

**SVEUČILIŠTE U ZAGREBU**  
**FAKULTET ORGANIZACIJE I INFORMATIKE**  
**V A R A Ž D I N**

**Josip Petanjek**

**Izrada akcijske igre iz prvog lica u Unreal  
Engine**

**DIPLOMSKI RAD**

**Varaždin, 2021.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Josip Petanjek**

**Matični broj: 0016124756**

**Studij: Informacijsko i programsко inženjerstvo**

**Izrada akcijske igre iz prvog lica u Unreal Engine**

**DIPLOMSKI RAD**

**Mentor/Mentorica:**

**Prof. dr. sc. Radošević Danijel**

**Varaždin, lipanj 2021.**

*Josip Petanjek*

**Izjava o izvornosti**

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

*Autor/Autorica potvrđio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi*

---

## **Sažetak**

Izrada te implementacija značajki ključnih za funkciju mrežnih akcijskih igara iz prvog lica. Implementirat će se umjetna inteligencija, projektili i penetracija, objekti koje se mogu izgraditi ili uništiti, vozila te će se uspostaviti pravila igre. Umjetna inteligencija podrazumijeva likove kojima igrač može davati naredbe ili preuzeti nad njima kontrolu. Projektili i penetracija podrazumijeva implementaciju balističkog sustava te simulacija penetracije objekata. Objekti koji se mogu izgraditi ili uništiti podrazumijeva objekte poput zaklona koji se mogu izgraditi od strane likova, te uništiti pomoću projektila. Implementacija vozila podrazumijeva upravljanje istima. Uspostavljanje pravila igre podrazumijeva uvjete pobjede, odnosno implementacija timova igrača te sustava bodovanja.

**Ključne riječi:** Unreal Engine, Spatial OS, Igra u prvom licu, Umreženo igranje, Umrežena simulacija fizike, Vozila, Uništivi objekti

# Sadržaj

Sadržaj .....	iii
1. Uvod .....	1
2. Metode i tehnike rada .....	2
2.1. Visual Studio 2019 .....	2
2.2. Unreal Engine .....	2
2.3. Spatial OS GDK za Unreal Engine 4 .....	2
2.4. Sourcetree i repozitorij .....	6
2.5. Adobe XD .....	6
3. Razrada teme .....	7
3.1. Igre kao medij .....	7
3.2. Igre sljedeće generacije .....	8
3.2.1. Trenutno stanje umreženih igara .....	8
3.2.2. Tehnička ograničenja trenutnih igara .....	10
3.2.2.1. Fortnite .....	10
3.2.2.2. Battlefield 4 .....	11
3.2.2.3. DayZ Standalone .....	13
3.2.2.4. Planetside 2 .....	13
3.2.3. Specifikacije igara sljedeće generacije .....	14
3.3. Umrežena simulacija fizike i stanje svijeta .....	15
3.3.1. Tehnike umreženja .....	16
3.3.1.1. Deterministički diskretni korak – slanje unosa .....	16
3.3.1.2. Interpolacija slike stanja – slanje pozicije .....	20
3.3.1.3. Sinkronizacija stanja .....	22
3.4. Više-serverska arhitektura .....	26
3.4.1. Spatial OS .....	27
3.4.1.1. Temeljni pojmovi .....	27

3.4.1.2. Distribucija simulacije i povezani koncepti .....	28
3.4.1.3. Ograničenja i maksimalni broj igrača .....	29
3.4.1.4. Dodatna rješenja za distribuirane svjetove .....	30
3.4.1.5. Inspektor svijeta .....	30
3.4.1.6. Istoimeni pojmovi u UE .....	32
3.4.1.7. Kako iterirati u Spatial OS .....	32
3.4.2. Upravljanje interesom klijenta igre .....	33
3.4.2.1. Umreženo izbacivanje po kvadratnoj udaljenosti .....	33
3.4.2.2. Umreženo izbacivanje po udaljenosti sa modifikacijom frekvencije.....	34
3.4.2.3. Entiteti nad kojima uvijek imamo interes .....	35
3.4.2.4. Komponenta interesa za lika .....	36
3.4.3. Dinamično upravljanje interesom klijenta igre .....	39
3.4.3.1. Istraživanje umreženog drivera.....	39
3.4.3.2. Interes na bazi vidljivog polja klijenta.....	41
3.4.3.3. Buduće iteracije.....	43
3.4.4. Uspostavljanje sloja za umjetnu inteligenciju .....	46
3.4.4.1. Stvaranje umjetne inteligencije .....	48
3.4.4.2. Preuzimanje kontrole nad umjetnom inteligencijom .....	50
3.4.4.3. Buduće iteracije AI .....	53
3.5. Umrežene značajke .....	54
3.5.1. Vozila .....	54
3.5.1.1. Prva iteracija – Replikacija kretnje.....	54
3.5.1.2. Druga iteracija - INetworkPredictionInterface.....	59
3.5.1.3. Treća iteracija – Ručno izrađena fizika .....	61
3.5.1.4. Četvrta iteracija – Chaos fizika .....	63
3.5.2. Umreženje simulacije fizike objekata.....	66
3.5.2.1. Algoritam .....	66
3.5.2.2. Prva iteracija – kucanje srca.....	66
3.5.2.3. Druga iteracija – kanal za buđenje.....	67

