrole.com.br/wp-content/uploads/2020/11/cuphead.jpg?w=1280&ssl=1 (pristupljeno 20. lipanj 2022.).

[16] „Unreal Engine 5“, *Unreal Engine*. https://www.unrealengine.com/en-US/unreal-engine-5 (pristupljeno 21. lipanj 2022.).

[17] U. Technologies, „Unity Real-Time Development Platform | 3D, 2D VR & AR Engine“. https://unity.com/ (pristupljeno 21. lipanj 2022.).

[18] U. Technologies, „Learn How to Code in C# for Beginners | Unity Learn“. https://unity.com/how-to/learning-c-sharp-unity-beginners (pristupljeno 21. lipanj 2022.).

[19] G. Engine, „Godot Engine - Free and open source 2D and 3D game engine“, *Godot Engine*. https://godotengine.org/ (pristupljeno 21. lipanj 2022.).

[20] „Easily Make Video Games with GameMaker“, *GameMaker*. https://gamemaker.io/en/gamemaker (pristupljeno 21. lipanj 2022.).

[21] „Unreal Engine vs CryEngine | 9 Most Valuable Differences To Learn“, *EDUCBA*, 29. studeni 2018. https://www.educba.com/unreal-engine-vs-cryengine/ (pristupljeno 21. lipanj 2022.).

[22] „Rider for Unreal Engine“, *JetBrains: Developer Tools for Professionals and Teams*. https://www.jetbrains.com/lp/rider-unreal (pristupljeno 21. lipanj 2022.).

[23] „Unity Games Development Tools“, *Visual Studio*. https://visualstudio.microsoft.com/vs/unity-tools/ (pristupljeno 21. lipanj 2022.).

[24] „Using MySQL with Unity“, *Simple Talk*, 15. travanj 2021. https://www.red-gate.com/simple-talk/development/dotnet-development/using-mysql-unity/ (pristupljeno 21. lipanj 2022.).

[25] „Saving Data in Unity3D Using SQLite | MongoDB“. https://www.mongodb.com/developer/code-examples/csharp/saving-data-in-unity3d-using-sqlite/ (pristupljeno 21. lipanj 2022.).

[26] „Add Firebase to your Unity project | Firebase Documentation“, *Firebase*. https://firebase.google.com/docs/unity/setup (pristupljeno 21. lipanj 2022.).

[27] „MongoDB For Gaming“, *MongoDB*. https://www.mongodb.com/use-cases/gaming (pristupljeno 21. lipanj 2022.).

[28] „What Is Extended Reality? Everything You Need To Know“, *Roundtable Learning*, 08. srpanj 2021. https://roundtablelearning.com/what-is-extended-reality-everything-you-need-to-know/ (pristupljeno 21. lipanj 2022.).

[29] U. Technologies, „Government & Aerospace Simulation Training Software | Unity“. https://unity.com/solutions/government-aerospace (pristupljeno 21. lipanj 2022.).

[30] HAAS, John K. A history of the unity game engine. *Diss. WORCESTER POLYTECHNIC INSTITUTE*, 2014, 483: 484. (pristupljeno 21. lipanj 2022.).

[31] „Languages“, *Visual Studio*. https://visualstudio.microsoft.com/vs/features/web/languages/ (pristupljeno 21. lipanj 2022.).

[32] „Visual Studio IntelliCode | Visual Studio“, *Visual Studio*. https://visualstudio.microsoft.com/services/intellicode/ (pristupljeno 21. lipanj 2022.).

[33] „What is IntelliCode/IntelliSense in VS Code?“, *Visual Studio 101*. https://visualstudio101.com/intellicode-intellisense (pristupljeno 21. lipanj 2022.).

[34] „Firebase“, *Firebase*. https://firebase.google.com/ (pristupljeno 21. lipanj 2022.).

[35] J. Clark, „Firebase vs. Firestore | What are the differences?“, bez dat. [Na internetu]. Dostupno: https://blog.back4app.com/firebase-vs-firestore/ (pristupljeno 21. lipanj 2022.).

[36] „Google Acquires Firebase To Help Developers Build Better Real-Time Apps“, *TechCrunch*. https://social.techcrunch.com/2014/10/21/google-acquires-firebase-to-help-developers-build-better-realtime-apps/ (pristupljeno 21. lipanj 2022.).

