Razvoj igre za više igrača u Unityu

Grbac, Bruno

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:368274

Sveučilišni diplomski studij Informatika

Bruno Grbac

Razvoj igre za više igrača u Unityu

Diplomski rad

Mentor: Doc. Dr. sc. Miran Pobar

Rijeka, lipanj 2024.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Rijeka, 2.5.2024.

Zadatak za diplomski rad

Pristupnik/ica: Bruno Grbac

Naziv diplomskog rada: Razvoj igre za više igrača u Unityju

Naziv diplomskog rada na eng. jeziku: Development of a multiplayer game in Unity

Sadržaj zadatka: Proučiti i opisati probleme i rješenja za implementaciju mrežnog igranja za više igrača. Osmisliti i opisati dizajn vlastite igre za više igrača. Opisati implementaciju igre s posebnim naglaskom na aspekte umrežavanja i mrežne igre.

Mentor:
Izv. prof. dr. sc. Miran Pobar

Voditeljica za diplomske radove:
Doc. dr. sc. Lucia Načinović Prskalo

Komentor/ica:

Zadatak preuzet: 2.5.2024.

(potpis pristupnika/ce)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sažetak

Cilj ovog rada je kreirati igru za više igrača unutar platforme Unity. U radu će biti objašnjen proces razvoja računalnih igara, s posebnim naglaskom na multiplayer igre. Istražit će se koncepti umrežavanja u igrama, te dostupna rješenja za umrežavanje u Unityu, s posebnim fokusom na Mirror Networking kao alatu za mrežnu komunikaciju. Rad je strukturiran kroz nekoliko faza. Prvo je objašnjen dizajn izrađene igre uključujući koncept igre i gameplay te su nakon toga opisani osnovni pojmovi vezani uz game engine i razvoj multiplayer igara. Zatim se razmatraju rješenja za umrežavanje u Unityu, s posebnim osvrtom na Mirror Networking. Kroz implementaciju tih procesa pokriva se postavljanje mrežne infrastrukture, izrada gameplay komponenti poput player controllera, neprijatelja i AI, te sustava bodovanja i napredovanja. Također se razmatra sinkronizacija podataka, dizajn korisničkog sučelja i 3D modela, te optimizacija i performanse igre.

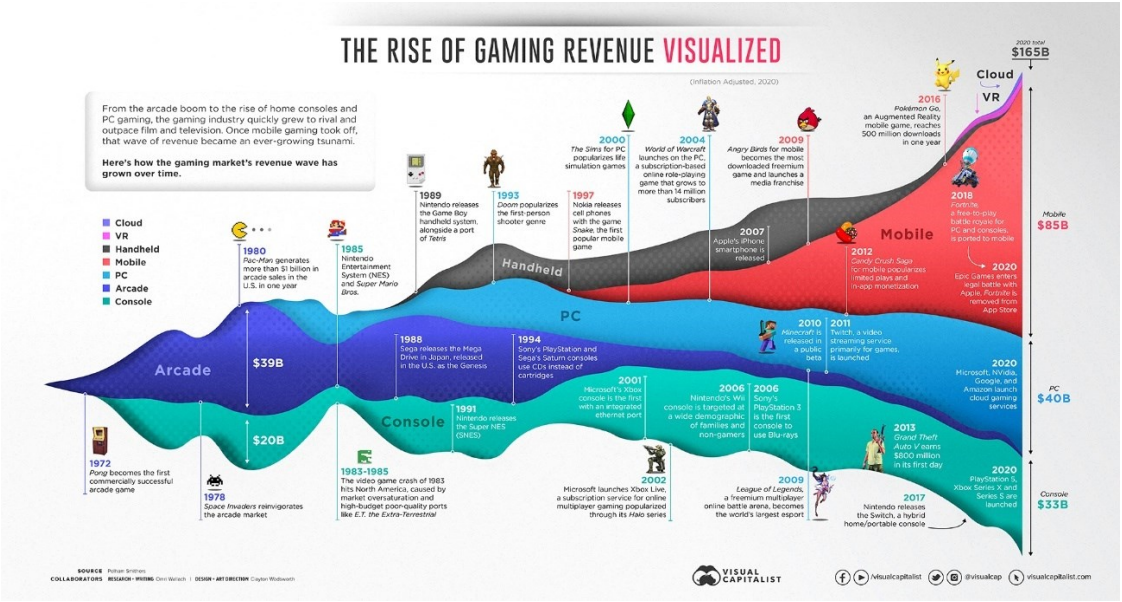
Ključne riječi:

Unity, Multiplayer igra, Game engine, Mirror Networking, Umrežavanje u igrama, Dizajn igre, Gameplay, Mrežna infrastruktura

Sadržaj

1. Uvod

Videoigre su postale jedan od najvažnijih oblika zabave u modernom društvu, privlačeći milijune igrača diljem svijeta. Ovaj medij ne samo da pruža zabavu, već i omogućava igračima da istražuju nove svjetove, razvijaju strategije i povezuju se s drugim igračima na globalnoj razini. Razvoj videoigara kombinira umjetnost, znanost i tehnologiju, stvarajući kompleksne virtualne svjetove. Povijest videoigara seže u pedesete i šezdesete dogine 20. stoljeća, kada su znanstvenici i inženjeri stvorili prve računalne igre kao eksperimentalne projekte [1]. Tijekom desetljeća, industrija videoigara rapidno je rasla, uvodeći nove tehnologije i konceptualne pristupe (Slika 1). Ključni trenutci u razvoju multiplayer igara uključuju lansiranje Microsoftove usluge Xbox Live 2001. godine, koja je popularizirala online igranje kroz Halo seriju. Godine 2004. izlazi World of Warcraft na PC-u, pretplatnička online role-playing igra koja je dosegla više od 14 milijuna pretplatnika, postavljajući nove standarde za MMO (Massively Multiplayer Online) igre. Današnje igre koriste napredne grafičke tehnologije, realističnu fiziku i složenu umjetnu inteligenciju kako bi pružile što uvjerljivije iskustvo.



Slika 1 Povijest igara prema tijeku prihoda. Izvor: [2]

Igre za više igrača (engl. multiplayer), koje omogućuju istovremeno igranje više igrača, postale su posebno popularne zahvaljujući razvoju internetske infrastrukture. Takve igre omogućuju igračima da se natječu ili surađuju u realnom vremenu, stvarajući dinamična i često nepredvidiva iskustva. Igranje s drugim ljudima dodaje novu dimenziju igračkom iskustvu, čineći ga socijalnim, kompetitivnim i kooperativnim. Razvoj multiplayer igara započeo je

krajem 1970-ih i početkom 1980-ih, kada su se prvi računalni sustavi počeli umrežavati. Jedan od prvih primjera multiplayer igara je "Spasim" (slika 2), igra iz 1974. godine koja je omogućavala do 32 igrača da istovremeno sudjeluju u svemirskoj borbi koristeći ARPAnet, preteču današnjeg interneta. S vremenom, kako su mrežne tehnologije napredovale, tako su i multiplayer igre postajale sve složenije i popularnije. U 1990-ima, dolaskom LAN (Local Area Network) zabava i internetskih kafića, igre poput Quakea i Counter-Strikea postale su pioniri u žanru kompetitivnih multiplayer pucačina. Danas su multiplayer igre jedan od najvažnijih segmenata industrije video igara, privlačeći milijune igrača diljem svijeta. Prema podacima s „SteamCharts“, Counter-Strike 2 redovito ima preko 1 milijun istovremenih igrača, dok Dota 2 bilježi prosječno preko 400 tisuća igrača dnevno [22]. Najpopularnije igre uključuju naslove kao što su Fortnite, League of Legends, Dota 2, Counter-Strike 2 i Apex Legends.Ove igre često nude razne modove igranja, uključujući timske borbe, battle royale (žanr igre koji kao cilj ima preživjeti i ostati zadnji na mapi koja se konstantno smanjuje)i suradničke misije, omogućujući igračima da se natječu ili surađuju s drugim igračima online. Popularnost ovih igara dodatno je povećana zahvaljujući razvoju e-sportova, gdje se profesionalni igrači natječu za značajne novčane nagrade i priznanja.

Slika 2 Igra Spasim. Izvor : [22]

Razvoj multiplayer igara predstavlja poseban izazov, jer zahtijeva učinkovitu mrežnu komunikaciju, sinkronizaciju podataka i optimizaciju performansi. Neki od najvećih problema s kojima se programeri suočavaju uključuju upravljanje latencijom, rješavanje problema sa skalabilnošću i osiguravanje sigurnosti podataka. Latencija, ili kašnjenje u komunikaciji između igrača i servera, može značajno utjecati na iskustvo igranja, stvarajući frustraciju kod igrača ako nije dobro riješeno. Skalabilnost je također kritična, jer multiplayer igre moraju biti sposobne podržati velik broj igrača bez pada performansi.

Cilj ovog rada je istražiti proces razvoja multiplayer igre koristeći Unity, jedan od najpopularnijih game engine-a danas. Poseban fokus ovog rada bit će na korištenju Mirror Networking-a, mrežnog rješenja koje pojednostavljuje razvoj multiplayer komponenti unutar Unitya.

U nastavku rada, prvo će biti objašnjen proces dizajna igre, uključujući faze raazvoja gri, koncepta igre i gameplaya. Potom će se razmotriti Unity i koncepti umrežavanja u igrama, uključujući njihove osnovne funkcionalnosti i način korištenja. Implementacijski dio rada pokrit će konkretne korake u razvoju igre Survivor A.R.C. izrađene u sklopu ovog projekta, od

postavljanja mrežne infrastrukture, preko izrade gameplay komponenti do sinkronizacije podataka i dizajna korisničkog sučelja.

2. Dizajn igre

Dizajn igre (engl. Game design) je proces stvaranja pravila i sadržaja igre koji će igračima omogućiti interaktivno i zabavno iskustvo, te kao takav on je ključan aspekt svakog uspješnog projekta, pružajući temelj na kojem se grade svi elementi igre, od mehanika i vizualnog stila do tehničkih rješenja i korisničkog iskustva [9].

U industriji videoigara, dizajn se dijeli na nekoliko ključnih elemenata: grafički dizajn, zvučni dizajn, dizajn nivoa, narativni dizajn i mehanički dizajn. Grafički dizajn određuje vizualni stil igre i može varirati od realističnih prikaza do stiliziranih i apstraktnih formi. Popularni stilovi uključuju high poly, low poly, pixel art... Zvučni dizajn uključuje stvaranje glazbe, zvučnih efekata i glasovnih zapisa koji poboljšavaju imerziju igrača. Dizajn nivoa obuhvaća stvaranje strukture i izgleda svake razine igre, osiguravajući da su izazovne, zabavne i intuitivne za navigaciju. Narativni dizajn odnosi se na razvoj priče i likova u igri, dok mehanički dizajn definira pravila i sustave igre, kao što su borbene mehanike, sustavi nagrađivanja i interakcija s okolinom.

2.1. Faze razvoja igri

Razvoj videoigara je složen i višefazni proces koji zahtijeva pažljivo planiranje, koordinaciju i izvršenje. Cjelokupni proces u većim kompanijama može se podijeliti u sedam ključnih faza (slika 3): planiranje, pred produkcija, produkcija, testiranje, pred-lansiranje, lansiranje i post-lansiranje [27]. Međutim, proces se često pojednostavljuje u tri glavne faze: pred produkciju, produkciju i post produkciju, što je često slučaj kod manjih timova i nezavisnih developera.

Slika 3 Faze razvoja igri. Izvor : [27]

Pred produkcija uključuje sve aktivnosti prije nego što započne stvarna izrada igre. Ova faza obuhvaća generiranje ideja, definiranje koncepta igre, odabir tehnologija i alata, te stvaranje prototipova. U ovoj fazi također se izrađuje dokument dizajna igre (GDD), koji detaljno opisuje sve aspekte igre i služi kao vodič za cijeli razvojni tim. Tijekom pred produkcije, tim postavlja temelje za razvoj igre, odlučuje o ključnim mehanikama igranja, umjetničkom stilu, priči i tehničkim aspektima. Ova faza je kritična jer promjene osnovnih postavki u kasnijim fazama razvoja mogu biti skupe i dugotrajne.

Produkcija je najduža i najintenzivnija faza razvoja igre. U ovoj fazi, stvaraju se svi elementi igre, uključujući grafiku, animacije, zvukove, likove, nivoe i kodiranje mehanika igranja. Timovi za razvoj, dizajn i umjetnost surađuju kako bi oživjeli svijet igre i osigurali da svi elementi budu međusobno usklađeni. Produkcija uključuje iterativni proces gdje se različiti dijelovi igre testiraju, poboljšavaju i usklađuju kako bi se postigla što bolja kvaliteta.

Post produkcija obuhvaća aktivnosti nakon što je igra tehnički dovršena. Ova faza uključuje testiranje igre kako bi se pronašle i ispravile greške, balansiranje igranja, poliranje svih aspekata igre i priprema za lansiranje. Nakon lansiranja, timovi nastavljaju raditi na održavanju igre, ispravljanju greški, dodavanju novog sadržaja i poboljšavanju korisničkog iskustva kroz ažuriranja i dodatke. Pojednostavljeni pristup s tri faze - pred produkcija, produkcija i post produkcija, pruža osnovni okvir za razumijevanje razvoja videoigara, no korištenje detaljnijih sedam faza omogućuje većim timovima bolju organizaciju i praćenje napretka, te osigurava da nijedan aspekt razvoja ne bude zapostavljen.

2.2. Dizajn igre Survivor A.R.C.

Pri izradi ove igre kreiran je GDD (engl. Game Design Document) kako bi se detaljno opisali svi aspekti igre, uključujući koncept, mehanike, likove, priču i vizualni stil. Ovaj dokument služi kao vodič tijekom cijelog procesa razvoja igre, omogućujući nam da se u bilo kojem trenutku ponovno osvrnemo na definirane ciljeve i smjernice projekta.

2.2.1. Game Design Document

Koncept igre

Survivor A.R.C. je co-op wave shooter koji omogućuje da do četiri igrača surađuju u preživljavanju valova neprijatelja. Cilj je stvoriti dinamično i uzbudljivo multiplayer iskustvo koje ističe timsku igru i strateško razmišljanje. Kako bi se to postiglo biti će potrebno sinkronizirati podatke poput dodjele bodova, pozicije igrača i neprijatelja i mnoge druge. Igrači će imati priliku udružiti se u predvorju (engl. lobby), birati mape i konfigurirati svoje likove kroz sustav talenata. Svaka mapa donosi različite izazove s raznim razinama težine, a igrači će trebati surađivati kako bi preživjeli napade neprijatelja. Tijekom igre, igrači će dobivati bodove za ubijanje neprijatelja, koje će moći koristiti za otključavanje novih dijelova mape ili kupovinu boljih oružja. Talenti koje igrači odabiru unaprijed utječu na njihove sposobnosti i način igre, a napredak se sprema u Unity Cloud, omogućujući kontinuirani razvoj i prilagodbu likova.

Dizajn igre

Gameplay:

Igrači se stvaraju na različitim mapama koje predstavljaju različite post apokaliptične lokacije, poput napuštenih gradova, laboratorija, vojnih baza ili šumskih područja.

Cilj igre je preživjeti valove mutiranih čudovišta i održati svoj tim na okupu što je duže moguće. Igra je usmjerena na 4-člani kooperativni gameplay, gdje igrači moraju taktički surađivati, pružati potporu jedni drugima i koordinirati napade kako bi se suprotstavili sve zahtjevnijim valovima neprijatelja. Igrači će se suočiti s različitim vrstama čudovišta, svako s jedinstvenim sposobnostima i slabostima. Svaka vrsta čudovišta zahtijevat će drugačiju strategiju i pristup borbi. Uspješno ubijanje čudovišta donosi bodove koji se mogu iskoristiti za nadogradnju oružja, otključavanje novih sposobnosti i dodataka te kupovinu opreme i resursa.

Okruženje:

Igra pruža raznolika okruženja, od urbanih područja do divljine, gdje igrači moraju snalažljivo koristiti svoje okruženje kako bi preživjeli. Svako okruženje ima svoje specifičnosti i taktičke mogućnosti. Na primjer, urbano područje može pružiti zaklon i više opcija za strategijsko pozicioniranje, dok divljina može pružiti prirodne zamke i resurse koje igrači mogu iskoristiti.

Okruženja su detaljno dizajnirana s naglaskom na atmosferu post apokaliptičnog svijeta. Ruševine, napuštena vozila, razbijeni objekti i opći osjećaj propadanja dodatno pridonose atmosferi igre.

Igrači

Opis igrača

- Hodanje, trčanje – daju igraču mogućnost kretanja
- Pucanje – igrač drži oružje, služi za ubijanje neprijatelja
- Početak igre – igrač se stvara na odabranoj mapi
- Informacije – zdravlje, broj metaka, team score te broj vala
- Gubitak – igrač gubi kada ostane bez zdravlja, igrač se ruši

Svojstva igrača

- Health – ako se isprazni u potpunosti igrač umire te pada na pod
- Talent System – sustava talenata koji omogućuje igračima unaprjeđivanje svojih likova

Nagrade i kazne igrača (*Power-ups & Pick-ups*)

Igrači dijele zajednički score koji se može koristiti za unaprjeđivanje oružja i dodavanje zdravlja/metaka

User Interface (UI)

Igrač se kreće sa tipkama W, A, S i D te brzo trči tipkom Shift. Tipkom R može nadopuniti spremnik s metcima te tipkom TAB se otvara izbornik za upravljanje oružjem i kupovinu nadogradnji. Korisnik vidi health bar, oružja koje nosi, broj metaka u spremniku i izvan spremnika, broj vala te team score. U otvorenom izborniku za oružje, igrač može vidjeti specifikacije trenutnog oružja gdje također može isto i nadograditi te će mu se specifikacije povećati.

Protivnički elementi

Igra trenutno sadrži veliki broj istog neprijatelja s kojim se stvaraju valovi napada.

Težina/složenost

Težina igre se mijenja u odnosu na broj vala na kojem se igrač nalazi, svaki val napada je sve teži i kompleksniji za preživjeti.