3.5.2.4. Treća iteracija – paljenje i gašenje simulacije .....	67
3.5.2.5. Finalna implementacija – razlika u poziciji .....	68
3.5.2.6. Buduće iteracije – zaključavanje Z osi .....	69
3.5.3. Deformacija terena .....	70
3.5.3.1. Tehnologija i algoritam .....	70
3.5.3.2. Prva iteracija – replikacija unosa .....	72
3.5.3.3. Finalna implementacija – serijalizirano polje vektora .....	73
3.5.3.4. Buduće iteracije .....	76
3.5.4. Izgradnja objekata .....	77
3.5.4.1. Prva iteracija – statički objekti .....	77
3.5.4.2. Buduće iteracije – fizički objekti .....	85
3.5.5. Uništivi objekti .....	86
3.5.5.1. Prva iteracija – PhysX APEX + DENT .....	86
3.5.5.2. Trenutna generacija – PhysX Blast .....	92
3.5.5.3. Buduća iteracija – Chaos destrukcija .....	92
3.5.6. Simulacija projektila i balistike .....	94
3.5.6.1. Projektili .....	94
3.5.6.2. Balistika .....	95
3.6. Funkcionalnost igre i meča .....	102
3.6.1. Dizajn funkcionalnosti igre i meča .....	102
3.6.1.1. Uvjet pobjede .....	103
3.6.1.2. Udruživanje kroz lanac zapovijedanja .....	103
3.6.1.3. Motiviranje sustavom bodovanja .....	105
3.6.1.4. Logistički sustav .....	107
3.6.2. Implementacija funkcionalnosti igre i meča .....	108
3.6.2.1. Kontrolne točke .....	109
3.6.2.2. Uvjeti pobjede .....	110
3.6.2.3. Testiranje servera .....	112
4. Zaključak .....	114

Popis literature .....	115
Popis slika.....	126
Popis tablica.....	130

# 1. Uvod

Umrežene igre su jedinstven medij koji dopušta interakciju između više ljudi, ovaj rad se bavi razvojem umrežene igre sljedeće generacije, sa stotinama igrača u visoko detaljnem svijetu.

Ova tema je značajna jer su igre trenutne generacije ograničene svojim tehnologijama, te zato često rade kompromise u detaljnosti svijeta i broju igrača, što ograničava broj interakcija i zabavu.

Osobno smatram da je ovo ključno, i veoma zapostavljeno područje u trenutnoj generaciji, te da će tehnologije obrađivane u ovom radu dopustiti istinito uvjerljive umrežene svjetove sa manje ograničenja i kompromisa, te najvažnije, sa više zabave.

## **2. Metode i tehnike rada**

### **2.1. Visual Studio 2019**

Visual Studio 2019 - za daljnje potrebe poznat kao VS, je integrirano razvojno okruženje tvrtke Microsoft [1]. Ono služi kao podloga za razvoj projekta jer se okruženja za razvoj baziraju na C++, kojeg VS jako dobro podržava.

Preuzeto kao dio edukacijskog paketa sa Microsoft Azure.

### **2.2. Unreal Engine**

Unreal Engine - za daljnje potrebe poznat kao UE, je okruženje za razvoj (eng. engine), primarno namijenjeno za razvoj igara. Stvorio ga je Tim Sweeney 1998.g. , osnivač tvrtke Epic Games [2]. U ovom radu koristiti će se Unreal Engine 4, inačica koja je izašla 2014.g [3].