[37] L. Rožić, „Izrada video igre žanra Metroidvania u programskom alatu Unity“, info:eu-repo/semantics/bachelorThesis, University of Zagreb. Faculty of Organization and Informatics. Department of Theoretical and Applied Foundations of Information Sciences, 2020. [Na internetu]. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:211:139263 (pristupljeno 21. lipanj 2022.).

[38] „Unity - Manual: GameObject“. https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html (pristupljeno 21. lipanj 2022.).

[39] „Unity - Manual: Using Components“. https://docs.unity3d.com/560/Documentation/Manual/UsingComponents.html (pristupljeno 21. lipanj 2022.).

[40] „Unity - Manual: Colliders“. https://docs.unity3d.com/560/Documentation/Manual/CollidersOverview.html (pristupljeno 21. lipanj 2022.).

[41] „Unity - Manual: Prefabs“. https://docs.unity3d.com/560/Documentation/Manual/Prefabs.html (pristupljeno 21. lipanj 2022.).

[42] U. Technologies, „Unity - Scripting API: MonoBehaviour“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (pristupljeno 21. lipanj 2022.).

[43] U. Technologies, „Unity - Scripting API: MonoBehaviour.Start()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html (pristupljeno 21. lipanj 2022.).

[44] U. Technologies, „Unity - Scripting API: MonoBehaviour.Awake()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html (pristupljeno 21. lipanj 2022.).

[45] U. Technologies, „Unity - Scripting API: MonoBehaviour.Update()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html (pristupljeno 21. lipanj 2022.).

[46] U. Technologies, „Unity - Scripting API: MonoBehaviour.FixedUpdate()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html (pristupljeno 21. lipanj 2022.).

[47] „Cinemachine for 2D: Tips and Tricks“, *Unity Blog*. https://blog.unity.com/technology/cinemachine-for-2d-tips-and-tricks (pristupljeno 21. lipanj 2022.).

[48] U. Technologies, „Unity - Scripting API: SerializeField“. https://docs.unity3d.com/ScriptReference/SerializeField.html (pristupljeno 22. lipanj 2022.).

[49] U. Technologies, „Unity - Scripting API: MonoBehaviour.OnTriggerEnter2D(Collider2D)“.

https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html (pristupljeno 22. lipanj 2022.).

[50] U. Technologies, „Unity - Scripting API: Mathf.Clamp“. https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html (pristupljeno 22. lipanj 2022.).

[51] U. Technologies, „Unity - Scripting API: MonoBehaviour.StartCoroutine“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html (pristupljeno 22. lipanj 2022.).

[52] U. Technologies, „Unity - Scripting API: Time.deltaTime“. https://docs.unity3d.com/ScriptReference/Time-deltaTime.html (pristupljeno 22. lipanj 2022.).

[53] „Changelog | Input System | 1.3.0“. https://docs.unity3d.com/Packages/com.unity.inputsystem@1.3/changelog/CHANGELOG.html (pristupljeno 22. lipanj 2022.).

[54] *CONTROLLER INPUT in Unity!*, [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=p-3S73MaDP8 (pristupljeno 23.lipanj 2022.).

[55] „C# | Sealed Class“, *GeeksforGeeks*, 13. srpanj 2018. https://www.geeksforgeeks.org/c-sharp-sealed-class/ (pristupljeno 23. lipanj 2022.).

[56] *Local Multiplayer with NEW Input System - Unity Tutorial*. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=g_s0y5yFxYg (pristupljeno 23. lipanj 2022.).

[57] „Remap Xbox One controller with powerful gamepad mapper“. https://www.rewasd.com/ (pristupljeno 23. lipanj 2022.).

[58] J. Farrell, Microsoft Visual C# 2010 An Introduction to Object-oriented programming, Boston, SAD, Course Technology. 2011.

[59] BillWagner, ?: „operator - C# reference“. https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator (pristupljeno 23. lipanj 2022.).

[60] U. Technologies, „Unity - Scripting API: Random.Range“. https://docs.unity3d.com/ScriptReference/Random.Range.html (pristupljeno 23. lipanj 2022.).

[61] U. Technologies, „Unity - Scripting API: Quaternion.identity“. https://docs.unity3d.com/ScriptReference/Quaternion-identity.html (pristupljeno 23. lipanj 2022.).