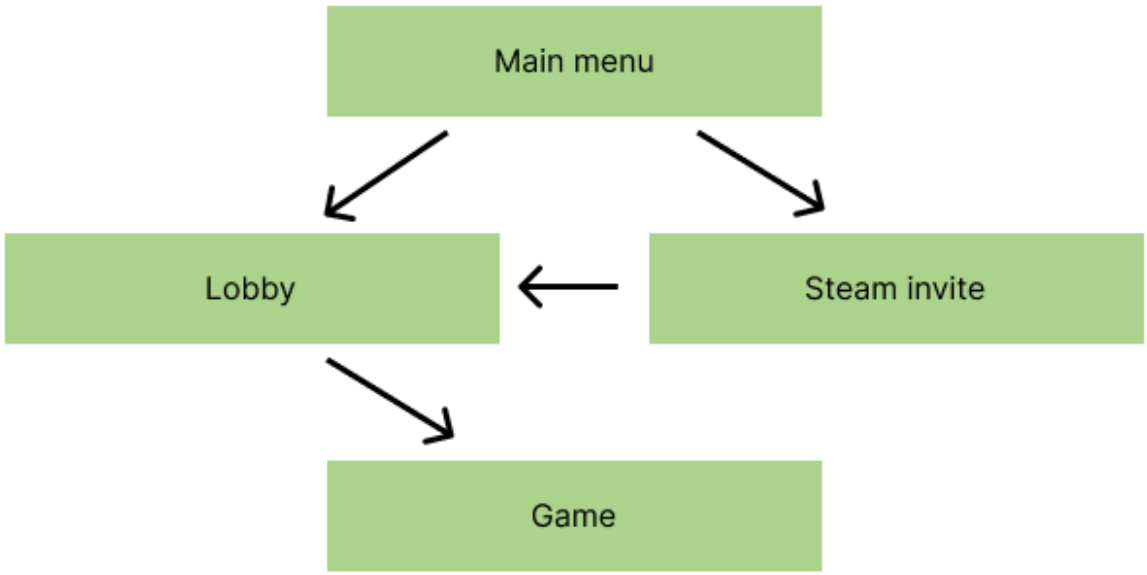
Zvuk

Igrač čuje pucanje iz oružja te izmjenu spremnika.

Grafika

Modeli u igri su low poly stila koji je poznat po svojoj jednostavnosti i učinkovitoj upotrebi poligona.

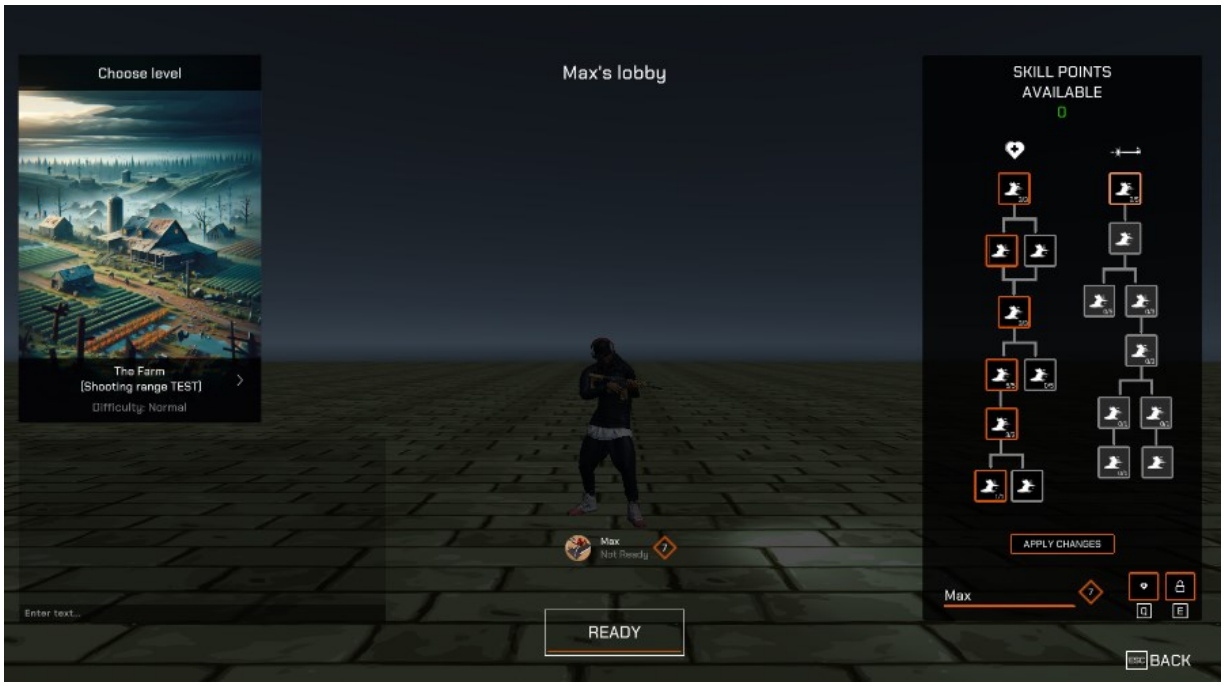
Dijagram arhitekture igre (slika 4)



Slika 4 Dijagram arhitekture igre.

2.2.2. Implementirani elementi igre

Igra započinje kada jedan igrač „domaćin“ otvori predvorje (engl. lobby) u koje se ostali igrači mogu pridružiti. U predvorju igrači mogu komunicirati, odabrati mapu i konfigurirati svoje likove. Svaka mapa dolazi s različitim razinama težine, što utječe na broj i snagu neprijatelja koje će igrači susresti. Na slici 5. može se vidjeti raspored svih funkcionalnosti koji lobby pruža od odabira talenta do levela (razine) igrača, te odabira mape.



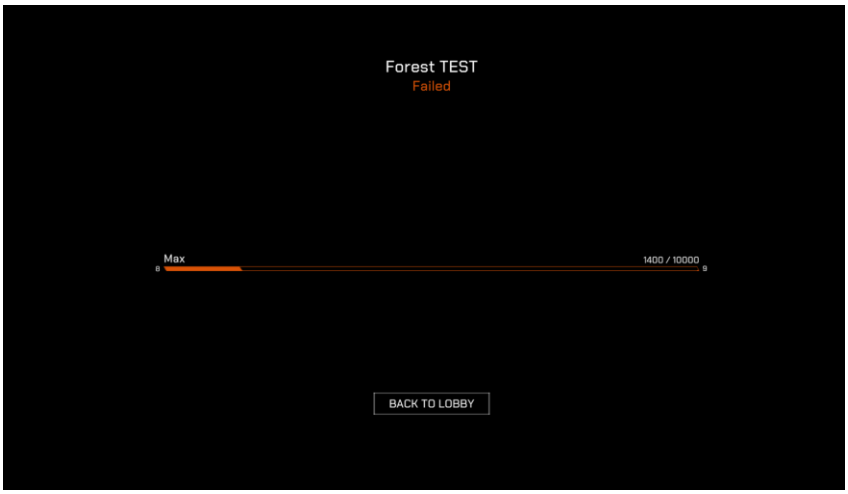
Slika 5 Prikaz lobbya

Prije ulaska na mapu, igrači imaju mogućnost odabrati svoje talente iz stabla talenata (engl. talent tree). Talenti se otključavaju napredovanjem na više razine (engl. level up), što se događa svaki put kad igrač završi mapu i dobije iskustvene bodove (engl. experience points), vidljivo na slici 8. Ovi talenti pružaju različite sposobnosti i pojačanja koja mogu značajno utjecati na način igre i strategije koje igrači koriste. Talenti se spremaju u Unity Cloud, omogućujući igračima da nastave svoj napredak neovisno o uređaju na kojem igraju. Kada igrači započnu mapu, neprijatelji počinju dolaziti u valovima (Slika 6).

Slika 6 Prikaz igre

Cilj igre je preživjeti te valove što je duže moguće. Igrači dobivaju bodove za svakog poraženog neprijatelja, a ukupni rezultat tima može se koristiti za otključavanje novih dijelova mape prikazano na slici 7 ili kupovinu boljih oružja kod automata za prodaju (engl. vending machine). Ova mehanika potiče timsku suradnju i strateško planiranje, jer igrači moraju odlučiti kako najbolje iskoristiti svoje resurse kako bi preživjeli i napredovali u igri. Igra također uključuje sustav napredovanja u kojem igrači mogu napredovati na više razine i otključavati nove talente, pružajući kontinuirani osjećaj napretka i postignuća (slika 8). Sve ove mehanike zajedno stvaraju bogato i dinamično iskustvo igranja koje nagrađuje suradnju i strateško razmišljanje.

Slika 7 Mogućnost otključivanja novih dijelova mape.



Slika 8 Prikaz završetka levela (razine).

Survivor A.R.C., koristi low poly stil koji je poznat po svojoj jednostavnosti i učinkovitoj upotrebi poligona za stvaranje vizualno privlačnog, ali tehnički laganog grafičkog prikaza (slika 7). Low poly stil omogućuje glatko izvođenje igre i na slabijim hardverskim konfiguracijama, što povećava dostupnost igre široj publici. Tematski, "Survivor A.R.C." je apokaliptična igra koja nastoji prenijeti osjećaj straha i napuštenosti. Dizajn okruženja i likova osmišljen je kako bi reflektirao tmurnu i opustošenu atmosferu svijeta pogođenog katastrofom. Korištenjem tamnih tonova, oštećenih struktura i detalja koji sugeriraju propadanje i napuštenost, igra uspijeva stvoriti snažan osjećaj nesigurnosti i napetosti. Ovaj pristup dizajnu doprinosi narativnoj dubini igre, omogućujući igračima da se emocionalno povežu s okruženjem i likovima te dožive iskustvo preživljavanja u post-apokaliptičnom svijetu.

Slika 9 Prikaz low poly stila unutar Survivor A.R.C.

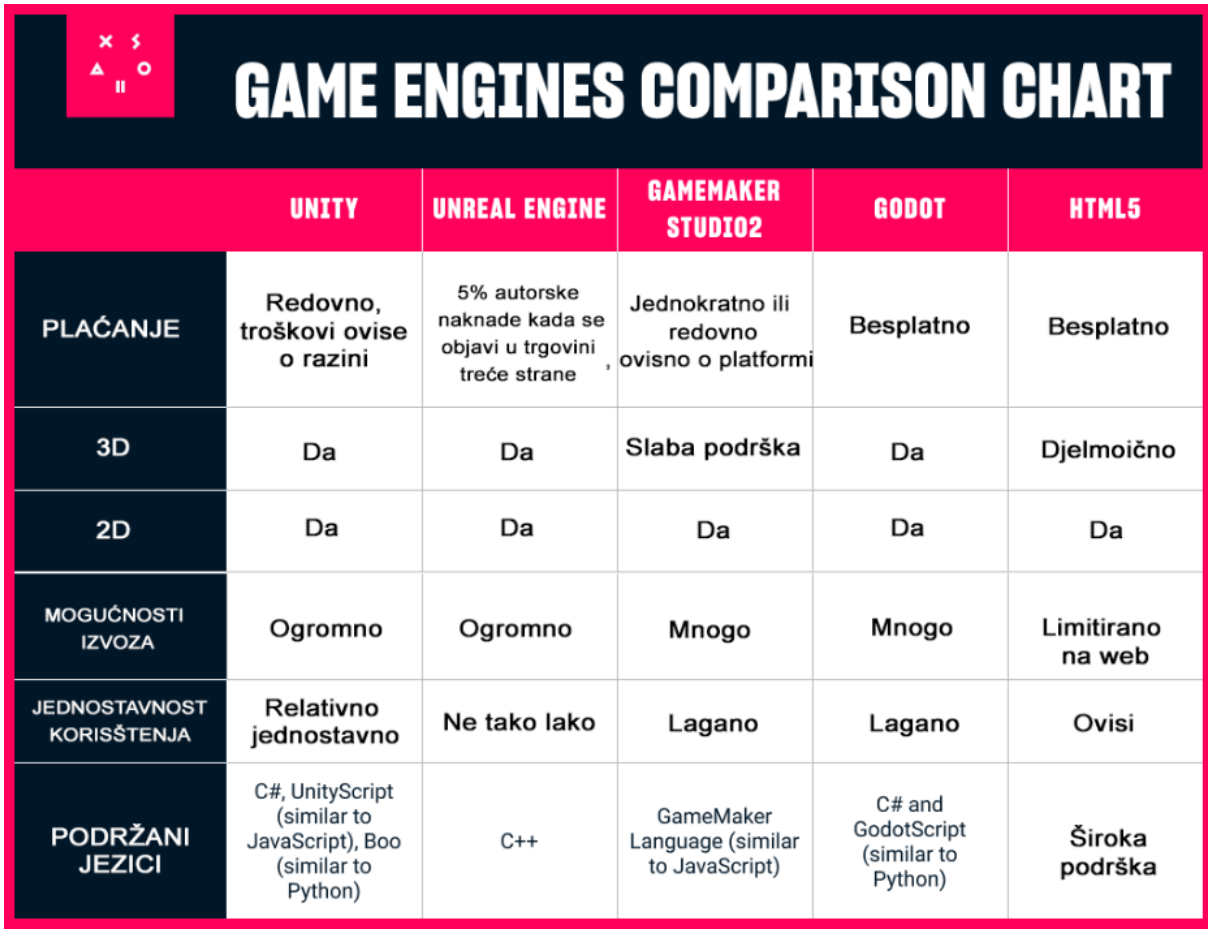
3. Izrada računalnih igara

Izrada računalnih igara kompleksan je proces koji uključuje brojne faze, od koncepta, razrade dizajna do konačne implementacije. Igre se danas izrađuju za različite platforme, uključujući računala (PC), konzole (kao što su PlayStation, Xbox i Nintendo Switch), mobilne uređaje (pametni telefoni i tableti) te web preglednike. Svaka od ovih platformi donosi svoje specifične izazove i zahtjeve, što uključuje prilagodbu različitim hardverskim specifikacijama, optimizaciju performansi i prilagođavanje korisničkog sučelja. Na kompleksnost razvoja igara također utječu mnoge dizajnerske odluke, kao što su izbor žanra i odluke o mehanikama igre, vizualnom stilu i zvučnom dizajnu. Akcijske igre zahtijevaju preciznu kontrolu i responzivno korisničko sučelje, dok avanturističke igre naglašavaju priču i istraživanje, što zahtijeva pažljivo dizajnirane svjetove i složene narativne strukture. Strategije se oslanjaju na taktičko planiranje i umjetnu inteligenciju, dok role-playing igre (RPG) uključuju razvoj likova i duboke priče, često s velikom količinom sadržaja i kompleksnim mehanikama napredovanja. U modernom razvoju igara, korištenje game engine-a postalo je standard, omogućujući programerima da uštede vrijeme i resurse te se fokusiraju na kreativne aspekte igre. Jedna od ključnih prednosti game engine-a je upravo mogućnost razvoja za više platformi uz minimalne prilagodbe. Bez obzira na žanr, sve igre dijele određene zajedničke elemente koje game engine-i efikasno rješavaju poput grafičkog prikaza, fizike, zvuka, animacija i umrežavanja, čineći razvoj igara pristupačnijim i učinkovitijim.

3.1. Game engine

Game engine, ili pogonski motor za igre, softverski je okvir dizajniran za razvoj i kreiranje video igara. Osnovna funkcija game engine-a je pružanje temeljnih komponenti i alata koji omogućuju programerima da se fokusiraju na kreativne aspekte razvoja igre, bez potrebe da iz temelja pišu kod za osnovne funkcionalnosti kao što su grafički prikaz, fizika, zvuk i animacija. Korištenje game engine-a započelo je krajem 1980-tih i početkom 1990-tih godina, kada su se programeri suočili s rastućom složenošću razvoja igara [26]. Prvi game engine-i pojavili su se kako bi olakšali rad na različitim aspektima razvoja, omogućujući programerima igara da ponovno koriste kod i resurse u različitim projektima. Jedan od prvih značajnih game engine-a bio je id Tech 1 (poznat i kao Doom engine), razvijen od strane id Software-a za igru Doom 1993. godine. Ovaj engine omogućio je kreatorima igara da stvore složene trodimenzionalne svjetove i bio je pionir u mnogim tehnikama koje se koriste i danas. Ovaj engine nije samo omogućio id Software-u da brže razvija nastavke i nove igre, već je postao osnova za mnoge druge igre koje su kreirali različiti timovi diljem industrije. Programeri su mogli koristiti već postojeće alate i module, prilagoditi ih svojim potrebama i tako značajno ubrzati proces razvoja.

Game engine-i obično sadrže nekoliko osnovnih komponenti koje omogućuju razvoj igara. Ove komponente uključuju grafički sustav, fizikalni sustav, zvučni sustav, komponentu za animaciju, sustav za umrežavanje, skriptne sustave i alate za korisničko sučelje. Grafički sustav je ključna komponenta game enginea koja je zadužena za prikazivanje vizualnih elemenata igre, uključujući 2D i 3D grafiku [18]. Ova komponenta koristi različite grafičke API-je, kao što su DirectX, OpenGL, Vulkan i Metal, za renderiranje grafike na ekranu [19]. Na primjer, DirectX nije sam po sebi game engine, već je skup API-ja koji omogućuju razvoj grafičkih aplikacija, uključujući igre. Game engine-i obično uključuju alate za vizualni dizajn scena, kao što je grafički editor scena, koji omogućuju programerima i dizajnerima da postavljaju objekte, svjetla, kamere i druge elemente unutar igre. Ovi alati često uključuju mogućnosti kao što su pregled u stvarnom vremenu (real-time preview), što omogućuje trenutan uvid u izgled i ponašanje scene bez potrebe za pokretanjem igre. Fizikalni sustav simulira fizikalne zakone unutar igre, kao što su gravitacija, sudari i kretanje objekata. Na primjer, kada objekt padne, fizikalni sustav će izračunati putanju pada uzimajući u obzir gravitaciju i eventualne sudare s drugim objektima. Zvučni sustav upravlja zvučnim efektima i glazbom unutar igre. Ova komponenta omogućuje reprodukciju zvučnih datoteka, manipulaciju zvuka u stvarnom vremenu (npr. promjena glasnoće, pitcha), te prostornu zvučnu obradu koja omogućuje igračima da čuju zvukove iz različitih smjerova i udaljenosti, čime se stvara realističan zvučni doživljaj.