U ovom radu koristiti će se alat zvan nacrti, on služi za programiranje jednostavnijih funkcionalnosti pomoću vizualnog kodiranja [4].

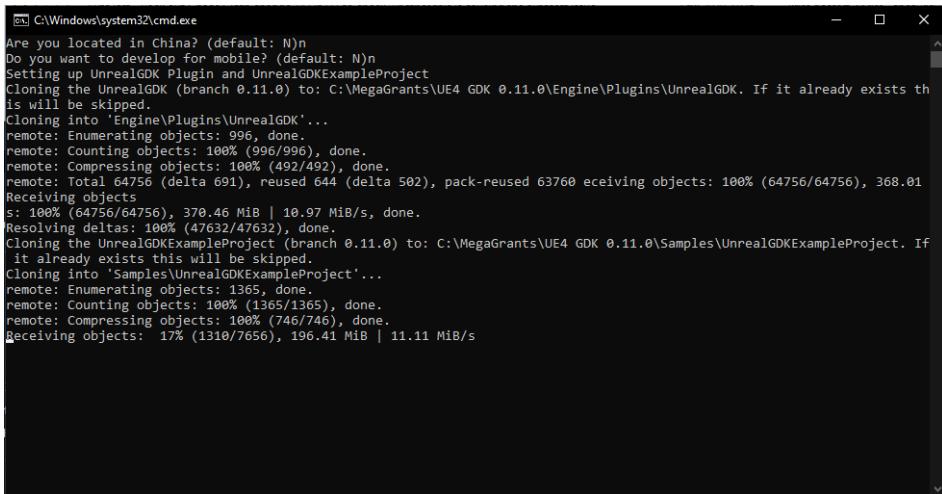
### **2.3. Spatial OS GDK za Unreal Engine 4**

Spatial OS tvrtke Improbable je operacijski sustav za distribuciju računalne obrade [5]. On se integrira u UE4 preko kompleta za razvoj igara (eng. Games Development Kit – GDK) [6].

Da bi smo mogli preuzeti ovaj alat, potrebno je prvo pridružiti se organizaciji Epic Games na platformi GitHub [7].

Proces instalacije počinje preuzimanjem radne grane sa odgovarajućeg repozitorija [8]. U ovom radu koristi se verzija „0.11.0“ iz 2020.g. , ovdje se dobiva i sam Unreal Engine 4.

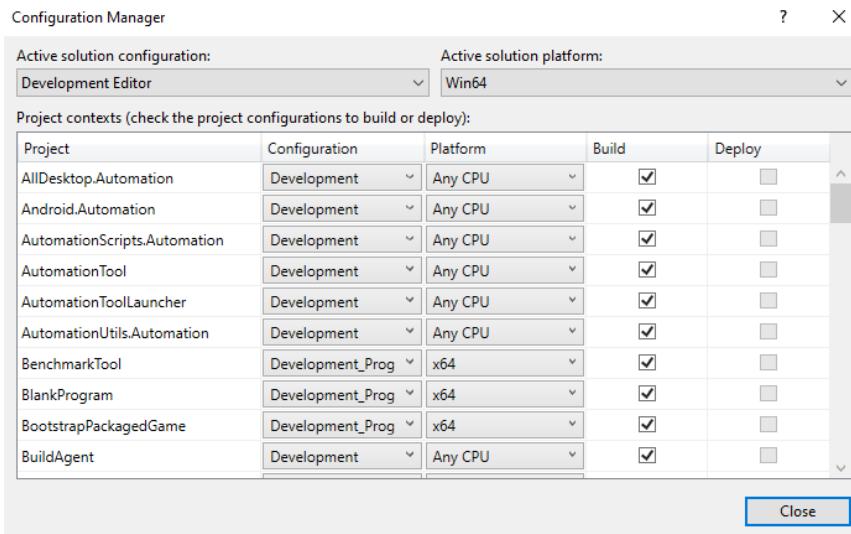
Nakon preuzimanja pokrećemo prvo „Setup.bat“ i „GenerateProjectFiles.bat“ da bi se instalirao UE4, za uspostavljanje kompleta za razvoj igara pokrećemo „InstallGDK.bat“ [9].



```
C:\Windows\system32\cmd.exe
Are you located in China? (default: N)
Do you want to develop for mobile? (default: N)
Setting up UnrealGDK Plugin and UnrealGDKExampleProject
Cloning the UnrealGDK (branch 0.11.0) to: C:\MegaGrants\UE4 GDK 0.11.0\Engine\Plugins\UnrealGDK. If it already exists th
is will be skipped.
Cloning into 'Engine\Plugins\UnrealGDK'...
remote: Enumerating objects: 996, done.
remote: Counting objects: 100% (996/996), done.
remote: Compressing objects: 100% (492/492), done.
remote: Total 64756 (delta 691), reused 644 (delta 502), pack-reused 63760 eceiving objects: 100% (64756/64756), 368.01
Receiving objects
s: 100% (64756/64756), 370.46 MiB | 10.97 MiB/s, done.
Resolving deltas: 100% (47632/47632), done.
Cloning the UnrealGDKExampleProject (branch 0.11.0) to: C:\MegaGrants\UE4 GDK 0.11.0\Samples\UnrealGDKExampleProject. If
it already exists this will be skipped.
Cloning into 'Samples\UnrealGDKExampleProject'...
remote: Enumerating objects: 1365, done.
remote: Counting objects: 100% (1365/1365), done.
remote: Compressing objects: 100% (746/746), done.
Receiving objects
s: 17% (1310/7656), 196.41 MiB | 11.11 MiB/s
```