[62] U. Technologies, „Unity - Scripting API: Quaternion.Euler“. https://docs.unity3d.com/ScriptReference/Quaternion.Euler.html (pristupljeno 23. lipanj 2022.).

[63] „About Cinemachine | Cinemachine | 2.8.6“. https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/index.html (pristupljeno 24. lipanj 2022.).

[64] C. G. Lasater, Design Patterns, Plano Texas, SAD, Worldwide Publishin Inc. 2007.

[65] D. Kermek, „Uzorci dizajna za ponašanje“, materijali s predavanja, FOI, https://elf.foi.hr/pluginfile.php/13875/mod_resource/content/16/predavanja/Kermek_UzDiz_05.pdf (pristupljeno 24. lipanj 2022.).

[66] „UML Class Diagram Tutorial“. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/ (pristupljeno 24. lipanj 2022.).

[67] „IBM Docs“, 02. ožujak 2021. https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-generalization-relationships (pristupljeno 24. lipanj 2022.).

[68] „Wizards“, The One Wiki to Rule Them All. https://lotr.fandom.com/wiki/Wizards (pristupljeno 24. lipanj 2022.).

[69] U. Technologies, „Unity - Scripting API: Vector2.MoveTowards“. https://docs.unity3d.com/ScriptReference/Vector2.MoveTowards.html (pristupljeno 24. lipanj 2022.).

[70] „Euclidean vector“, Wikipedia. 09. lipanj 2022. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Euclidean_vector&oldid=1092257966 (pristupljeno 24. lipanj 2022.).

[71] E. Monica, „Answer to ‚Extending vector and getting its x and y end coordinates‘“, Mathematics Stack Exchange, 25. travanj 2019. https://math.stackexchange.com/a/3206062 (pristupljeno 25. lipanj 2022.).

[72] U. Technologies, „Unity - Scripting API: Quaternion.Slerp“. https://docs.unity3d.com/ScriptReference/Quaternion.Slerp.html (pristupljeno 25. lipanj 2022.).

[73] „Add Firebase to your Unity project | Firebase Documentation“, *Firebase*. https://firebase.google.com/docs/unity/setup (pristupljeno 25. lipanj 2022.).

[74] „rest-apis“. https://www.ibm.com/cloud/learn/rest-apis (pristupljeno 25. lipanj 2022.).

[75] dotnet-bot, „Dictionary<TKey,TValue> Class (System.Collections.Generic)“. https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2 (pristupljeno 25. lipanj 2022.).

[76] BillWagner, „ref keyword - C# Reference“. https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/ref (pristupljeno 26. lipanj 2022.).

[77] „Language Integrated Query“, *Wikipedia*. 17. prosinac 2021. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Language_Integrated_Query&oldid=1060776115 (pristupljeno 26. lipanj 2022.).

[78] U. Technologies, „Unity - Manual: Tile Palette“. https://docs.unity3d.com/2018.3/Documentation/Manual/Tilemap-Palette.html (pristupljeno 26. lipanj 2022.).

[79] D. Kermek, „Uzorci dizajna za kreiranje“, materijali s predavanja, FOI, https://elf.foi.hr/pluginfile.php/13875/mod_resource/content/16/predavanja/Kermek_UzDiz_05.pdf (pristupljeno 27. lipanj 2022.).

[80] U. Technologies, „Unity - Scripting API: Resources.Load“. https://docs.unity3d.com/ScriptReference/Resources.Load.html (pristupljeno 27. lipanj 2022.).

[81] U. Technologies, „Unity - Scripting API: AudioSource.PlayOneShot“. https://docs.unity3d.com/ScriptReference/AudioSource.PlayOneShot.html (pristupljeno 27. lipanj 2022.).

[82] U. Technologies, „Unity - Scripting API: PlayerPrefs“. https://docs.unity3d.com/ScriptReference/PlayerPrefs.html (pristupljeno 28. lipanj 2022.).

[83] U. Technologies, „Unity - Scripting API: SceneManagement.SceneManager.LoadScene“.

https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html (pristupljeno 29. lipanj 2022.).

[84] U. Technologies, „Unity - Scripting API: MonoBehaviour.OnDisable()“. https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnDisable.html (pristupljeno 29. lipanj 2022.).