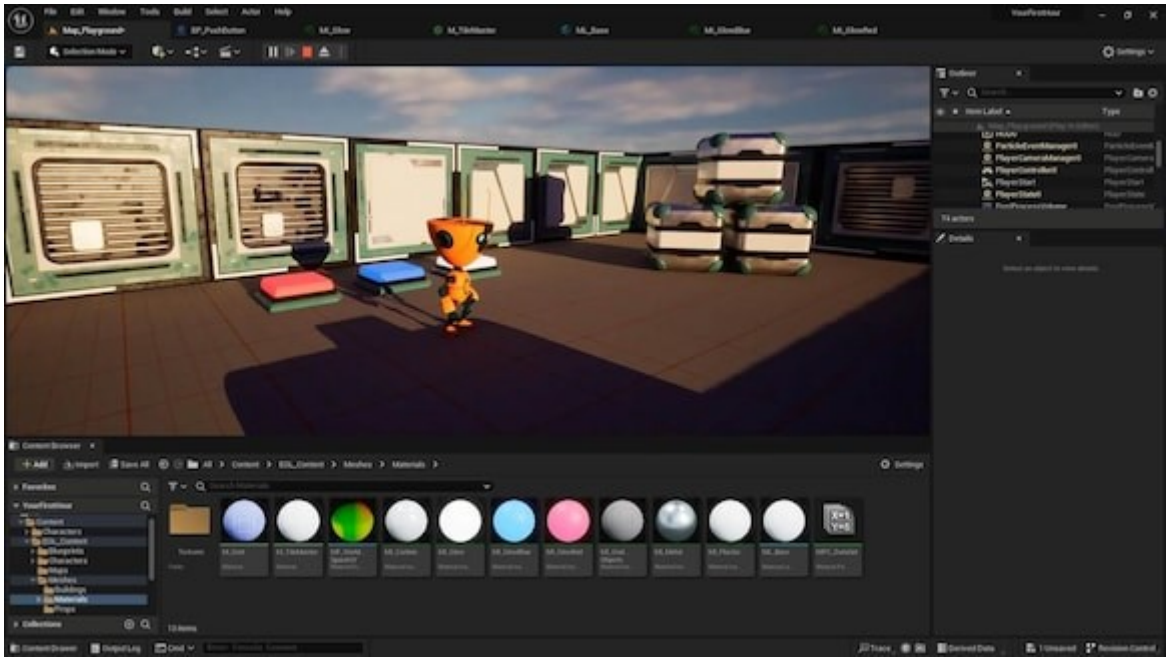


Slika 10 Usporedba game engine-a. Izvor : [3]

Tablica s slike 10 uspoređuje pet popularnih game engine-a: Unity, Unreal Engine, GameMaker Studio 2, Godot i HTML5, fokusirajući se na ključne aspekte važnosti za razvoj igara.

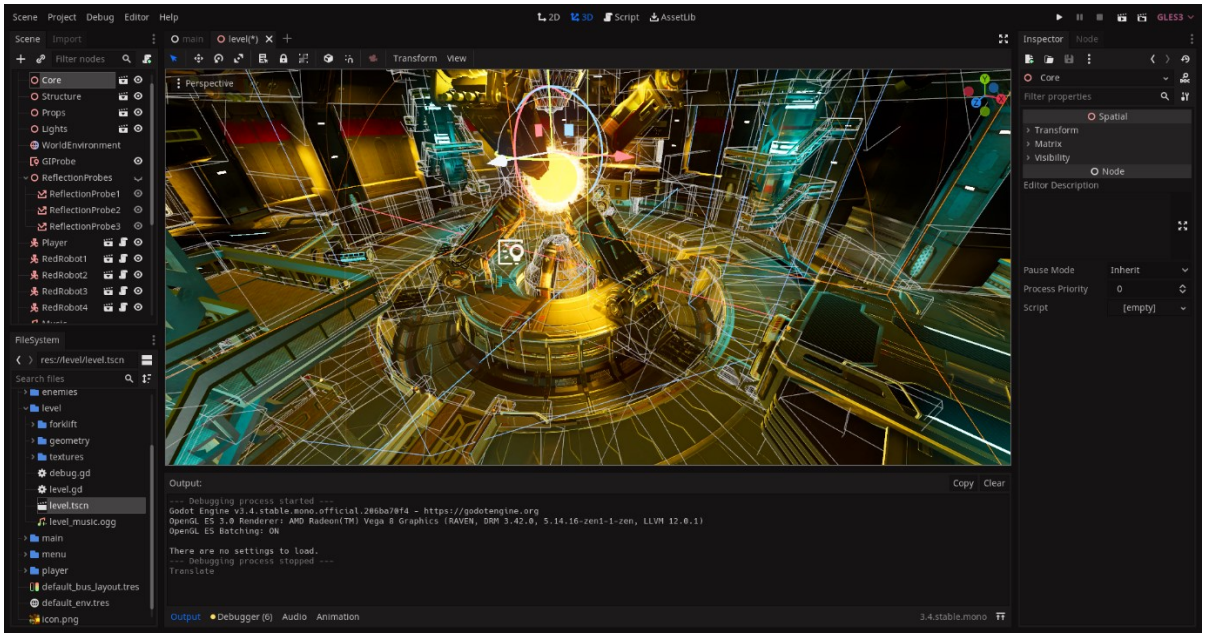
Iako HTML5 nije kompletan game engine, već više softverski sloj, vrijedan je spomena. Okviri (engl. Frameworks) poput Construct i Phaser koriste ovu tehnologiju i platformu za stvaranje visokokvalitetnih igara unutar preglednika, koje mogu biti online ili za jednog igrača. Informacija poput modela plaćanja značajna je jer direktno utječe na budžet projekta. Unity koristi redoviti model naplate, što podrazumijeva godišnje plaćanje ovisno o odabranom planu, dok Unreal Engine naplaćuje 5% tantijema, što može biti prednost za projekte s ograničenim početnim sredstvima. Godot i HTML5 su besplatni, što ih čini privlačnima za indie developere i projekte s niskim budžetom. Podrška za 3D i 2D grafiku je ključna za definiranje opsega i vizualnog stila igre. Unity i Unreal Engine nude sveobuhvatnu podršku za obje vrste grafike, omogućujući raznovrsne kreativne projekte. Nasuprot tome, HTML5 pruža samo ograničenu podršku za 3D grafiku, što ga čini pogodnim uglavnom za jednostavnije igre.

Unreal Engine, razvijen od strane Epic Games, prvi je put predstavljen 1998. godine. Poznat je po svojoj moćnoj grafici i širokoj upotrebi u visokobudžetnim igrama, a koristi se u igrama kao što su Fortnite, Gears of War i Unreal Tournament (slika 11).



Slika 11 Prikaz Unreal Enginea. Izvor [20]

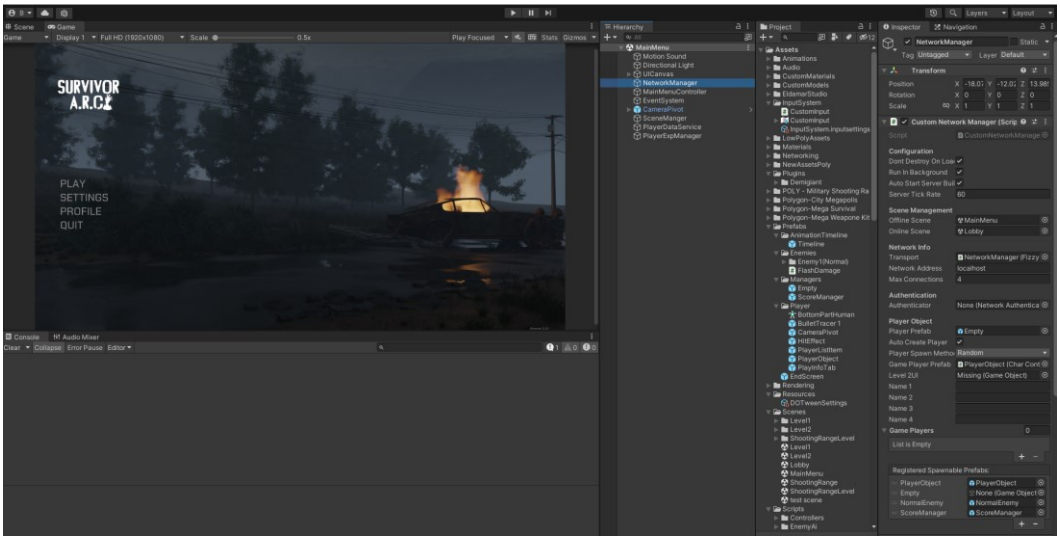
Unity, kojeg je razvila tvrtka Unity Technologies, prvi put je predstavljen 2005. godine. Ovaj engine postao je popularan zbog svoje prilagodljivosti i pristupačnosti, posebno među indie developerima. Indie developeri, skraćeno od (engl. independent developers) su pojedinci ili mali timovi koji rade na razvoju video igara bez podrške velikih izdavačkih kuća. Oni često imaju ograničene resurse i budžete, ali su poznati po svojoj kreativnosti i inovativnim pristupima u stvaranju igara. CryEngine, razvijen od strane Crytek-a i prvi put predstavljen 2002. godine, poznat je po svojim realističnim grafičkim mogućnostima. Koristi se u igrama kao što su Far Cry, Crysis i Warface. Godot Engine, koji je open-source i prvi put predstavljen 2014. godine, također je stekao popularnost zbog svoje fleksibilnosti i mogućnosti prilagodbe, a koristi se za razvoj mnogih indie i komercijalnih igara (slika 12). U ovom projektu odabran je Unity zbog njegove sveobuhvatne podrške za 2D i 3D grafiku, te relativno jednostavnog korisničkog sučelja koje omogućuje brzi razvoj i prototipiranje.



Slika 12 Prikaz Godot Engine. Izvor [21]

3.1.1. Osnove Unity engine-a

Jedna od ključnih prednosti Unityja je njegova pristupačnost i jednostavnost korištenja. Sa svojim intuitivnim korisničkim sučeljem vidljivim na slici 13, čak i početnici u svijetu razvoja igara mogu brzo početi stvarati svoje projekte. Uz to, Unity nudi obilan izbor resursa poput tutorijala, dokumentacije i zajednice korisnika, što olakšava učenje i rješavanje problema tokom razvoja igre. Unity podržava izradu skripti u C# programskom jeziku. C# je popularan, moderni programski jezik s jasnom sintaksom i bogatim ekosustavom alata, što ga čini idealnim izborom za razvoj igara. Iako je C# moćan jezik, neki programeri mogu smatrati da nije najbolji izbor zbog nekih ograničenja, kao što su performanse u usporedbi s jezicima poput C++ ili brzinom razvoja u odnosu na neke skriptne jezike. Kada se radi o korištenju Unityja, ključno je razumijevanje njegovih osnovnih komponenata i koncepta. Razvoj igara u Unityju sastoji se od stvaranja scene, dodavanja objekata, definiranja njihovih svojstava i pisanja skripti za upravljanje ponašanjem. Kroz interakciju s različitim komponentama, kao što su transformacije, kolizije, animacije i umrežavanje, developeri mogu stvoriti bogate i interaktivne svjetove.



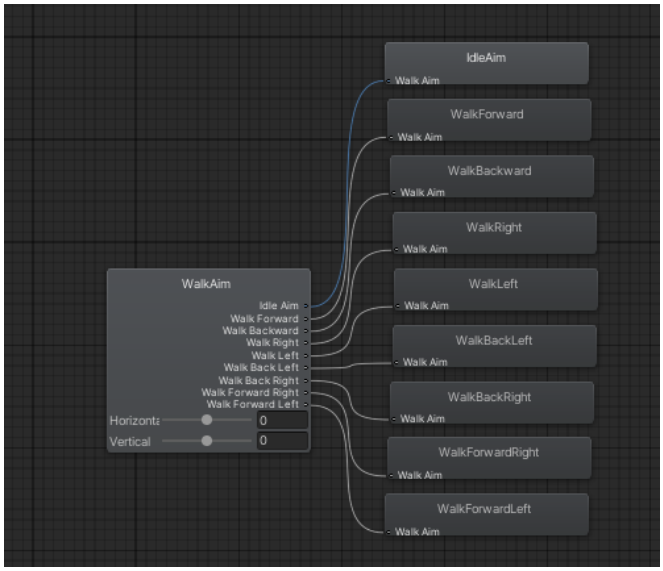
Slika 13 Unity project window.

Osnovne komponente u Unity platformi korištene za izradu igre Survivor A.R.C su sljedeće :

- **Fizika**: Unity koristi fiziku kako bi simulirao realistične pokrete i interakcije između objekata u igri. Ovo uključuje gravitaciju, sudare, trenje i druge fizičke fenomene koji pomažu u stvaranju uvjerljivih scenarija unutar igre. U „Survivor A.R.C“ fizika se koristila kroz komponente poput „Rigidbody“ i „Colliders“. Na primjer, „Rigidbody“ se koristi za dodavanje fizikalnih svojstava objektima poput gravitacije, omogućujući im da reagiraju na sile i sudare, dok se „Colliders“ koriste za detekciju kolizija, poput kada neprijatelj zamahne rukom i pogodi igrača.
- **Particle System**: Particle System je korišten za kreiranje vizualnih efekata kao što je vatra vidljiva na slici 14. Ova komponenta omogućava detaljnu kontrolu nad ponašanjem i izgledom čestica, što pridonosi vizualnoj atraktivnosti igre.
- **NavMeshAgent**: NavMeshAgent komponenta se koristi za navigaciju likova unutar igre. Ova komponenta omogućava likovima da se kreću po unaprijed definiranoj navigacijskoj mreži (engl. NavMesh) (Slika 15), izbjegavajući prepreke i optimalno pronalazeći put do cilja.
- **UI (User Interface):** Unityjev UI sustav omogućava kreiranje interaktivnih elemenata korisničkog sučelja kao što su izbornici, gumbi, tekstualni okviri i drugi elementi.
- **Animacija**: Animacijski sustav u Unityju omogućava kreiranje i kontrolu animacija likova i objekata. Animacijski sustav podrazumijeva „Animator“ sučelje koje dopušta developerima da stvaraju prijelaze između animacija, i mogućnost pozivanja animacija kroz kod napisan u personaliziranim skriptama (slika 16).
- **Zvuk**: Zvuk je ključan za stvaranje atmosfere i poboljšanje doživljaja igre. Unity omogućava integraciju zvučnih efekata, glazbe i drugih audio elemenata, te njihovu kontrolu kroz razne komponente i skripte.

Slika 14 Prikaz vatre uz pomoću Particle systema

Slika 15 Prikaz NavMesh mreže



Slika 16 Unity Animator

3.2. Razvoj multiplayer igara

Razvoj multiplayer igara predstavlja niz izazova koji ih čine složenijima za izradu u usporedbi s single-player igrama. Jedan od glavnih izazova je tehnička kompleksnost umrežavanja. Multiplayer igre moraju osigurati stabilnu i brzu komunikaciju između svih igrača kako bi iskustvo igranja bilo glatko i sinkronizirano. Glavni tehnički izazov u razvoju multiplayer igara je sinkronizacija podataka između svih igrača. Sinkronizacija podataka podrazumijeva da svaki igrač vidi iste događaje u igri u stvarnom vremenu. S tehničke strane, to podrazumijeva kontinuiranu razmjenu informacija između klijenata (igrača) i servera, koji centralizira sve podatke i distribuira ih natrag igračima. Ova razmjena informacija uključuje pozicije likova, stanja objekata, rezultate interakcija i druge važne događaje unutar igre.

Osim sustava koji podržava sinkronizaciju podataka unutar igre, potrebno je razmišljati i povezivanju igrač. Povezivanje igrača u multiplayer igrama ključno je za stvaranje kvalitetnog iskustva. Postoje različiti načini na koje igre omogućuju igračima da se spoje, a ti načini variraju ovisno o tipu i žanru igre.

Jedan od najčešćih pristupa je korištenje javnih servera, gdje igrači mogu izravno pristupiti dostupnim serverima i pridružiti se igri. Ovaj model često se koristi u sandbox ili MMO igrama, gdje igrači imaju mogućnost odabrati server prema svojim preferencijama ili lokaciji. Na taj način, igrači mogu birati između različitih zajednica i stilova igranja. Drugi pristup uključuje privatne servere, gdje igrači sami kreiraju servere i pozivaju druge da im se pridruže. Ovaj model omogućuje veću kontrolu nad igrom i često se koristi u strategijskim ili igrama preživljivanja.

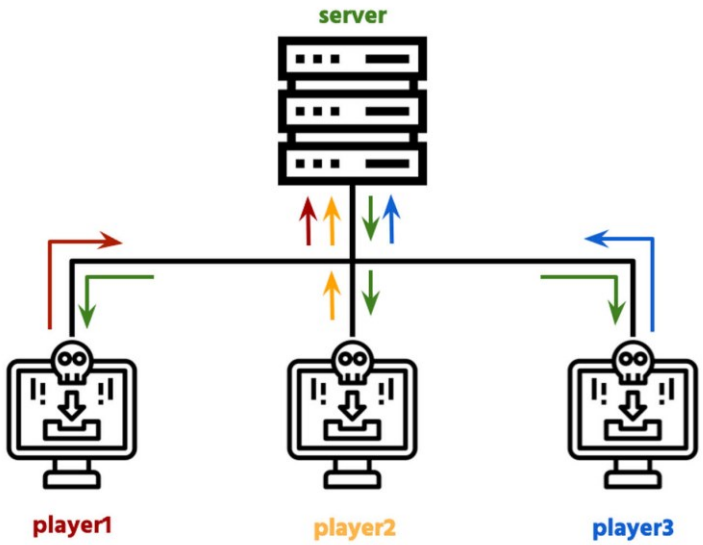
U kompetitivnim igrama, potrebno je koristi malo kompleksniji sustav povezivanja. Takav kompleksni sustav zove se Matchmaking, to je proces koji automatski spaja igrače u temelju različitih kriterija, poput vještina, ranga i preferencija. Cilj ovog sustava je osigurati

uravnotežene i poštene mečeve, čime se povećava zadovoljstvo igrača i smanjuje frustracija. Matchmaking sustavi često koriste algoritme koji analiziraju podatke o igračima, uključujući njihove performanse u prethodnim igrama. Na temelju tih informacija, sustav predlaže protivnike ili suigrače koji su na sličnom nivou vještina.