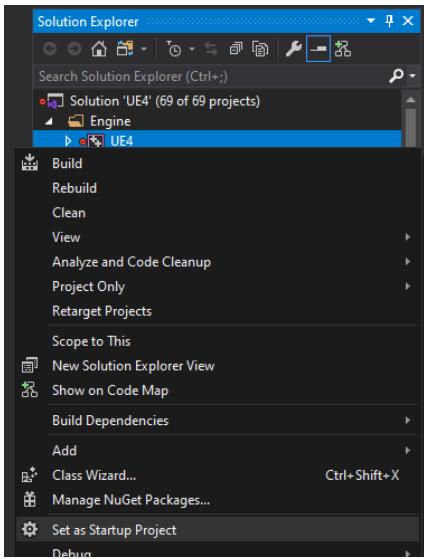
Slika 1 InstallGDK.bat

Sada je potrebno konfigurirati Visual Studio, kako bi se projekt mogao dobro izgraditi.



Slika 2 Configuration Manager Visual Studio 2019

Nadalje, potrebno je postaviti UE4 kao početni projekt, kako bi se pravilno izgradio.



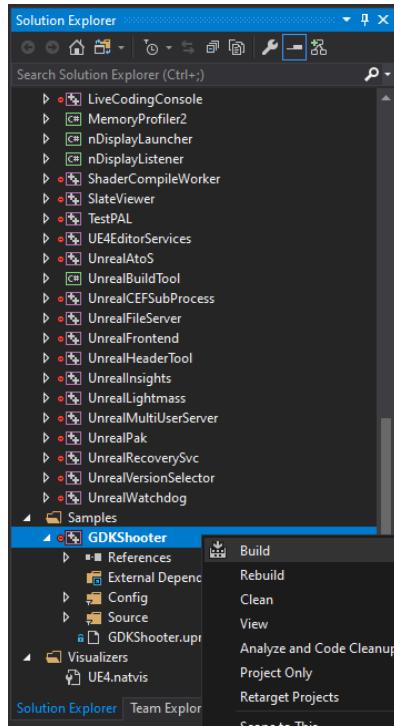
Slika 3 Set as Startup Project

Sada možemo izgraditi okruženje za razvoj.

```
1> [4222/4223] UE4Editor-UnrealEd.dll
1>     Creating library C:\MegaGrants\UE4 GDK 0.11.0\Engine\Intermediate\Build\Win64\UE4Editor.lib
1> [4223/4223] UE4Editor.target
1>Total time in Parallel executor: 3673,83 seconds
1>Total execution time: 3815,59 seconds
1>Done building project "UE4.vcxproj".
===== Build: 1 succeeded, 0 failed, 2 up-to-date, 0 skipped =====
```

Slika 4 Build engine 1 sat

Naposljeku, moramo izgraditi sam projekt igre, kako bi ju mogli razvijati u okruženju.