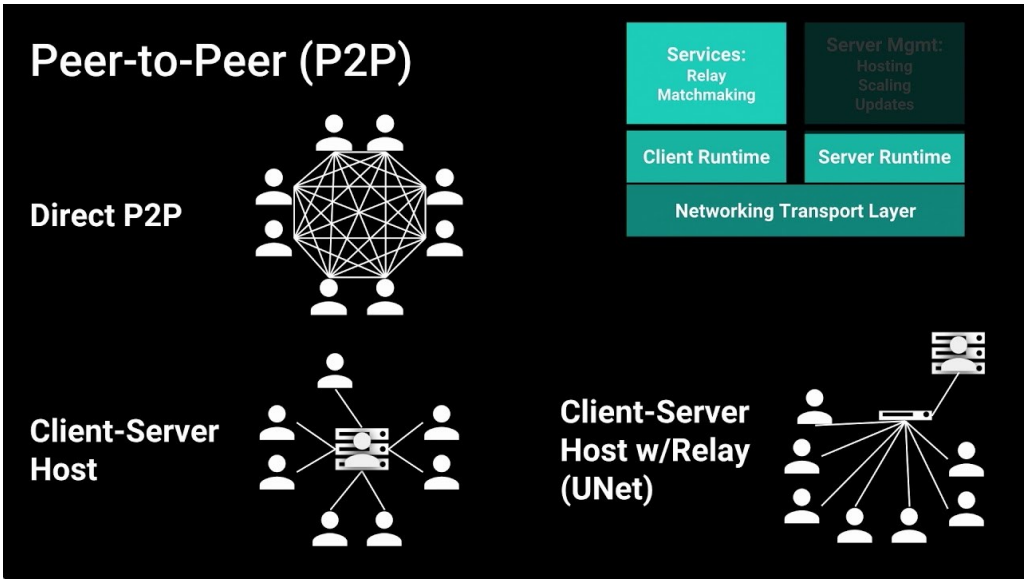
Uz to sigurnosni aspekti su kritični, jer multiplayer igre često postaju meta hakera i varalica [25]. Implementacija sigurnosnih mjera uključuje autentifikaciju igrača, šifriranje komunikacije i detekciju varanja (anti-cheat). Anti-cheat sustavi kao što su VAC (Valve Anti-Cheat) ili BattlEye detektiraju i sprječavaju varanje analizom ponašanja igrača i skeniranjem memorije igre. Na primjer, u kompetitivnim igrama kao što su "Fortnite" ili "Valorant", prisutnost varalica može ozbiljno narušiti iskustvo igranja i poštenje igre. Iz razloga što varanje u takvim igrama stvara velike probleme za igrače, veće kompanije pokušavaju to zaustaviti na različite načine. Jedan od glavnih načina u borbi protiv varalica u modernim igrama je implementacija vrlo nametljivih (engl. intrusive) anti-cheat sustava, poput Riot Gamesovog Vanguarda za igru "Valorant". Vanguard je postao sinonim za naprednu tehnologiju detekcije varanja koja se, iako kontroverzna zbog svoje invazivnosti, pokazala izuzetno efikasnom. Vanguard funkcionira na razini kernela, što znači da ima pristup većini sistemskih resursa i može pratiti ponašanje softvera na dubljoj razini nego većina drugih anti-cheat sustava.

3.3. Koncepti umrežavanja u igrama

Način rješavanja problema sinkronizacije ovisi dijelom i o odabranoj mrežnoj arhitekturi. Postoji nekoliko različitih mrežnih arhitektura koje se koriste u razvoju multiplayer igara, uključujući peer-to-peer (P2P), klijent-server (slika 17), hibridne modele i model klijent-domaćin (engl. client-host) (slika 18). Svaka od ovih arhitektura ima svoje prednosti i nedostatke, a izbor odgovarajuće arhitekture ovisi o specifičnim zahtjevima igre i ciljanom broju igrača.



Slika 17 Klijent-server arhitektura. Izvor : [4]



Slika 18 Koncepti umrežavanja. Izvor: [5]

3.3.1. Peer-to-peer arhitektura

Peer-to-peer (P2P) arhitektura omogućava svakom računalu (peer) da direktno komunicira s drugim računalima u mreži. Na početku, jedan sudionik identificira drugog koristeći tehnike kao što su posrednički serveri za inicijalno uparivanje ili prijenos IP adresa putem posrednika. U ovoj konfiguraciji, nema centralnog servera, već svaki igrač šalje i prima podatke od drugih igrača. P2P umrežavanje najčešće se koristi u igrama koje zahtijevaju brzu i direktnu komunikaciju između malog broja igrača, kao što su igre s malim brojem sudionika ili co-op igre. Prednosti P2P arhitekture uključuju niže troškove, budući da nema potrebe za centralnim serverom, čime se smanjuju troškovi održavanja mrežne infrastrukture. Također, direktna komunikacija između igrača može smanjiti latenciju, što rezultira bržim odzivom u igri. Međutim, nedostaci P2P arhitekture uključuju sigurnosne probleme, jer su P2P mreže podložnije varanju i hakiranju budući da svaki peer ima pristup svim podacima igre. Primjer igre koja koristi P2P arhitekturu je Age of Empires II, gdje se često javljaju problemi s varanjem jer svaki igrač ima pristup svim podacima igre, omogućavajući manipulaciju igrom. Također, kako broj igrača raste, održavanje sinkronizacije među svim sudionicima postaje sve teže, budući da se eksponencijalno povećava broj potrebnih komunikacija između igrača, što može dovesti do problema s performansama. Ovaj rast u broju komunikacija opterećuje mrežu, povećava latenciju i može uzrokovati kašnjenja, gubitak podataka ili nestabilnost u prikazu igre kod različitih igrača. Kao rezultat toga, iskustvo igranja može biti narušeno, s vidljivim zastojima, desinkronizacijom i općim padom performansi igre.

3.3.2. Klijent-server arhitektura

Klijent-server arhitektura koristi centralni server koji upravlja svim komunikacijama između igrača. Svi podaci o igri šalju se serveru, koji zatim distribuira potrebne informacije

klijentima. Ovaj model je najčešće korišten u modernim multiplayer igrama, jer omogućuje bolju kontrolu nad podacima i sigurnost. Prednosti klijent-server arhitekture uključuju veću sigurnost, jer server centralno upravlja svim podacima igre i može provoditi zaštitne mjere protiv varanja. Primjer igre koja koristi klijent-server arhitekturu je World of Warcraft. Ova arhitektura omogućuje centraliziranu kontrolu koja održava konzistentnost igre za sve igrače. Također, ovaj model olakšava održavanje sinkronizacije među igračima, što je ključno za igre s velikim brojem sudionika. Nedostaci klijent-server arhitekture uključuju veće troškove održavanja, jer je potrebno ulagati u servere koji mogu podržati veliki broj igrača. Uz to, centralizirana priroda ovog modela može dovesti do problema s latencijom, osobito ako su igrači geografski udaljeni od servera.

3.3.3. Hibridni model

Hibridni modeli kombiniraju elemente P2P i klijent-server arhitekturu kako bi iskoristili prednosti oba pristupa. Na primjer, neki dijelovi igre mogu koristiti P2P komunikaciju za direktne interakcije među igračima, dok se ključni podaci o igri pohranjuju i kontroliraju putem centralnog servera. Ova arhitektura može smanjiti latenciju za određene operacije, dok istovremeno osigurava sigurnost i sinkronizaciju putem servera. Prednosti hibridnih modela uključuju fleksibilnost u dizajnu mrežnih sustava, omogućujući optimizaciju za specifične potrebe igre. Međutim, nedostaci uključuju složenost implementacije, jer je potrebno koordinirati različite vrste komunikacije i osigurati da sustav radi besprijekorno pod različitim uvjetima. Igra Halo Reach koristi hibridni model, gdje se P2P koristi za manje kritične podatke, dok se važni podaci prenose putem servera.

3.3.4. Model klijent-domaćin

Model klijent-domaćin (client-host), poznat i kao "listen server", kombinira elemente klijent-server i P2P modela. U ovom modelu, jedan od igrača djeluje kao domaćin (host) igre, dok ostali igrači djeluju kao klijenti koji se povezuju na domaćina. Domaćin ne samo da igra igru, već i upravlja svim aspektima mrežne igre, poput servera. Prednosti modela klijent-domaćin uključuju niže troškove jer nije potreban namjenski server i jednostavnije postavljanje za male grupe igrača. Međutim, glavni nedostatak je što, ako domaćin napusti igru ili dođe do problema s njegovom vezom, cijela igra može biti prekinuta. Također, domaćin može imati prednost u latenciji, što može utjecati na pravednost igre. Zbog prednosti koje nudi ovakva arhitektura biti će korištena u izradi ovog projekta.

Također jedan aspekt o kojem je važno razmisliti kod odabira ovakve arhitekture jest način na koji će igrači pronaći server na koji se žele povezati. Postoji nekoliko načina za to, poput direktnog povezivanja putem IP adrese ili korištenja server browsera. Direktno povezivanje putem IP adrese (engl. Direct IP connection) omogućava brzo i direktno povezivanje, ali je nepraktično za igrače koji ne znaju IP adresu servera. U slučaju Server browsera svaki server koji želi biti dostupan igračima registrira se na centralnom master serveru, šaljući mu svoje podatke poput IP adrese, statusa, broja igrača, geografske lokacije i drugih relevantnih informacija. Kada igrač pokrene server browser unutar igre, njegova igra šalje zahtjev master serveru za popis svih registriranih servera. Master server zatim odgovara s ažuriranim popisom, uključujući sve potrebne podatke koje je dobio od pojedinačnih servera.

U ovom projektu odlučeno je koristiti Steam platformu, omogućavajući igračima da putem liste prijatelja kliknu na "Join Game" i povežu se s igrom koristeći Steamworks API, koji omogućava integraciju i olakšava povezivanje igrača na servere preko Steama.

3.3.5. Predikcija s strane klijenta (engl. Client side prediction) i kompenzacija kašnjenja

Za neke naslove, posebno one s brzim tempom i visokim zahtjevima za preciznošću, poput pucačina (shooters), sinkronizacija je od ključne važnosti. U ovim igrama, čak i mala kašnjenja mogu značajno utjecati na iskustvo igranja, jer igrači moraju reagirati u djeliću sekunde na akcije drugih igrača. Da bi se osigurala sinkronizacija i smanjila percepcija kašnjenja, koriste se razne tehnike poput predviđanja s strane klijenta (engl. client-side prediction) i kompenzacije kašnjenja (lag compensation).

Client side prediction omogućuje klijentima da predvide buduće pozicije objekata, kao što su likovi ili metci, kako bi se smanjila percepcija kašnjenja između ulaza igrača i odgovora igre. Osnovna ideja leži u tome da klijenti lokalno predviđaju kako će se ti objekti kretati na temelju najnovijih dostupnih podataka, umjesto da čekaju potvrdu od servera za svaku promjenu stanja. Ovaj pristup omogućuje klijentima da odmah reagiraju na lokalne akcije poput pucanja ili kretanja, stvarajući dojam trenutne interakcije. Međutim, moguće su razlike između predviđanja klijenata i ažuriranja servera. Kada se takve razlike pojave, server koristi mehanizme poput povratka na prethodno stanje ili usklađivanja (engl. reconciliation) kako bi ispravio klijente, šaljući im ažurirane podatke o trenutnom autoritativnom stanju igračkih objekata. Predviđanje pokreta posebno je važno kod igara kao što su "Call of Duty" ili "Counter-Strike", gdje je brzina reakcije kritična, no u sklopu kooperativnih igri poput Survivor A.R.C. nije toliko nužna [23].

Kompenzacija kašnjenja je tehnika koja se koristi u mrežnim igrama kako bi se smanjio utjecaj mrežnog kašnjenja na igračko iskustvo. Kašnjenje može uzrokovati razliku između onoga što igrač vidi i onoga što se stvarno događa na poslužitelju, što može rezultirati netočnim ili nepoštenim ishodima. Kompenzacija kašnjenja omogućava da igrači imaju konzistentno iskustvo bez obzira na kašnjenje u mreži. Radi tako da poslužitelj simulira prošlost kada obrađuje ulaze igrača. To znači da kada poslužitelj primi ulaz od igrača, uzima u obzir vrijeme koje je potrebno da taj ulaz stigne do poslužitelja. Na temelju toga, poslužitelj pozicionira objekte u stanju u kojem su bili u trenutku kada je igrač poslao ulaz. Na primjer, ako igrač puca prema protivniku u igri poput „Counter-Strike“, poslužitelj će vratiti stanje igre na trenutak kada je metak ispaljen, te će provjeriti je li metak pogodio protivnika u tom trenutku. Na ovaj način, čak i ako postoji mrežno kašnjenje, igrač će imati osjećaj da se njegovi potezi događaju točno u trenutku kada ih izvrši, čime se poboljšava cjelokupno igračko iskustvo [24].

3.4. Rješenja za umrežavanje u Unityu

Iz razloga što će projekt u ovom radu biti napravljen pomoću platforme Unity, potrebno je bilo odlučiti koja će se infrastruktura koristiti i koji tip umrežavanja će biti implementiran. Unity nudi različite opcije za umrežavanje (slika 20), od kojih svaka ima svoje specifične prednosti i nedostatke [6]. Neke od najpopularnijih biblioteka za umrežavanje u Unityu uključuju Mirror Networking, Fish-Net, Photon i Unity Networking (MLAPI).

	Stability/ support	Ease-of-use	Perfor-mance	Scalability	Feature breadth	Cost*	Customers recommend for
MLAPI	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★	Free	Most client-server games for up to ~64 players that want a stable breadth of mid-level features
DarkRift 2	★★★★★	★★★☆☆	★★★★★	★★★★★	★★★☆☆	$100 for source	Games with high perf/ scale requirements that want to build on a stable LL layer
Photon PUN	★★★★☆	★★★★★	★★★☆☆	★☆☆☆☆	★★★★☆	$0.30/PCU	Simple and small (2–8 players) mesh-topology games
Photon Quantum 2.0	★★★★☆	★★★★☆	★★★★★	★★☆☆☆	★★★★☆	$1000/mo + $0.50/PCU	Games desiring deterministic roll-back, like MOBA games, for up to 32 players
Mirror	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆	Free	Stable and proven client-server solution, loved best for its community and ease-of-use

* Note that Photon pricing provides access to the networking libraries and services, whereas other solutions are standalone networking libraries, and the cost of services is separate.

Slika 19 Usporedba rješenja za umrežavanje u Unityu. Izvor: [6]

Mirror Networking je open-source rješenje koje se temelji na starom Unity UNet sustavu, ali je znatno poboljšano i redovito se ažurira. Prednosti Mirror Networkinga uključuju jednostavnost korištenja, veliku zajednicu korisnika te dobru dokumentaciju (Slika 19). Također, Mirror podržava širok spektar topologija umrežavanja, uključujući klijent-server, P2P i klijent-domaćin. Ova fleksibilnost omogućuje programerima da odaberu najprikladniji model za svoj projekt.

Fish-Net je još jedno open-source rješenje koje se fokusira na visoku skalabilnost i performanse. Fish-Net nudi napredne mogućnosti optimizacije mrežnog prometa i sinkronizacije podataka, što ga čini pogodnim za igre koje zahtijevaju veliki broj simultanih igrača. Iako je Fish-Net moćan alat, njegova kompleksnost može predstavljati izazov za manje iskusne programere. Photon je komercijalno rješenje za umrežavanje koje pruža robusne i pouzdane usluge. Photon nudi različite varijante svog proizvoda, uključujući Photon PUN (Photon Unity Networking) i Photon Fusion, koje su prilagođene različitim potrebama

razvojnih timova. Prednosti Photona uključuju odličnu podršku i dokumentaciju, kao i integraciju s drugim uslugama, poput PlayFab-a za upravljanje korisnicima i podacima. Međutim, korištenje Photona može biti skuplje u usporedbi s open-source rješenjima.

Unity Networking, također poznat kao MLAPI (Mid-level API), najnovije je rješenje za umrežavanje koje je Unity službeno predstavio prije godinu dana. MLAPI je dizajniran da bude modularan, fleksibilan i jednostavan za korištenje, omogućujući developerima da brzo implementiraju mrežne funkcionalnosti u svojim igrama. Prednosti MLAPI-a uključuju jednostavnost integracije s Unity platformom, podršku za različite mrežne topologije i aktivnu podršku i ažuriranja od strane Unity tima.

S obzirom na to da će igra u ovom radu biti co-op tipa, najprikladnija arhitektura za umrežavanje je model klijent-domaćin (client-host). U ovom modelu, jedan od igrača djeluje kao domaćin (host), dok ostali igrači djeluju kao klijenti koji se povezuju na domaćina. Ovaj pristup omogućuje niže troškove i jednostavnije postavljanje za manje grupe igrača, što je idealno za co-op igre.

Za implementaciju umrežavanja u ovom projektu koristit će se Mirror Networking. Mirror nudi jednostavan način postavljanja klijent-domaćin arhitekture, uz visoku razinu fleksibilnosti i podrške. Osim toga, kao open-source rješenje, Mirror omogućuje potpunu kontrolu nad kodom i mogućnost prilagodbe specifičnim potrebama projekta. U usporedbi s drugim rješenjima, Mirror Networking nudi dobru ravnotežu između jednostavnosti korištenja i naprednih mogućnosti.

Slika 20 Rješenja za umrežavanje u Unityu. Izvor: [7]

Za co-op wave shooter igru Survivor A.R.C odabrana je klijent-domaćin arhitektura, gdje jedan igrač djeluje kao domaćin (host) dok ostali igrači djeluju kao klijenti. Ovakva arhitektura je poželjna jer omogućuje lakše upravljanje podacima i sinkronizaciju između igrača. Iako je moguće izraditi ovakvu igru bez klijent-host arhitekture koristeći, na primjer, potpuno distribuiranu peer-to-peer mrežu, klijent-domaćin arhitektura pruža stabilniju i učinkovitiju

komunikaciju te smanjuje latenciju. Alternativno, moglo bi se koristiti dedicirane servere, koji bi osigurali još veću stabilnost i performanse. Međutim, takvo rješenje je preskupo i logistički zahtjevno za implementaciju, te stoga nema smisla za manje razvojne timove. Na temelju tih potreba odabrana je klijent-domaćin arhitektura i Mirror se pokazao kao optimalno rješenje za ovakvu vrstu arhitekture.

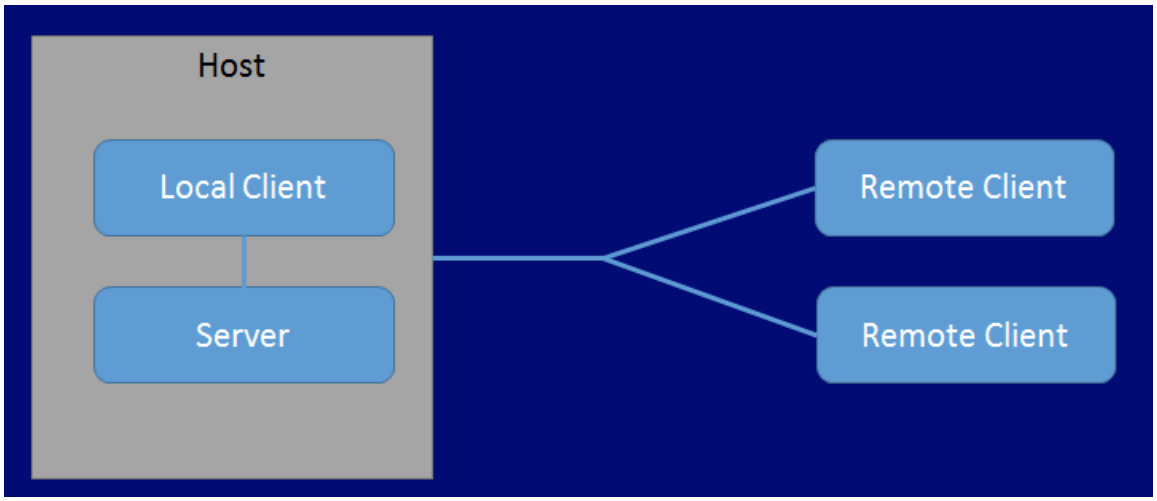
3.5. Sinkronizacija stanja igre

Jedan od ključnih izazova u razvoju multiplayer igara je sinkronizacija stanja igre među svim igračima. Na primjeru co-op wave shootera, zamislimo da imamo igrače koji izravno kontroliraju svoje likove te valove neprijatelja koje generira igra (Slika 21). U co-op wave shooter igri, svaki igrač kontrolira svoj lik, dok neprijatelji dolaze u valovima i napadaju igrače. U ovakvom scenariju, sinkronizacija igra ključnu ulogu u osiguravanju da svi igrači vide isto stanje igre bez značajnih kašnjenja ili razlika. Server (host) je odgovoran za generiranje i kontrolu neprijatelja. On proračunava kretanje neprijatelja, njihove napade i interakcije s igračima. Također, server bilježi stanje svih neprijatelja i šalje ta ažuriranja svim klijentima. Klijenti, s druge strane, šalju informacije o akcijama svojih likova (poput kretanja, pucanja i korištenja posebnih sposobnosti) serveru, koji te podatke obrađuje i šalje ažuriranja natrag svim klijentima kako bi svi igrači vidjeli iste akcije u stvarnom vremenu. Jedna od alternativa je da klijenti sami proračunavaju kretanje svojih likova i lokalno simuliraju neprijatelje, ali to može dovesti do desinkronizacije zbog mrežnih kašnjenja i različitih brzina računanja. Odlučeno je da server proračunava sve ključne aspekte igre kako bi se osigurala konzistentnost stanja igre za sve igrače. Klijenti razmjenjuju samo ključne informacije, poput pozicija likova i rezultata njihovih akcija, dok server održava centraliziranu kontrolu nad stanjem igre.

Slika 21 Prikaz dva igrača unutar igre

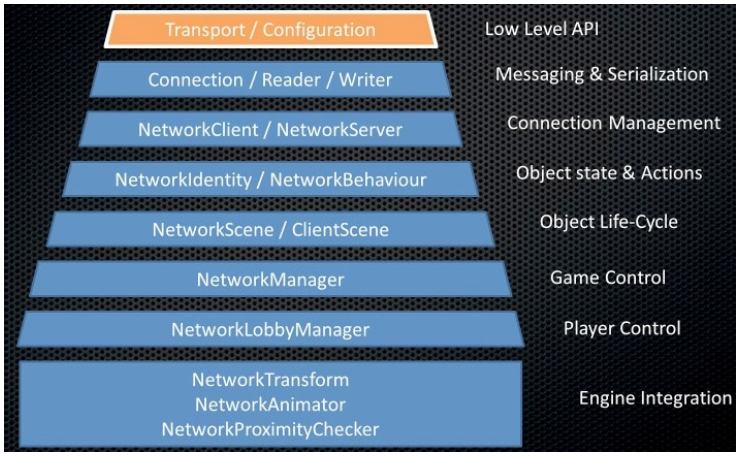
Za implementaciju sinkronizacije stanja, u ovom projektu je korišten Mirror Networking, open-source mrežno rješenje specifično dizajnirano za Unity. Mirror Networking može koristiti različite mrežne arhitekture, uključujući peer-to-peer, klijent-server i klijent-host arhitekturu vidljivu na slici 22, ali je najbolje razvijena klijent-host arhitektura, što ga čini posebno pogodnim ako je odabrana ta arhitektura.

Implementacija Mirror Networkinga započinje uvođenjem Mirror paketa u Unity projekt. Nakon što je paket instaliran, započinje konfiguracija mrežnih postavki unutar Unity Editora. Ovo uključuje definiranje mrežnih identifikatora, pridruživanje mrežnih komponenti objektima u sceni i konfiguriranje mrežnih postavki kao što su maksimalni broj igrača, propusnost i sl. Jedna od ključnih odluka prilikom implementacije Mirror Networkinga je odabir transportnog sloja za komunikaciju između klijenata i servera. U ovom projektu korišten je Fizzy Steamworks za uspostavu veze preko Steam platforme. FizzySteamworks je transportni sloj za Mirror Networking koji omogućuje povezivanje igrača putem Steam platforme. Pruža pouzdanu i sigurnu mrežnu komunikaciju koristeći Steamov API, što olakšava povezivanje igrača i osigurava stabilnu mrežnu vezu s minimalnim kašnjenjem [17].



Slika 22 Dijagram s 3 igrača. Izvor: [8]

Nakon instalacije Mirror paketa, svaki objekt može biti opremljen s komponentama poput NetworkIdentity i NetworkTransform (slika 23). Ove komponente omogućuju objektima da komuniciraju preko mreže. Primjerice, NetworkIdentity određuje identitet objekta u mreži, dok NetworkTransform sinkronizira transformacije objekta među različitim klijentima i poslužiteljem. Nakon što su komponente postavljene, programeri mogu koristiti atribute poput Command i ClientRpc u skriptama kako bi izvršavali zadatke preko mreže [8].



Slika 23 Slojevi funkcionalnosti Mirror Networking-a. Izvor: [8]

Metoda označena atributom (Command) je metoda koja se izvršava samo na poslužitelju, ali se može pozvati s klijenta. To omogućuje klijentima da šalju zahtjeve poslužitelju da izvrši određene zadatke. Na Slici 24. može se vidjeti korištenje Command metoda kako bi jedan od priključenih klijenata promijenio stanje rezultata. U slučaju ovog projekta to se pozivanje događa svaki put kada igrač uspješno porazi neprijatelja.

S druge strane, metoda označena atributom ClientRpc (Client Remote Procedure Call) je metoda koja se izvršava na klijentu, ali se može pozvati s poslužitelja. Ova komanda omogućuje poslužitelju da šalje ažuriranja klijentima o događajima koji se događaju na mreži. Korištenje ovih komandi omogućuje programerima da izgrade kompleksne interakcije između klijenata i poslužitelja te upravljaju sinkronizacijom podataka i ponašanja objekata preko mreže. Na primjer, korištenjem [ClientRpc], omogućeno je da se animacija pucanja igrača prikaže na svim klijentima istovremeno. Kada klijent pritisne tipku za pucanje, na njegovom klijentu se pokreće animacija pucanja. Nakon toga se poziva [Command] funkcija koja šalje informaciju serveru o tome da je klijent izvršio pucanj. Server zatim koristi [ClientRpc] kako bi obavijestio sve ostale klijente o tom događaju, čime se osigurava da se animacija pucanja sinkronizira i prikaže svim sudionicima igre istovremeno.. Na slici 25. vidimo dio koda objašnjenog u prijašnjem primjeru gdje se uz pomoću [ClientRpc] funkcije osigurava sinkronizacija leta metka nakon što igrač pritisne tipku za pucanj. Funkcija nakon što je pozvana s strane servera uzima objekt koji simulira let metka te ga prosljeđuje u funkciju „TrailFallOff“ koja kroz vrijeme pomiče objekt i na taj način ostvaruje dojam leta metka.

```
[Command]
1 reference
private void CmdDecreaseScore(int amount)
{
    if (ScoreManager.Instance != null)
    {
        ScoreManager.Instance.DecreaseScore(amount);
    }
}
```

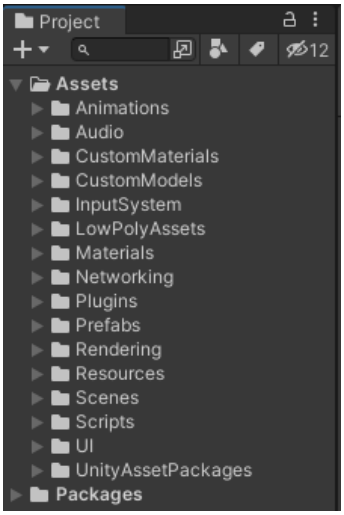
Slika 24 Primjer korištenja atributa [Command]

```
[ClientRpc]
1 reference
public void RpcShowBulletTrail(Vector3 hitPoint, Vector3 hitNormal, Vector3 rayOrigin)
{
    if (!hasAuthority)
    {
        GameObject bulletTrail = GetBulletFromPool();
        StartCoroutine(TrailFalloff(bulletTrail, hitPoint, hitNormal, rayOrigin));
    }
}
```

Slika 25 Primjer korištenja atributa [ClientRpc]

4. Implementacija projekta

Implementacija projekta obuhvaća sve korake potrebne za realizaciju zamišljenih ideja i koncepta igre u stvarnom, funkcionalnom obliku. Ovaj proces uključuje nekoliko ključnih faza, počevši od odlučivanja arhitekture i postavljanja mrežne infrastrukture, preko razvoja gameplay komponenti, do sinkronizacije podataka i optimizacije performansi. Prvi korak u razvoju igre jest postavljanje osnovnog okvira projekta unutar Unity-a. Ovaj korak uključuje stvaranje novog Unity projekta, postavljanje osnovnih postavki za renderiranje i unos, te organiziranje strukture mapa i datoteka, vidljivo na slici 26. Važno je od početka definirati jasnu strukturu projekta kako bi kasnija implementacija bila što učinkovitija i preglednija.



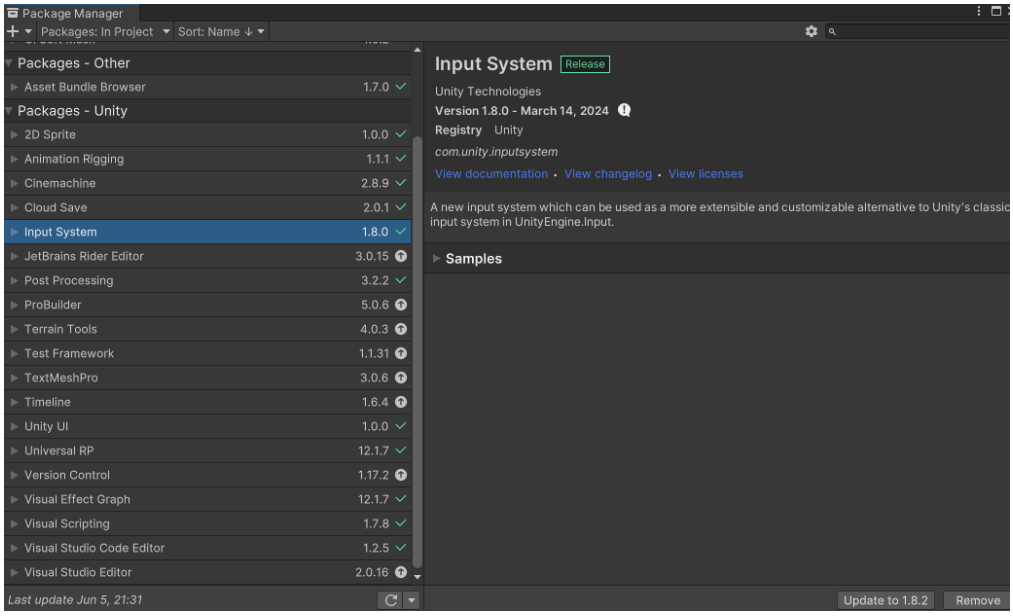
Slika 26 Organizacija mapa.

Nakon postavljanja projekta, slijedi faza detaljnog planiranja i razmišljanja o ključnim aspektima igre. Ovo uključuje definiranje glavnih gameplay mehanika, odlučivanje o arhitekturi mrežne infrastrukture, te izbor alata i tehnologija koje će se koristiti. U ovom trenutku, važno je imati jasnu viziju o tome kako će se igra razvijati, te koji su ciljevi koje treba postići u svakoj fazi razvoja. Sama implementacija započinje postavljanjem mrežne infrastrukture, što uključuje konfiguraciju Mirror paketa za mrežnu komunikaciju i odabir odgovarajućeg transport layera. Za ovaj projekt odabrana je Fizzy Steamworks tehnologija koja

omogućava povezivanje igrača preko Steam platforme. Nakon što je mrežna infrastruktura postavljena, prelazi se na izradu gameplay komponenti, uključujući razvoj kontrolera za igrače, dizajn neprijatelja i umjetne inteligencije, te implementaciju sustava bodovanja i napredovanja. Svaka od ovih komponenti zahtijeva pažljivo planiranje i testiranje kako bi se osiguralo da igra funkcionira glatko i da pruža zadovoljavajuće iskustvo igranja. Sinkronizacija podataka između klijenata i servera ključan je aspekt u multiplayer igrama, te će biti detaljno obrađena kako bi se osiguralo da svi igrači imaju dosljedno iskustvo igre. Uz to, bit će opisani i koraci dizajna korisničkog sučelja i 3D modela, koji su važni za vizualnu privlačnost igre. Na kraju, implementacija završava fazom optimizacije i poboljšanja performansi igre. Ovo uključuje detaljnu analizu i optimizaciju koda, renderiranja i mrežnih operacija kako bi se osigurala stabilnost i visoka razina performansi na svim podržanim platformama.

4.1. Asseti i alati

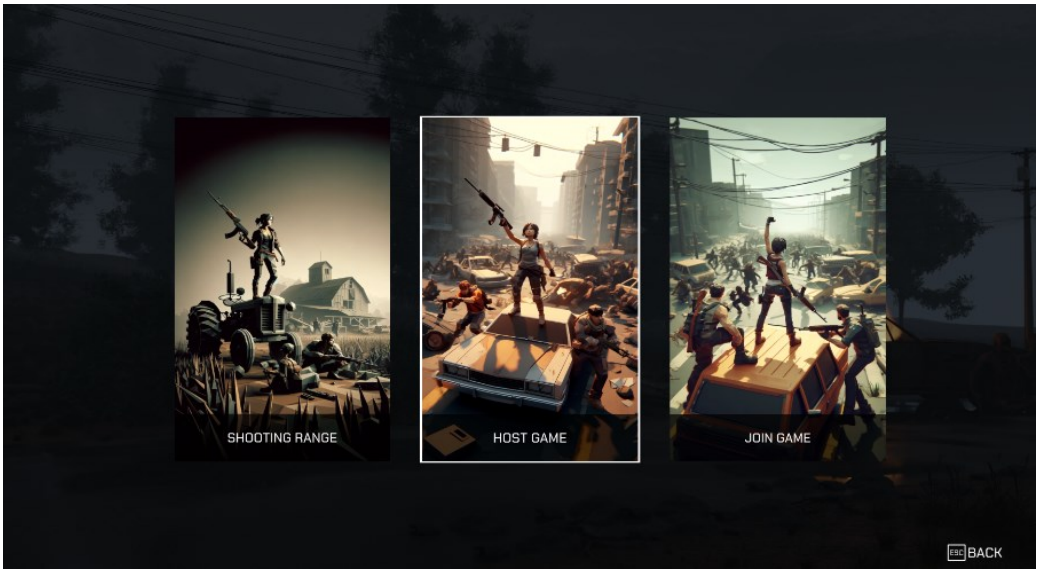
U razvoju Survivor A.R.C.-a većina komponenti je razvijena iznova, bez prevelikog oslanjanja na vanjska sredstva (engl. assets). Assets u kontekstu razvoja igara u Unityju odnosi se na gotove resurse poput modela, tekstura, zvukova ili skripti koje su razvijali drugi korisnici ili kompanije te se mogu koristiti u vlastitim projektima radi uštede vremena i resursa. Unity nudi Package Manager (Slika 27) kao alat koji omogućuje integraciju i upravljanje raznim paketima (packages) ili assetima direktno iz Unity Asset Storea ili drugih izvora. Ovaj pristup omogućio je veću fleksibilnost i prilagodbu specifičnim potrebama projekta, osiguravajući jedinstvenost i optimiziranost svih aspekata igre. Za upravljanje unosima korisnika korišten je novi Unity Input System. Ovaj sustav omogućava fleksibilniju i jednostavniju integraciju raznih kontrola, uključujući tipkovnicu, miš i kontrolere, što poboljšava korisničko iskustvo i prilagodljivost igre različitim platformama. Universal Render Pipeline (URP) korišten je za prikazivanje (engl. rendering), pružajući visokokvalitetne vizualne efekte uz optimizirane performanse. URP omogućuje napredne grafičke mogućnosti i efikasnost, osiguravajući da igra izgleda impresivno na svim uređajima, od računala do mobilnih platformi. Mirror asset korišten je za mrežnu komunikaciju, omogućujući pouzdano i efikasno implementiranje multiplayer funkcionalnosti. Osim Mirrora, korišten je i DoTween za animiranje različitih elemenata u igri. DoTween je moćan alat za animacije koji omogućuje jednostavno stvaranje kompleksnih animacijskih sekvenci putem skripti. Za izradu terena korišteni su određeni third-party asseti, što je značajno ubrzalo proces stvaranja kompleksnih i vizualno atraktivnih okruženja [11]. Ovi asseti omogućili su stvaranje mapa, koje pružaju bogato iskustvo igranja. Za upravljanje pokretima neprijatelja korišten je NavMeshAgent, Unityjev alat za navigaciju i umjetnu inteligenciju. NavMeshAgent omogućuje neprijateljima da se kreću kroz kompleksne terene, slijedeći unaprijed definirane rute i izbjegavajući prepreke. Ovaj alat je ključan za stvaranje realističnog ponašanja neprijatelja i osiguranje izazovnog gameplaya. Osim ovih glavnih alata, korišteni su i razni drugi Unity alati i skripte za specifične potrebe projekta. Na primjer, Unityjev Audio Mixer korišten je za upravljanje zvukovima i glazbom u igri. Također su korišteni razni alati za optimizaciju performansi, uključujući profiler alate za praćenje i poboljšanje performansi igre na različitim uređajima.