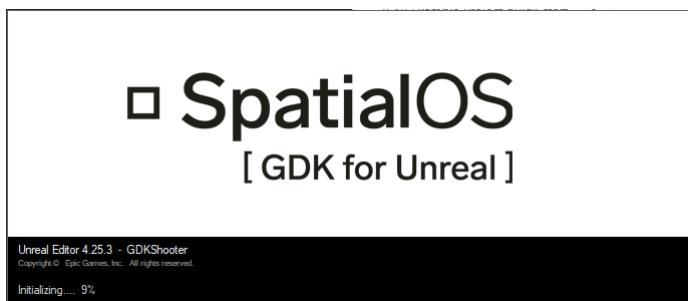


Slika 5 Build projekta

```
1> [7/7] GDKShooterEditor.target
1>Total time in Parallel executor: 18,06 seconds
1>Total execution time: 32,77 seconds
===== Build: 1 succeeded, 0 failed, 2 up-to-date, 0 skipped =====
```

Slika 6 Build projekta

Ovo nam dopušta da pokrenemo UE4 verziju sa ugrađenim Spatial OS alatom.

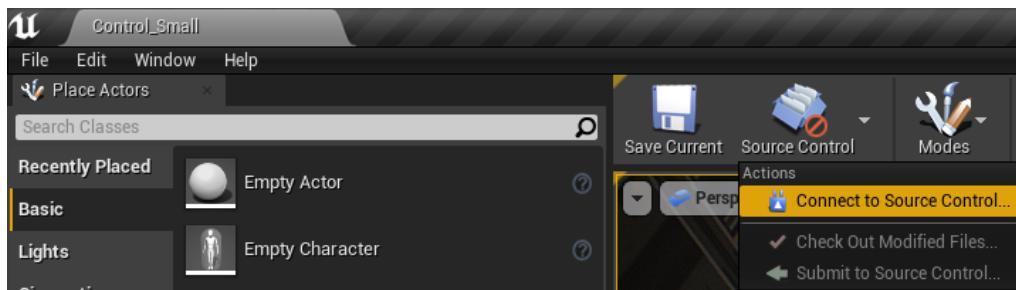


Slika 7 Pokretanje projekta

## 2.4. Sourcetree i repozitorij

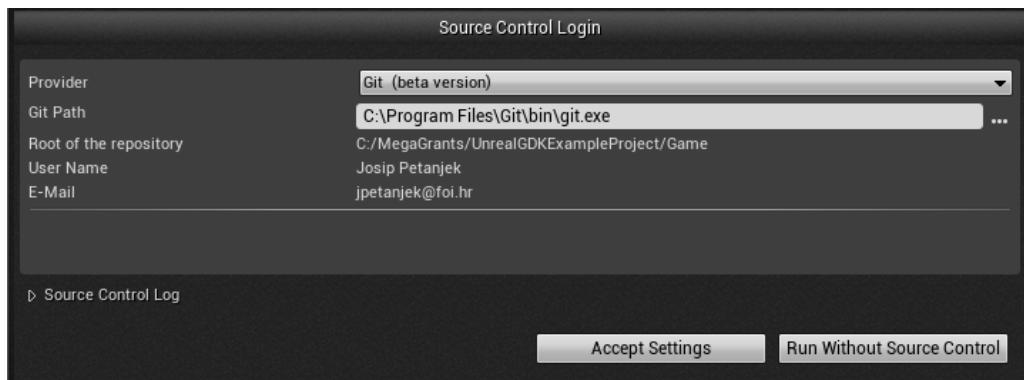
Sourcetree [10] se koristi za održavanje repozitorija projekta [11].

Sam sustav kontrole izvora može se povezati sa UE4.



Slika 8 Povezivanje UE4 sa sustavom kontrole izvora

Za njega je potrebna lokalna instalacija programa Git [12], te pravilna konfiguracija.



Slika 9 Konfiguracija sustava kontrole izvora

## 2.5. Adobe XD

Adobe XD je vektorski alat [13] koji će se u ovom radu koristiti za izradu koncepata raznih sučelja igre. Koristi se besplatna verzija.

### **3. Razrada teme**

Da bi smo postavili podlogu za izradu ovog rada, prvo ćemo objasniti igre kao medij te jedinstvenost umrežene interakcije. Da bi smo mogli specificirati naše potrebe za sljedeću generaciju, proučiti ćemo trenutnu generaciju i njezina tehnološka ograničenja. Nadalje, kako bi mogli obraditi temu, potrebno je obraditi problem umreženja simulacije svjetova. Naposljeku, izraditi ćemo igru sljedeće generacije, korištenjem više-serverske arhitekture, uz umrežene značajke, te ćemo dizajnirati funkcionalnost igre i meča.