Slika 27 Prikaz korištenih asseta.

4.2. Postavljanje mrežne infrastrukture

Postavljanje mrežne infrastrukture ključno je za omogućavanje multiplayer funkcionalnosti u igri. Prvi korak u ovom procesu je instalacija Mirror paketa, koji će služiti kao osnovni okvir za mrežnu komunikaciju unutar igre. Nakon instalacije Mirror paketa, potrebno je kreirati Network Manager objekt unutar Unity-ja. Ovaj objekt će imati prilagođenu NetworkManager komponentu koja će služiti kao središnje mjesto za upravljanje svim mrežnim funkcijama igre. Network Manager omogućava praćenje i kontrolu svih aspekata mrežne komunikacije i interakcije između klijenata i servera. Kroz ovaj objekt, tijekom igre će se moći komunicirati i dohvaćati informacije o igračima, njihovim stanjima, te upravljati njihovim sesijama. Kada igrač želi započeti novu igru, on klikne gumb za kreaciju lobby-a (slika 28). Ova akcija pokreće proces kreiranja host servera.



Slika 28 Mogućnost kreiranja servera.

Host server je instanca igre koja omogućuje povezivanje drugih igrača. Kroz integraciju sa Steam platformom, koristeći Fizzy Steamworks, ostali korisnici se mogu lako priključiti lobby-u. Fizzy Steamworks osigurava pouzdanu i sigurnu konekciju preko Steamovog sustava, što pojednostavljuje proces povezivanja i upravljanja igračima. Network Manager igra ključnu ulogu u upravljanju povezivanjem igrača. Kada se novi igrač pridruži lobby-u, Network Manager mu dodjeljuje jedinstveni identifikator i odgovarajuće privilegije. Također, Network Manager prati sve aktivne veze, upravlja mrežnim sesijama i osigurava da su svi podaci pravilno sinkronizirani između klijenata i servera. U trenutku kada igrač započne hosta server ili se pridruži u postojeći lobby „CustomNetworkManager“ također kreira glavni gameObject igrača koji će od trenutka biti u igri sve dok taj igrač ne izađe iz sesije. Na slici 29. su vidljive dvije glavne funkcije koje Network manager upravlja i to su „OnServerAddPlayer“ i „StartGame“.

```
public class CustomNetworkManager : NetworkManager
{
    [SerializeField] private CharControllerPlayer GamePlayerPrefab;
    [SerializeField] private GameObject Level2UI;


    public List<CharControllerPlayer> GamePlayers = new List<CharControllerPlayer>();

    5 references
    public override void OnServerAddPlayer(NetworkConnectionToClient conn)
    {
        if (SceneManager.GetActiveScene().name == "Lobby")
        {
            CharControllerPlayer GamePlayerInstance = Instantiate(GamePlayerPrefab);

            GamePlayerInstance.ConnectionID = conn.connectionId;
            GamePlayerInstance.PlayerIDnumber = GamePlayers.Count + 1;
            GamePlayerInstance.PlayerSteamID = (ulong)SteamMatchmaking.GetLobbyMemberByIndex((CSteamID)SteamLobby.Instance.CurrentLobbyID, GamePlayers.Count);
            NetworkServer.AddPlayerForConnection(conn, GamePlayerInstance.gameObject);

        }
    }

    1 reference
    public void StartGame(string SceneName)
    {
        ServerChangeScene(SceneName);
    }
```

Slika 29 Custom Network Manager.

Unutar igre, Network Manager omogućava hostu da upravlja različitim aspektima igre. Ovo uključuje sinkronizaciju stanja igre, praćenje i ažuriranje stanja igrača, te slanje i primanje mrežnih poruka. Na primjer, kada igrač izvrši neku akciju, Network Manager osigurava da se ta akcija pravilno prenese svim ostalim igračima kako bi svi imali dosljedno iskustvo igranja.

Zahvaljujući korištenju Steam platforme, Network Manager može povući informacije o igračima kao što su njihovo ime, slika profila i jedinstveni Steam ID. Ovi podaci će se kasnije koristiti za spremanje podataka u bazi, što omogućava praćenje napretka igrača, pohranu postignuća i personalizaciju iskustva unutar igre. Kroz sve ove funkcionalnosti, Network Manager osigurava stabilno i pouzdano multiplayer iskustvo. On je ključna komponenta koja omogućava sinkronizaciju svih mrežnih aktivnosti, te omogućava hostu da efikasno upravlja igrom i održava konstantnu komunikaciju s klijentima.

Nakon uspješnog kreiranja objekata igrača te njihovog uspostavljanja dodaju se skripte. Ukoliko je potrebno da se akcije iz tih skripti sinkroniziraju sa ostatkom igrača u multiplayer igri koristeći Mirror, potrebno je da ta skripta nasljeđuje klasu NetworkBehaviour. Ova klasa pruža osnovne funkcionalnosti i metode koje omogućuju mrežnu komunikaciju i sinkronizaciju stanja između klijenata i servera. Prvi korak je osigurati da objekt igrača posjeduje komponentu NetworkIdentity. Ta komponenta jedinstveno identificira objekt unutar mreže i omogućava Mirror sustavu praćenje i sinkronizaciju njegovog stanja. Bez NetworkIdentity komponente, objekt se neće moći pravilno sinkronizirati ili komunicirati s mrežnim sustavom.

Kada skripta nasljeđuje NetworkBehaviour i objekt ima NetworkIdentity komponentu, moguće je koristiti posebne metode i atribute koje Mirror pruža za mrežnu komunikaciju:

- **Command** metode: Ove metode se pozivaju na klijentskoj strani, ali se izvršavaju na serveru.
- **ClientRpc** metode: Ove metode se pozivaju na serveru, ali se izvršavaju na svim povezanim klijentima.
- **SyncVar** atribute: Ovi atributi se koriste za automatsku sinkronizaciju varijabli između servera i klijenata.

Dodatno, moguće je koristiti svojstva kao što su isLocalPlayer i hasAuthority za provjeru specifičnih stanja i ovlasti objekta u mrežnom okruženju. isLocalPlayer određuje je li trenutni objekt lokalni igrač na klijentu (slika 30), dok hasAuthority provjerava ima li objekt mrežnu ovlast za izvršavanje određenih radnji, poput primjera s slike 31 gdje se koristi kao kondicija za promjene stanja spremnosti.

```
public void ChangeReady()
{
    if (hasAuthority)
    {
        CMDSetPlayerReady();
    }
}
```

Slika 30 Primjer korištenja „hasAuthority“ za izmjenu stanja spremnosti.

```
if (!isLocalPlayer && !dead)
{
    RotateGunTowards(currentAimPos);
    RotateTowards(lookAtPositionSync);
}
```

Slika 31 Primjer korištenja "isLocalPlayer".

4.3. Dizajn multiplayer komponenti

Dizajn multiplayer komponenti u Survivor A.R.C. igri temelji se na korištenju arhitekture klijent domaćin implementiranih pomoću mrežnih alata i funkcionalnosti koje pruža Mirror Networking. Prvi korak u dizajnu multiplayer komponenti u Survivor A.R.C. igri je sinkronizacija ključnih elemenata igre, kao što su rezultat, pozicije i broj neprijatelja, te pozicije metaka. Sinkronizacija rezultata je neophodna jer svi igrači moraju vidjeti i koristiti točne informacije o stanju igre u stvarnom vremenu. Pozicije svih objekata, uključujući igrače, neprijatelje i metke, također moraju biti sinkronizirane kako bi igra imala smisla iz online perspektive. Bez ovih sinkronizacija, igrači bi vidjeli različite scenarije, što bi rezultiralo kaotičnim iskustvom. Server igra ključnu ulogu u ovom procesu, odlučujući i šaljući klijentima informacije o novim neprijateljima, te izračunavajući njihove pozicije koristeći AI. Glavna zadaća servera je sinkronizacija ciljeva svakog neprijatelja. Nakon što server odredi cilj neprijatelja, svaki objekt neprijatelja koristi NavMeshAgent kako bi izračunao najbliži put do igrača i omogućio neprijatelju da ih napadne. Na ovaj način, neprijatelji mogu napadati igrače, a svi igrači vide iste pokrete i akcije neprijatelja.

Mirror omogućuje jednostavnu implementaciju i sinkronizaciju multiplayer elemenata igre, što je ključno za osiguravanje konzistentnog i responzivnog iskustva igranja za sve igrače. Jedan od ključnih aspekata dizajna je korištenje NetworkTransform i NetworkIdentity komponenti (Slika 33).

NetworkIdentity komponenta osigurava jedinstveni identitet svakog mrežnog objekta u igri. Ova komponenta je neophodna za bilo koji objekt koji treba biti sinkroniziran preko mreže, jer omogućuje Mirroru da prepoznaje i upravlja mrežnim objektima. Primjerice, zamislimo multiplayer igru u kojoj svaki igrač upravlja vlastitim likom. Svaki lik u igri mora imati NetworkIdentity komponentu kako bi mu bio dodijeljen jedinstveni identifikator. Taj identifikator omogućuje serveru i klijentima da prate koji objekt pripada kojem igraču. Bez NetworkIdentity komponente, ne bi bilo moguće razlikovati između različitih igrača ili drugih mrežnih objekata.

NetworkTransform komponenta, s druge strane, omogućuje sinkronizaciju pozicije, rotacije i skale objekata između svih klijenata u mreži, kada jedan igrač pomakne svog lika, ta promjena mora biti vidljiva svim ostalim igračima u igri. Ovo je posebno važno za objekte kao što su igrači i neprijatelji, gdje je precizna sinkronizacija pozicije ključna za pravilno funkcioniranje igre.

Cijeli gameplay je sinkroniziran kako bi se postiglo najbolje moguće iskustvo za sve igrače. Na primjer, računanje rezultata (engl. score) se obavlja na centralnom upravitelju rezultata (engl. Score Manager). Kao što je prikazano na slici 32. Score Manager koristi SyncVar varijablu za praćenje rezultata tima. SyncVar je varijabla koju Mirror koristi za automatsku sinkronizaciju podataka između servera i klijenata. Kada se rezultat promijeni na serveru, SyncVar osigurava da su te promjene odmah vidljive svim klijentima. Ovo osigurava da svi igrači uvijek imaju točan i ažuran rezultat, bez obzira na latenciju ili druge mrežne probleme. Sav pokret i akcije igrača također su sinkronizirani putem mreže. Ovo uključuje

kretanje igrača, interakcije s okolinom, te akcije poput pucanja i korištenja posebnih sposobnosti.

Korištenje NetworkTransform komponente omogućuje glatko i precizno praćenje pozicije i pokreta igrača u stvarnom vremenu. Na primjer, kada jedan igrač pomakne svog lika, NetworkTransform osigurava da se ta promjena odmah prenese na sve druge igrače, stvarajući iluziju zajedničkog prostora u kojem su svi igrači u međusobnoj interakciji. Neprijatelji (engl. enemies) se također stvaraju (engl. spawn) preko mreže koristeći Mirror. Kada server odluči da je vrijeme za spawnanje novih neprijatelja, koristi se metoda za mrežno spawnanje koja osigurava da se novi neprijatelji pojave na svim klijentima. Njihovo kretanje i akcije također su sinkronizirane koristeći NetworkTransform komponentu. Ovo omogućuje precizno praćenje i prikazivanje kretanja neprijatelja na svim klijentima, što je ključno za održavanje konzistentnosti i pravičnosti igre. Napadi neprijatelja sinkronizirani su na sličan način, osiguravajući da svi igrači vide iste akcije i mogu pravovremeno reagirati na prijetnje.

Korištenje Fizzy Steamworks kao transport layera omogućuje povezivanje igrača preko Steam platforme [10]. Ovo znači da se konekcija na hosta odvija preko Steama, što olakšava povezivanje igrača i upravljanje sesijama. Steam također pruža dodatne pogodnosti kao što su prijateljske liste, pozivnice u igru i druge socijalne funkcionalnosti koje poboljšavaju iskustvo igranja. Implementacija multiplayer funkcionalnosti pomoću Mirror paketa omogućuje efikasno i skalabilno upravljanje mrežnim elementima igre. Svi igrači u mreži dijele istu instancu igre, a server djeluje kao autoritativni entitet koji osigurava dosljednost svih važnih podataka i događaja u igri. Ovaj pristup omogućuje stabilno i pouzdano multiplayer iskustvo, čak i kada su igrači raspoređeni na različitim fizičkim lokacijama.

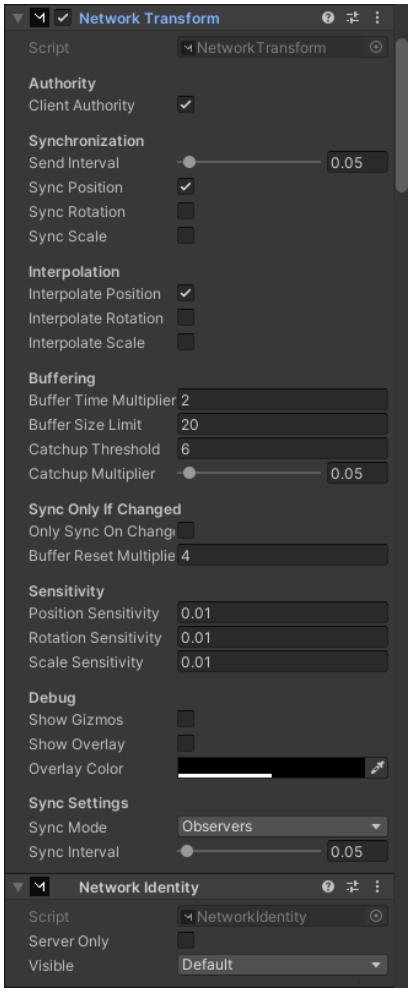
```
[SyncVar(hook = nameof(OnTeamScoreChanged))]
public int TeamScore;

1 reference
private void OnTeamScoreChanged(int oldScore, int newScore)
{
    OnScoreChanged?.Invoke(newScore);
}

[Server]
1 reference
public void IncreaseScore(int scoreAmount)
{
    TeamScore += scoreAmount;
}

[Server]
2 references
public void DecreaseScore(int scoreAmount)
{
    TeamScore -= scoreAmount;
}
```

Slika 32 Korištenje SyncVar varijable.



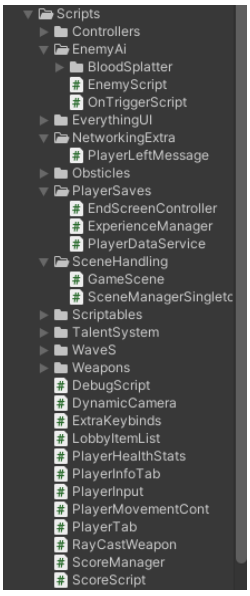
Slika 33 Prikaz NetworkTransform i NetworkIdentity komponenti.

4.4. Izrada gameplay komponenti

Izrada gameplay komponenti ključni je dio razvoja svake igre, jer direktno utječe na iskustvo igrača i kvalitetu igranja. Za projekt Survivor A.R.C. osnovni elementi uključuju mehaniku kretanja, borbe, interakcije s okolinom i progresiju kroz igru.

Mehanika kretanja u igri je responzivna i intuitivna, omogućujući igračima lako navigiranje kroz 3D svijet. Igra se održava u stvarnom vremenu što omogućuje dinamičnu interakciju s okruženjem, bez ograničenja grid sustava. Igrač se kreće kroz igru iz top-down perspektive uz pomoću kamere koja je postavljena pod kutem i prati svaki korak igrača, što osigurava neprekidni prikaz njihove okoline i omogućuje im bolju orijentaciju u prostoru. Osim toga, fleksibilnost kamere dopušta igračima da preuzmu kontrolu nad njezinim pozicioniranjem te da prebace fokus na druge igrače u timu ako njihov lik umre. Igrač se kreće pomoću tipki WASD, dok mišem cilja i puca, što pruža preciznu kontrolu nad akcijama unutar igre. Jedna od glavnih komponenti odgovornih za kretanje u igri Survivor A.R.C. je „Player Movement Controller“. Ovaj kontroler omgućuje igračima da upravljaju svojim likom. Kao

takav „Player Movement Controller“ upravlja s više različitih komponenti kako bi postigao svoj cilj. Od korištenja „Input Controllera“ koji prati sve ulaze igrača te poziva željene funkcije poput izmjene stanja tračanja s klikom na tipku „Shift“ do „Dynamic Camera“ komponente čiji je cilj pozicionirati kameru iznad odabranog lika unutar igre, „Player Movement Controller“ spaja sve potrebne elemente za fluidno kretanje. Uz kretanje, borba je centralni dio gameplaya, a to je omogućeno putem „Shooting Controller“ komponente. Ovaj kontroler upravlja svim aspektima pucanja, uključujući praćenje streljiva, aktivaciju odgovarajućih animacija i generiranje projektila. Integracija s „Player Movement Controllerom“ osigurava da se akcije poput pucanja i kretanja odvijaju bez poteškoća. Svaka gameplay komponenta najčešće se implementira kao skripta u C# programskom jeziku, koja se zatim dodaje na odgovarajuće gameobjekte unutar igre [12]. Na primjer, komponenta za kretanje igrača „Player Movement Controller“ kontrolira inpute uz pomoću „Input Controllera“ i koristi fizičke komponente poput Rigidbody-a za kretanje i sudaranje s objektima unutar igre. Skripte za borbu poput „RayCastWeapon“ sadržavaju logiku za napade dok skripta „PlayerHealthStats“ se koristi za primjenu štete. Korištenje Unityjevog novog input systema omogućava precizno mapiranje kontrola za igrače, pružajući fleksibilnost u definiranju različitih kontrolnih shema za različite uređaje. Universal Render Pipeline (URP) koristi se za postizanje visoke kvalitete grafike i optimizaciju performansi, osiguravajući da igra teče glatko na različitim platformama.



Slika 34 Prikaz podjele skripti u igri Survivor A.R.C.

Vidljivo na slici 34, igra Survivor A.R.C. koristi razne skripte organizirane u nekoliko mapa kako bi se postigao željeni gameplay i funkcionalnost. Ove skripte su grupirane prema njihovoj funkcionalnosti i dijelovima igre koje upravljaju.

Kontroleri

- CharControllerPlayer: Skripta koja sadržava podatke igrača unutar servera (npr. Ime, steamID, slika).
- PlayerInput: Skripta koja prati unose igrača (npr. Tipka na tipkovnici), te poziva određene funkcije.

- PlayerMovementCont: Skripta koja upravlja s više različitih komponenti (CameraController, PlayerInput) kako bi omogućila igraču kretanje u 3D svijetu.
- CameraController: Skripta za pozicioniranje kamere iznad lika igrača.
- DynamicCamera: Skripta za odabir igrača kojeg kamera prati.
- LobbyController: Skripta za upravljanje lobbyem.

Umjetna inteligencija neprijatelja

- EnemyAI: Skripta koja upravlja ponašanjem neprijatelja, uključujući njihove napade i kretanje.
- EverythingUI
- LobbyItemList: Skripta za popis stavki unutar lobbyja.
- LobbyInfoTab: Skripta za informacijski tab igrača.
- PlayerTab: Skripta za tab igrača.
- PlayerHealthStats: Skripta za praćenje zdravlja igrača.
- ScoreManager: Skripta za upravljanje bodovima.
- ScoreScript: Skripta za bodovanje.

Skripta za spremanje igračevih podataka

- EndScreenController: Skripta za upravljanje završnim ekranom igre.
- ExperienceManager: Skripta koja upravlja iskustvenim bodovima igrača.
- PlayerDataService: Skripta za upravljanje podacima o igraču.

Skripta za upravljanje scenama

- SceneHandling: Skripte koje upravljaju prijelazima između scena.

Skriptabilni objekti (engl. Scriptables)

- Scriptables: Skripte vezane uz ScriptableObjecte, koji se koriste za lakše upravljanje podacima.

4.4.1. Izrada player controllera

U igri Survivor A.R.C. skupina controllera omogućuje igračima da hodaju, trče, pucaju i komuniciraju s objektima poput prepreka namijenjenih za otključavanje novih dijelova mape. Trenutačno Survivor A.R.C. podržava upravljanje uz pomoću tipkovnice i miša no iz razloga što je korišten novi „Input System“ kod praćenja ulaza igrača, moguće je lagano mapirati kontrole od gamepada kao primarni način upravljanja igre. Pri izradi player controllera, važno je osigurati da su svi elementi fluidni što se postiže tako da se osigura brza i odzivna reakcija na igračeve akcije, te da se animacije glatko prilagođavaju svakom koraku i pokretu lika. To

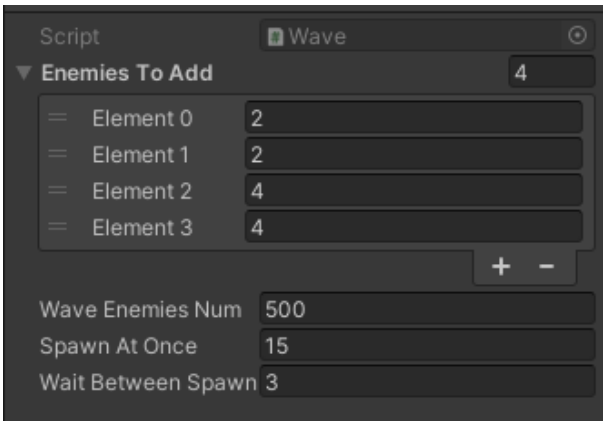
uključuje provjeru kolizija, provjeru animacija i integraciju s drugim sustavima igre, kao što su fizički motor (engl. Physics engine) i umjetna inteligencija.

Za Survivor A.R.C. igru, bilo je potrebno osmisliti sustav koji će podržavati način gledanja u igri gdje je kamera iznad igrača i prati igrača. Uz to, sustav mora podržavati ciljanje pomoću pokreta miša. Zbog kuta kamere, bilo je izazovno osmisliti sustav koji će omogućiti igračima da pucaju točno gdje žele. Ovo je postignuto izradom ravnine tla (engl. groundplane) i korištenjem projiciranih zraka (engl. raycast). Osnovna ideja je bila koristiti ravninu tla za određivanje točke na koju igrač cilja. Kamera iznad igrača prati pokrete miša i projicira zraku od kamere prema tlu. Kada zraka dotakne tlo, moguće je saznati koordinate dodira zrake s tlom. Na taj način, bez obzira na kut kamere, igrač može precizno ciljati. Korišten je i sloj (engl. layer) koji označava površinu tla, kako bi se raycast mogao ograničiti samo na relevantne površine. Dodatno, implementirana je i metoda za provjeru udaljenosti između ciljne točke i igrača, kako bi se osiguralo da igrač uvijek cilja unutar dosega. Ako je cilj preblizu, prilagođava se pozicija kako bi se omogućilo prirodno i realistično ciljanje. Osim ovih tehnika, korištena je i inverzna kinematika (IK) kako bi se osiguralo da igračev lik točno cilja u određeno mjesto (Slika 35).

Slika 35 Prikaz ciljanja uz pomoću inverzne kinematike.

4.4.2. Dizajn neprijatelja i AI

Neprijatelji u igri Survivor A.R.C traže najbližeg igrača te ga postavljaju kao svoj cilj. Nakon toga neprijatelji počinju trčati prema igraču gdje ukoliko dođu dovoljno blizu pokreću svoj napad koji zamahom ruke provjerava ukoliko je došlo do kolizije s igračem te se na taj način nanosi šteta na igrača. Neprijatelji koriste Unityjev NavMeshAgent komponentu koja omogućuje navigaciju kroz teren. NavMeshAgent je ključna komponenta u Unityju koja se koristi za automatizirano kretanje AI likova po unaprijed definiranoj navigacijskoj mreži (NavMesh) [13]. Ova navigacijska mreža predstavlja površinu igre koju AI agenti mogu koristiti za kretanje. Unity koristi informacije o scenama kao što su tereni, prepreke i statični objekti kako bi stvorio mrežu koja određuje gdje AI agenti mogu i ne mogu ići. Površine po kojima se agenti mogu kretati pretvaraju se u poligone, dok se objekti koji blokiraju put identificiraju i dodaju u navigacijsku mrežu kao neprohodne zone. Postupak stvaranja navigacijske mreže na osnovu geometrije scene zove se *bakeanje* (eng. baking). Parametri bakeanja, kao što su maksimalna visina stepenica i nagib površine, također se postavljaju kako bi se simulirala fizička ograničenja kretanja različitih vrsta likova.



Slika 36 Primjer Scriptable objecta za val neprijatelja.

Neprijatelji se stvaraju (engl. spawn) u valovima, što znači da dolaze u unaprijed definiranim vremenskim intervalima i količinama (Slika 37). Svaki val neprijatelja opisan je pomoću Scriptable Objecta koji sadrži informacije o brzini stvaranja i broju neprijatelja, vidljivo na slici 36. Na taj način, lako je prilagoditi težinu igre jednostavnim mijenjanjem parametara u Scriptable Objectu.

Slika 37 Prikaz vala neprijatelja.

```
2 references
private GameObject GetClosestTarget()
{
    GameObject closestTarget = null;
    float closestDistance = Mathf.Infinity;

    foreach (GameObject player in PlayerObjects)
    {
        if (player != null && !player.GetComponent<PlayerMovementCont>().dead)
        {
            float distance = Vector3.Distance(transform.position, player.transform.position);
            if (distance < closestDistance)
            {
                closestDistance = distance;
                closestTarget = player;
            }
        }
    }
    return closestTarget;
}
1 reference
private void UpdateTarget()
{

    timeSinceLastTargetChange += Time.deltaTime;
    if (timeSinceLastTargetChange >= targetChangeInterval || currentTarget == null || currentTarget.GetComponent<PlayerMovementCont>().dead)
    {
        currentTarget = GetClosestTarget();
        timeSinceLastTargetChange = 0f;
        targetChangeInterval = Random.Range(10f, 15f);
    }
}
```

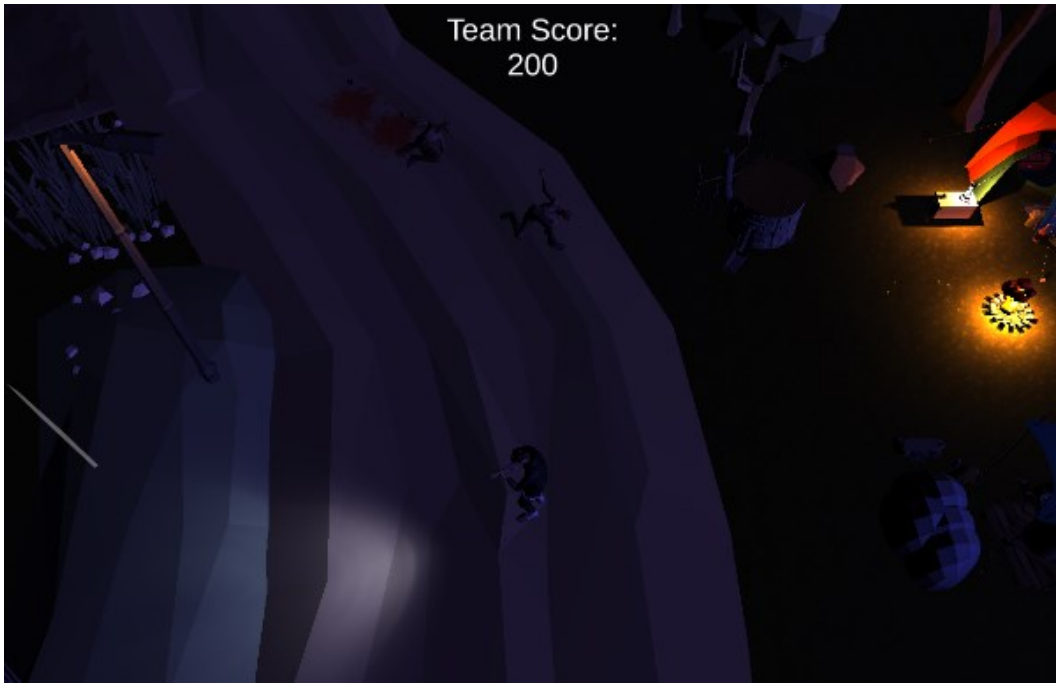
Slika 38 Kod za postavljanje cilja neprijatelja

Slika 38 prikazuje kako neprijatelji u igri traže najbližeg igrača te ga postavljaju kao svoj cilj. Funkcija GetClosestTarget prolazi kroz sve igrače u igri i provjerava udaljenost od neprijatelja do svakog igrača. Ako je igrač najbliži i nije mrtav, postavlja se kao trenutni cilj neprijatelja. Funkcija UpdateTarget redovito provjerava je li potrebno promijeniti cilj, što omogućuje dinamičnu reakciju neprijatelja na promjene u okruženju. U svrhu optimizacije performansi, igra koristi tehnike poput "object poolinga" za upravljanje neprijateljima. Kada neprijatelj umre, njegov se GameObject ne uništava, već se premješta ili deaktivira kako bi se kasnije mogao ponovno koristiti. Ova metoda smanjuje opterećenje na sustav jer izbjegava skupe operacije stvaranja i uništavanja objekata. Za multiplayer aspekt igre svi neprijatelji se

spawnaju i sinkroniziraju preko mreže tako da ih svi igrači mogu vidjeti i interagirati s njima. Kada neprijatelj umre, njegov status se ažurira za sve igrače, osiguravajući dosljedno iskustvo igre.

4.4.3. Sustav bodovanja i napredovanja

Sustav bodovanja i napredovanja u igri Survivor A.R.C. dizajniran je kako bi potaknuo timsku suradnju i strateško odlučivanje među igračima. Bodovi se prikupljaju ubijanjem neprijatelja, pri čemu se količina dobivenih bodova razlikuje ovisno o težini neprijatelja. Na primjer, obični neprijatelji daju 100 bodova. Ovi bodovi se skupljaju kao timski resurs, što znači da svaki igrač može vidjeti trenutni ukupni broj timskih bodova (slika 39). Igrači zatim moraju zajednički odlučiti kako žele iskoristiti te bodove za napredovanje.



Slika 39 Prikaz bodova unutar igre.

Postoje dvije glavne opcije za trošenje timskih bodova. Prva opcija je napredovanje kroz mapu i otključavanje novih dijelova, poput uklanjanja prepreka u obliku vrata, ograda i sličnog. Ove akcije zahtijevaju određeni broj bodova, a igrači moraju strateški planirati kada i gdje će otključavati nove dijelove mape.

Druga opcija je unapređivanje oružja i resursa putem vending machine-a. Korištenjem timskih bodova, igrači mogu kupiti nova oružja i dodatke za oružja, što im pomaže u suočavanju s sve jačim neprijateljima. S tehničke strane, bodovi su pohranjeni kao SyncVar varijabla, što omogućuje sinkronizaciju bodova među svim igračima putem mreže. Sustav bodovanja kontrolira ScoreManager koji je zaseban kontroler unutar igre. Svaki put kada igrač ubije neprijatelja, bodovi se dodjeljuju putem funkcije unutar ScoreManager skripte. Ovaj sustav osigurava da su bodovi uvijek točno ažurirani i dostupni svim igračima u realnom

vremenu. U praksi, kada igrač ubije neprijatelja, poziva se funkcija koja dodaje određeni broj bodova timu. Na primjer, u slučaju ubijanja običnog neprijatelja, dodaje se 100 bodova, vidljivo na slici 40. Ova funkcija također može uključivati logiku za provjeru težine neprijatelja kako bi se dodijelila odgovarajuća količina bodova.

```
[Server]
2 references
public void TakeDamage(float BulletDamage, float bulletForce, Vector3 hitpoint, string NameOfPlayer)
{
    RpcEmitHit();
    EnemyHealth -= BulletDamage;
    if (EnemyHealth <= 0)
    {
        RpcActivateRagdoll(bulletForce, hitpoint);
        ScoreManager.Instance.IncreaseScore(ScoreToGive);
    }
}
```

Slika 40 Funkcija za kontrolu štete nad neprijateljima.

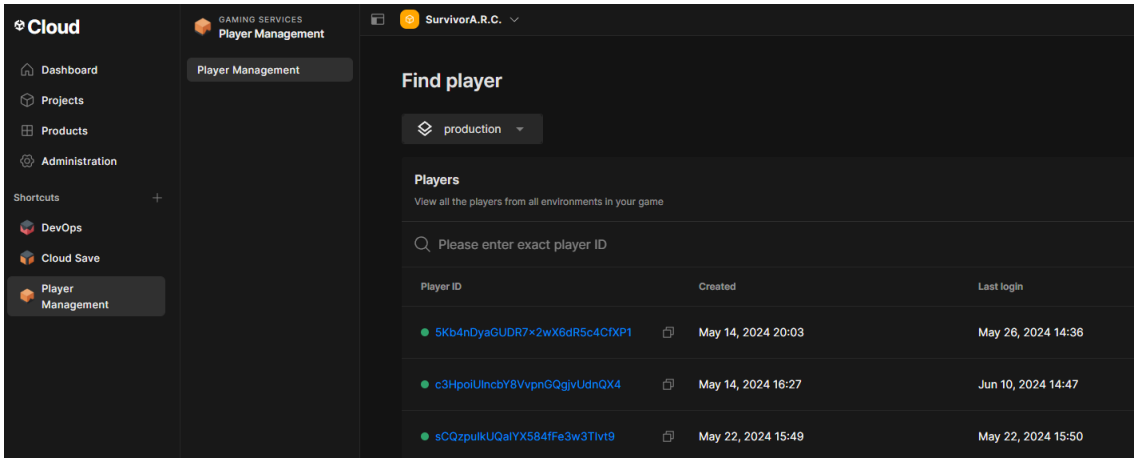
Funkcija „TakeDamage“ implementirana je na serverskoj strani igre Survivor A.R.C. i služi za rukovanje štetom koju neprijatelj prima kada ga pogodi metak. Ova funkcija prihvaća nekoliko parametara: „BulletDamage“ koji označava količinu štete koju metak nanosi, „bulletForce“ koji određuje silu udara metka, „hitpoint“ koji predstavlja točku na neprijateljskom tijelu gdje je metak pogodio, te „NameOfPlayer“ koji sadrži ime igrača koji je ispalio metak. Kada se pozove, funkcija prvo izvršava RpcEmitHit metodu, koja je zadužena za emitiranje vizualnog efekta koji prikazuje da je neprijatelj pogođen. Zatim se smanjuje vrijednost „EnemyHealth“ varijable za iznos „BulletDamage“, što predstavlja gubitak zdravlja neprijatelja uslijed udara metka.

Ako nakon smanjenja zdravlja neprijateljev „EnemyHealth“ padne na nulu ili ispod nule, aktivira se sljedeći dio funkcije. Prvo se poziva metoda „RpcActivateRagdoll“ koja, pomoću parametara „bulletForce“ i „hitpoint“, aktivira ragdoll efekt na neprijatelju. Ovaj efekt omogućuje realistično padanje i fizičko reagiranje tijela neprijatelja nakon smrti. Nakon toga, funkcija poziva „IncreaseScore“ metodu iz ScoreManager instance, povećavajući ukupni broj bodova koje tim dobiva za eliminaciju neprijatelja, koristeći vrijednost „ScoreToGive“ koja označava koliko bodova taj neprijatelj vrijedi.