#### **3.1. Igre kao medij**

Chris Crawford, utemeljitelj GDC-a, često zvan Sokratesom igara, u svom „Zmajevskom“ govoru priča o temeljnoj ideji igara [14]. On ideje o igrama smatra kao sне, nejasne, mutne slike, koje se sastoje samo od obrisa rubova, te ove ideje predstavlja u metaforama kao sне o zmaju.

Igre smatra jedinstvenim medijem zbog svoje interaktivnosti, te smatra da one temeljno mogu promijeniti načine masovnog komuniciranja, umjesto da smo statični promatrač koji sluša, kao u klasičnim medijima, igre nam dopuštaju interakciju i dinamičnost.

Dakle, dolazimo do temeljnog principa je prednost ovog medija: interakcija.

Da bi se interaktivnost mogla postići, Crawford zaziva nekoliko potrebnih tehnoloških napredaka, koji se zapravo mogu ukuhati na sljedeće koncepte: vizualni i auditivni paritet sa pravim svijetom ( on to predstavlja kao 3D ekspresije lica ) te simulirano ljudsko ponašanje ostalih likova ( za njega je to napredni AI ).

Crawford smatra da igre trebaju biti o ljudima, ne stvarima, da igre treba zamišljati u detaljima, te da je potrebno u potpunosti vizualizirati zamišljeni dizajn.

Pošto je simuliranje ljudskog ponašanja ( u potpuno dinamičkom smislu, gdje simulirani čovjek može reagirati na svaku moguću situaciju ) iznimno teško, ako ne i nemoguće, ne možemo se oslanjati na njega kao podlogu za istinito interaktivni medij. Jedino očito rješenje za ovaj problem je da ne simuliramo ljudsko ponašanje, nego da umrežimo ljude tako da zajedno mogu biti u interakciji. Ovo je temeljna ideja iza umreženog igranja, te se zato ovaj rad na isto fokusira.

Crawford zaziva da je igra komunikacija između njena stvaratelja i igrača, ovaj koncept se u umreženim igrama stvaratelj postavlja okruženje u kojem će igrači međusobno komunicirati, odnosno stvaratelj postavlja pravila svijeta i interakcija. Ograničenje broja mogućih interakcija mijenja se iz ograničenja postavljenog algoritma, na praktički neograničeni broj ideja ljudskog bića. A jedan od najvažnijih faktora zabave umreženim igrama je broj interakcija [15].

Interakcija tako igra važnu ulogu u umreženim igrama, logički slijedi da će okružje sa više igrača imati više interakcija. Zbog toga je bitno da se broj igrača u umreženoj igri proširi do tehnološke granice.

Princip realizma je vrlo blizak interakciji, može se promatrati kao preciznost ili detaljnost simulacije, igrači su navikli na pravi svijet, i očekuju da se igra ponaša kao što i pravi svijet (vizualno, auditivno itd.).

## 3.2. Igre sljedeće generacije

Trendovi u industriji pokazuju da su umrežene igre dominantne na tržištu [16], jer, kao što smo prije naveli, zbog nepredvidljivosti ljudskog ponašanja i broja mogućih interakcija one daju izazov koji je vrlo zabavan za igrače. U ovom poglavlju proučiti ćemo ograničenja trenutne generacije, te specificirati sljedeću generaciju.

### 3.2.1. Trenutno stanje umreženih igara

Trenutno stanje umreženih igara je vrlo uzbudljivo te se može podijeliti u dva dijela, igre koje se temelje na mečevima, te igre koje se temelje na trajnim svjetovima. Kada govorimo o mečevima, misli se na igre koje se provode u zasebnim instancama te nemaju utjecaj na kontinuitet svijeta igre, odnosno jedna runda ne utječe na drugu, jedan od primjera je Call of Duty Warzone, gdje jedan meč nema direktnih posljedica na sljedeći.

Kada govorimo na igrama temeljenim na trajnim svjetovima (eng. persistent), mislimo na živući svijet kojeg igrači oblikuju, u području igara preživljavanja to su Minecraft, Rust ili DayZ, u području simulacije svijeta to su Star Citizen, te razni MMO-RPG poput Mortal Online 2 itd.