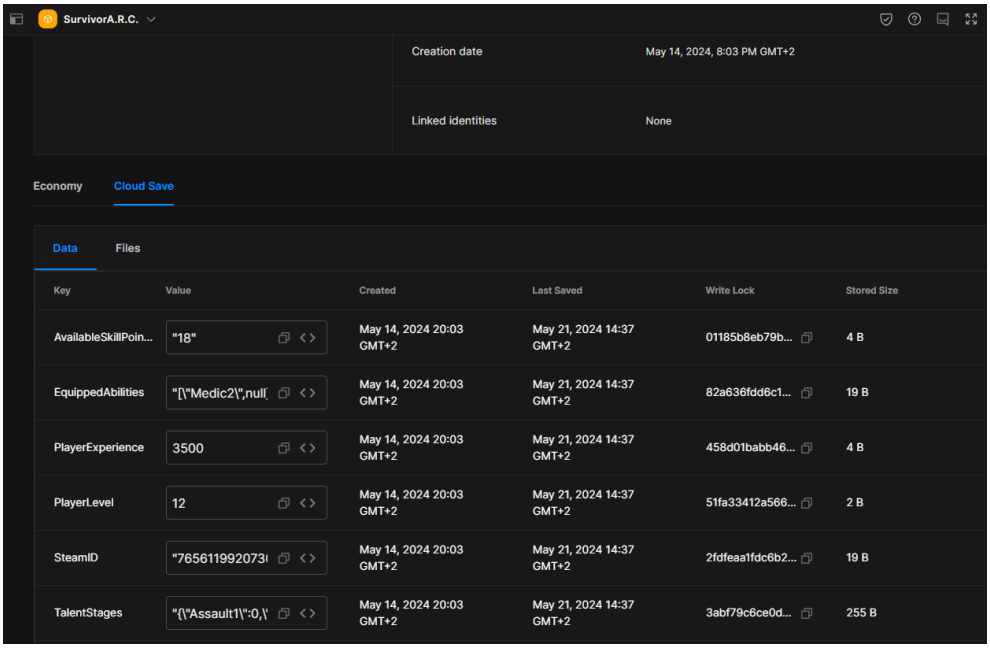
4.5. Sinkronizacija podataka

Sinkronizacija podataka ključna je za osiguranje konzistentnog i ujednačenog iskustva za sve igrače u mrežnoj igri. U ovom projektu sustav sinkronizacije podatka odrađen je uz pomoć dva glavna sistema, mirror netoworkinga za sinkronizaciju unutar igre i Unity Clouda za pohranu i dohvaćanje trajnih podataka korištenjem Unity Clouda [14]. Pomoću Mirror networkinga, podaci o igračima, njihovi pokreti, statusi i druge bitne informacije mogu se lako sinkronizirati između svih sudionika u igri. Korištenje komponenti kao što su NetworkIdentity i NetworkTransform omogućuje automatsku sinkronizaciju položaja, rotacije i drugih atributa objekata između servera i klijenata. Komande (Commands) i daljinski procedure calls (RPCs) koriste se za komunikaciju između klijenata i servera, osiguravajući da se sve promjene podataka dosljedno provode na svim uređajima. SyncVars i SyncLists omogućuju automatsku sinkronizaciju varijabli i lista, čime se pojednostavljuje razvoj mrežnog koda i osigurava dosljednost podataka.

Uz Mirror networking za sinkronizaciju podataka tijekom igre, korišten je Unity Cloud za pohranu i dohvaćanje trajnih podataka igrača kao što su nivo, talenti i iskustvo. Unity Cloud omogućava pohranjivanje podataka igrača te njihov dohvat u realnom vremenu. Korištenjem Unity Player Management sustava, moguće je pohraniti informacije vezane za igrače, te ih dohvaćati kad god je to potrebno (slika 41). Autentifikacija osigurava da se svaki igrač može prijaviti i da su njihovi podaci sigurni, dok sustav za pohranu podataka (Cloud Save) omogućuje pohranjivanje i dohvaćanje podataka igrača na Unity Cloud. Na ovaj način, podaci igrača su uvijek dostupni i mogu se brzo dohvatiti i ažurirati.



Slika 41 Prikaz spremljenih igrača unutar Unity Clouda.



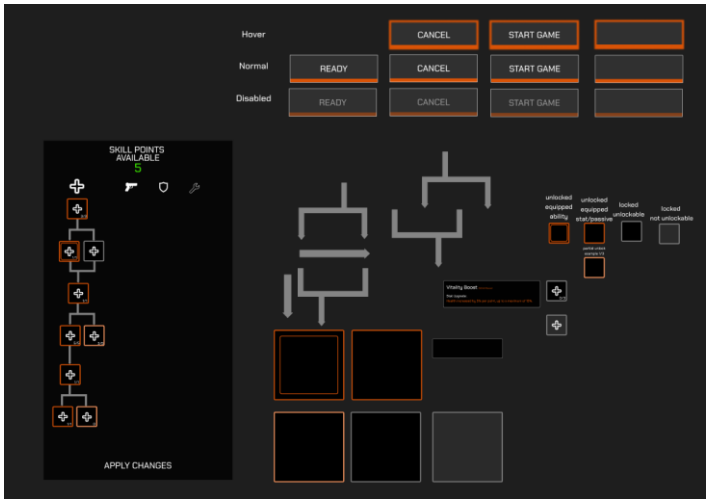
Slika 42 Prikaz spremljenih podataka za specifičnog igrača.

Kao što je vidljivo na slici 42, podaci igrača se pohranjuju u bazi putem Unity Cloud Save funkcionalnosti. U bazi se pohranjuju različiti ključevi i njihove vrijednosti, uključujući "AvailableSkillPoints", "EquippedAbilities", "PlayerExperience", "PlayerLevel", "SteamID" i "TalentStages". Ovi podaci pružaju specifične informacije o igraču, kao što su dostupni bodovi vještina, opremljene sposobnosti, iskustvo, razina igrača, SteamID te stupnjevi talenata. Svaki podatak ima zapis o vremenu kreiranja i zadnjem spremanju, što omogućuje precizno praćenje napretka igrača.

4.6. Korisničko sučelje

Korisničko sučelje (UI) u video igrama predstavlja most između igrača i igre, omogućujući interakciju i pružanje informacija na intuitivan i pristupačan način [15]. Dizajn UI-a uključuje stvaranje elemenata kao što su meniji, indikatori zdravlja, kartice s inventarom, mape i druge vizualne komponente koje pomažu igračima da se kreću kroz igru i razumiju njezine mehanike.

U slučaju igre "Survivor A.R.C.", dizajn korisničkog sučelja odrađen je korištenjem platforme Figma. Figma je moćan alat za dizajn i prototipiranje koji omogućuje suradnju u stvarnom vremenu. Kroz Figmu, odabran je željeni font, paleta boja i ostali vizualni elementi sučelja. Font je odabran tako da bude čitljiv i prilagođen apokaliptičnom tonu igre, dok je paleta boja usklađena s općom atmosferom igre, koristeći tamnije tonove koji odražavaju tmurnu i napuštenu tematiku. Korištenjem Figma platforme omogućena je učinkovita izrada i testiranje UI elemenata, osiguravajući da konačni dizajn bude funkcionalan, intuitivan i estetski usklađen s ostatkom igre, nakon izrade komponenti poput obruba tipki vidljih na slici 43 one se izvoze u obliku slika koje se onda mogu koristi unutar igre.



Slika 43 Prikaz komponenti na figma platformi.

Nakon što su dizajnirani UI elementi u Figma platformi za igru, sljedeći korak je njihova implementacija u Unity okruženju. U Unityju, dizajnirani elementi iz Figme mogu se uvesti kao slike u različitim formatima poput „PNG“. Zatim kako bi prikazali izrađeni dizajn potrebno je kreirati UI elemente odnosno objekte s UI komponentama poput „Button“ ili „Image“. Kada smo kreirali željene UI elemente te ih postavili na željene pozicije potrebno je povezati komponente s željenim funkcijama. Unutar inspektora unity pruža mogućnost dodavanja jednostavnih funkcionalnosti poput manipulacije s objektima, no ukoliko želimo dodavati kompleksnije funkcionalnosti potrebno je izraditi nove skripte u kojima možemo izraditi personalizrane funkcije te ih onda povezati na UI elemente poput „Button“. Ovaj proces omogućuje integraciju dizajniranih UI elemenata iz Figme u Unity okruženje te njihovo daljnje proširenje i prilagodbu kroz izradu personaliziranih skripti i funkcija.

4.7. Optimizacija i performanse

Optimizacija i performanse ključni su aspekti u razvoju igara, posebno kada je riječ o multiplayer igrama. U industriji igara, optimizacija se odnosi na proces poboljšanja performansi igre kako bi se postigla bolja igrivost, smanjila potrošnja resursa i osigurala stabilnost aplikacije. Performanse se mjere kroz različite metrike, uključujući FPS (frames per second), vrijeme učitavanja, latenciju mreže i potrošnju memorije. Pravilno optimizirana igra može pružiti glatko i ugodno iskustvo za igrače, dok loša optimizacija može rezultirati problemima poput trzanja, zaostajanja i rušenja igre [16]. U igri "Survivor A.R.C.", koja je wave shooter top-down co-op igra s low poly stilom, optimizacija je provedena kroz nekoliko ključnih tehnika i alata.

Jedna od glavnih tehnika korištenih u ovoj igri za optimizaciju je object pooling. Object pooling je metoda koja smanjuje troškove instanciranja i uništavanja objekata tijekom igre. Umjesto da se novi objekti kreiraju i uništavaju svaki put kada su potrebni, objektni bazeni unaprijed stvaraju skup objekata koji se ponovno koriste kada su potrebni, ukoliko objekti više ne budu potrebni čekaju na raspolaganju do promjene scene gdje se onda unište. U "Survivor A.R.C." ova tehnika je korištena za neprijatelje (zombije) koji se pojavljuju u valovima. Time se značajno smanjuje opterećenje na garbage collector i poboljšavaju performanse igre.

Kroz razvoj igre korišten je Unity Profiler za praćenje performansi i identifikaciju uskih grla. Profiler je omogućio detaljan uvid u korištenje CPU-a, GPU-a, memorije i mrežnih resursa, što je bilo ključno za optimizaciju. Na primjer, identificirana su područja u kodu gdje su resursno intenzivne operacije mogle biti optimizirane ili raspoređene na drugačiji način. Također je omogućeno praćenje performansi tijekom mrežnog igranja, što je bilo ključno za glatko multiplayer iskustvo. S obzirom na to da "Survivor A.R.C." koristi low poly stil grafike, većina grafičkih resursa već je bila relativno optimizirana. Međutim, dodatni koraci su poduzeti kako bi se osiguralo da igra radi glatko na različitim uređajima. Korišteni su LOD (Level of Detail) modeli kako bi se smanjila složenost objekata, a korištene su i teksture niske rezolucije kako bi se smanjila potrošnja memorije. Multiplayer aspekt igre zahtijevao je dodatnu pažnju pri optimizaciji mrežnih postavki. Korištenjem Mirror networking sustava u Unityju, optimizirane su postavke NetworkTransform komponente. Uvedene su promjene kako bi se smanjila količina podataka koja se šalje preko mreže, uključujući smanjenje frekvencije ažuriranja pozicija i rotacija igrača i neprijatelja.

5. Zaključak

Ovaj diplomski rad detaljno istražuje proces izrade multiplayer igre "Survivor A.R.C." koristeći Unity i Mirror Networking. Kroz pregled teoretskih osnova, tehničkih detalja i praktične implementacije, rad prikazuje kako se suvremene tehnologije i alati mogu koristiti za razvoj složenih računalnih igara.

Razvoj "Survivor A.R.C." započeo je s željom za izradom računalne igre koja se može igrati s više ljudi. Razmatrani su različiti game enginei, koncepti umrežavanja i specifična rješenja za umrežavanje u Unityju. Izbor Unity game enginea i Mirror Networkinga pokazao se kao dobro rješenje za razvoj igre "Survivor A.R.C.". Unity je pružio robustan i fleksibilan temelj za izradu igre, dok je Mirror Networking omogućio efikasnu i stabilnu mrežnu komunikaciju među igračima. Ovi alati omogućili su stvaranje koherentnog i dinamičnog gameplay iskustva. Dizajn igre bio je ključan aspekt u "Survivor A.R.C." igri, koja je zamišljena kao wave shooter top-down co-op igra, gdje igrači surađuju kako bi preživjeli napade valova zombija. Implementacija igre obuhvatila je postavljanje mrežne infrastrukture, razvoj gameplay komponenti poput player controllera, neprijatelja i njihovog AI, te sustava bodovanja i napredovanja. U ovom radu glavni cilj bio je izraditi igru koja podržava igranje preko mreže gdje igrači mogu zajedno surađivati i zabavljati se. Zbog kompleksnosti izrade igre preko mreže potrošeno je jako puno vremena sakupljajući nova znanja i vještine. Igre ovog tipa najčešće se izrađuju u većim timovima zbog samog opseg znanja i količine vremena koji su potrebni za završetak ovakvog projekta. Iz razloga što je multiplayer bio glavni cilj ovog projekta, kroz cijeli rad naglašena je važnost kontinuirane sinkronizacije svih elemenata od samih likova do podataka koji se koriste. Bilo je potrebno sinkronizirati sve elemente igre, od UI-a koji svaki igrač vidi do pokreta i borbe kako bi se postiglo pravo multiplayer iskustvo.

U zaključku, ovaj diplomski rad je pokazao potrebne korake za izradu online igre , te kao takav ističe da je u današnje vrijeme uz malo truda moguće izraditi igre koje bi u prošlosti samo velike kompanije bile u mogućnosti kreirati. Kroz pažljivo planiranje, implementaciju i optimizaciju, stvorena je igra koja pruža zabavno multiplayer iskustvo.

Popis slika

Literatura

[1] S. L. &. M. P. Kent, The ultimate history of video games: From Pong to Pokémon and beyond the story behind the craze that touched our lives and changed the world (1st ed)., Three rivers press., 2001.

[2] O. Wallach, »Visual Capitalist,« 23 Studeni 2020. [Mrežno]. Available: https://www.visualcapitalist.com/50-years-gaming-history-revenue-stream/. [Pokušaj pristupa 6 Lipanj 2024].

[3] »Xsolla,« 10 Svibanj 2022. [Mrežno]. Available: https://xsolla.com/blog/which-game-engine-is-best-for-you. [Pokušaj pristupa 6 Lipanj 2024].

[4] Ably, »Ably,« 11 Listopad 2023. [Mrežno]. Available: https://ably.com/blog/building-realtime-multiplayer-games-has-never-been-easier. [Pokušaj pristupa 6 Lipanj 2024].

[5] »Unity,« Unity, [Mrežno]. Available: https://unity.com/solutions/build-backend. [Pokušaj pristupa 8 Lipanj 2024].

[6] B. House, »Unity Blog,« Unity, 8 Rujan 2020. [Mrežno]. Available: https://blog.unity.com/games/how-to-choose-the-right-netcode-for-your-game. [Pokušaj pristupa 8 Lipanj 2024].

[7] D. Ivashchenko, »Hackernoon,« 31 Kolovoz 293. [Mrežno]. Available: https://hackernoon.com/unity-realtime-multiplayer-part-8-exploring-ready-made-networking-solutions. [Pokušaj pristupa 8 Lipanj 2024].

[8] »Mirror Networking,« [Mrežno]. Available: https://mirror-networking.gitbook.io/docs/manual/general. [Pokušaj pristupa 8 Lipanj 2024].

[9] N. Stefyn, »Cgspectrum,« 13 Listopad 2019. [Mrežno]. Available: https://www.cgspectrum.com/blog/what-is-game-design. [Pokušaj pristupa 8 Lipanj 2024].

[10] »Valve Developer Community,« Valve, [Mrežno]. Available: https://developer.valvesoftware.com/wiki/Steamworks#:~:text=Steamworks%20is%20an%20API%20intended,together%20with%20a%20unified%20system.. [Pokušaj pristupa 8 Lipanj 2024].

[11] »Unity Learn,« Unity Technologies, 15 Rujan 2020. [Mrežno]. Available: https://learn.unity.com/tutorial/the-package-manager#. [Pokušaj pristupa 8 Lipanj 2024].

[12] »Unity Documentation,« Unity, [Mrežno]. Available: https://docs.unity3d.com/Manual/Components.html. [Pokušaj pristupa 8 Lipanj 2024].

[13] »Unity Manual,« Unity, 12 Srpanj 2017. [Mrežno]. Available: https://docs.unity3d.com/560/Documentation/Manual/class-NavMeshAgent.html. [Pokušaj pristupa 8 Lipanj 2024].

[14] A. Sergeev, »Bolu,« 29 Siječanj 2024. [Mrežno]. Available: https://80.lv/articles/here-s-how-unity-cloud-helps-in-solving-game-development-tasks/. [Pokušaj pristupa 8 Lipanj 2024].

[15] M. Bowers, »Toptal designers,« Toptal, [Mrežno]. Available: https://www.toptal.com/designers/gui/game-ui#:~:text=What%20is%20a%20game%20user,Diegetic%2C%20Meta%2C%20and%20Spatial.. [Pokušaj pristupa 8 Lipanj 2024].

[16] M. Hergaarden, »Game Developer,« 6 Srpanj 2021. [Mrežno]. Available: https://www.gamedeveloper.com/programming/optimizing-performance-of-unity-games-a-battle-tested-plan. [Pokušaj pristupa 8 Lipanj 2024].

[17] »Mirror-Networking,« 2021. [Mrežno]. Available: https://mirror-networking.gitbook.io/docs/manual/transports/fizzysteamworks-transport. [Pokušaj pristupa 6 Lipanj 2024].

[18] »BleedingEdge,« 25 Srpanj 2923. [Mrežno]. Available: https://bleedingedge.studio/blog/mastering-graphics-api-in-unity-in-2023/. [Pokušaj pristupa 6 Lipanj 2024].

[19] A. Galavan, »Alain,« 31 Siječanj 2021. [Mrežno]. Available: https://alain.xyz/blog/comparison-of-modern-graphics-apis. [Pokušaj pristupa 6 Lipanj 2024].

[20] »UnrealEngine,« [Mrežno]. Available: https://www.unrealengine.com/en-US/unreal-engine-5. [Pokušaj pristupa 7 Lipanj 2024].

[21] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Godot_%28game_engine%29. [Pokušaj pristupa 6 Lipanj 2024].

[22] »Steam Charts,« [Mrežno]. Available: https://steamcharts.com/. [Pokušaj pristupa 6 Lipanj 2024].

[23] G. Gambetta, »GabrielGambetta,« [Mrežno]. Available: https://gabrielgambetta.com/client-side-prediction-server-reconciliation.html. [Pokušaj pristupa 7 Lipanj 2024].

[24] G. Gambetta, »GabrielGambetta,« [Mrežno]. Available: https://www.gabrielgambetta.com/lag-compensation.html. [Pokušaj pristupa 6 Lipanj 2024].

[25] A. Ellis, »Wired,« Wired, 30 Siječanj 2022. [Mrežno]. Available: https://www.wired.com/story/kernel-anti-cheat-online-gaming-vulnerabilities/. [Pokušaj pristupa 7 Lipanj 2024].

[26] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Game_engine. [Pokušaj pristupa 6 Lipanj 2024].

[27] R. Bramble, »GameMaker,« GameMaker, 10 Svibanj 2023. [Mrežno]. Available: https://gamemaker.io/en/blog/stages-of-game-development. [Pokušaj pristupa 7 Lipanj 2024].