Trenutno dominiraju igre temeljene na mečevima, inspirirane igramama preživljavanja, takozvani „battle royale“ koji je zapravo amalgamacija igara preživljavanja i klasičnih igara temeljenih na mečevima.

### 3.2.2. Tehnička ograničenja trenutnih igara

Trenutni industrijski standard za umrežene igre je raznolik, te ovisi o detaljnosti svijeta, u sljedećih nekoliko odlomaka biti će dan opis nekoliko igara i detaljnosti njihovog okruženja. Generalno, igre koje ćemo promatrati koriste princip slanja pozicijskih ažuriranja za objekte, ovo je relevantno zapamtiti za daljnja poglavljia.

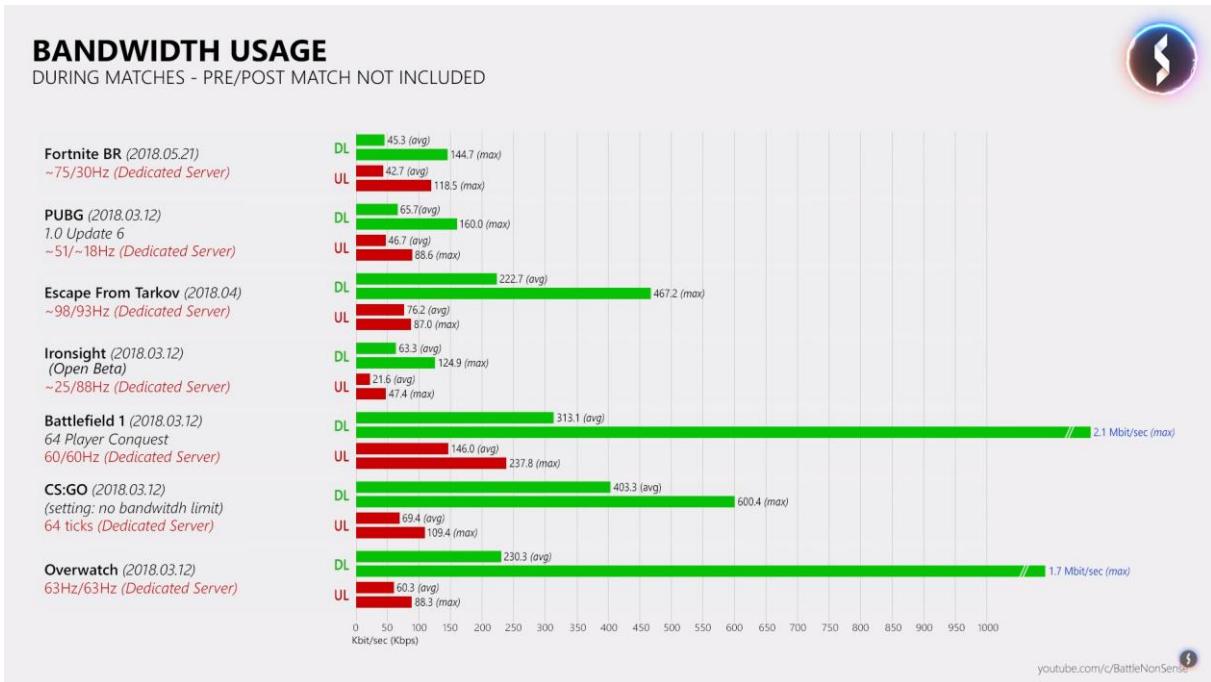


Slika 10 Pregled umreženih specifikacija trenutno popularnih igara [17]

#### 3.2.2.1. Fortnite

Fortnite, izrađen na UE4, sadrži 100 igrača, sa 40-50 tisuća repliciranih likova [18], odnosno interaktivnih elemenata. Ti interaktivni elementi su uništivi objekti (drveća, zidovi), oružja koja se mogu kupiti, umjetna inteligencija i slično.

Svijet se osvježava vrlo brzo, odnosno server klijentima šalje osvježavanje 30 puta u sekundi odnosno 30 Hz, dok klijent serveru šalje svoje podatke na 60 Hz. Osvježavanje može biti varijabilno, dakle na početku runde gdje je svih 100 igrača na jednom mjestu, osvježavanje se dinamički smanjuje do 10 Hz [19]. To rezultira sa prosječnim prometom od 45.3 Kbit/sek za preuzimanje, te 42.7 Kbit/sek za učitavanje [20], što je više nego prihvatljivo za žičnu vezu.



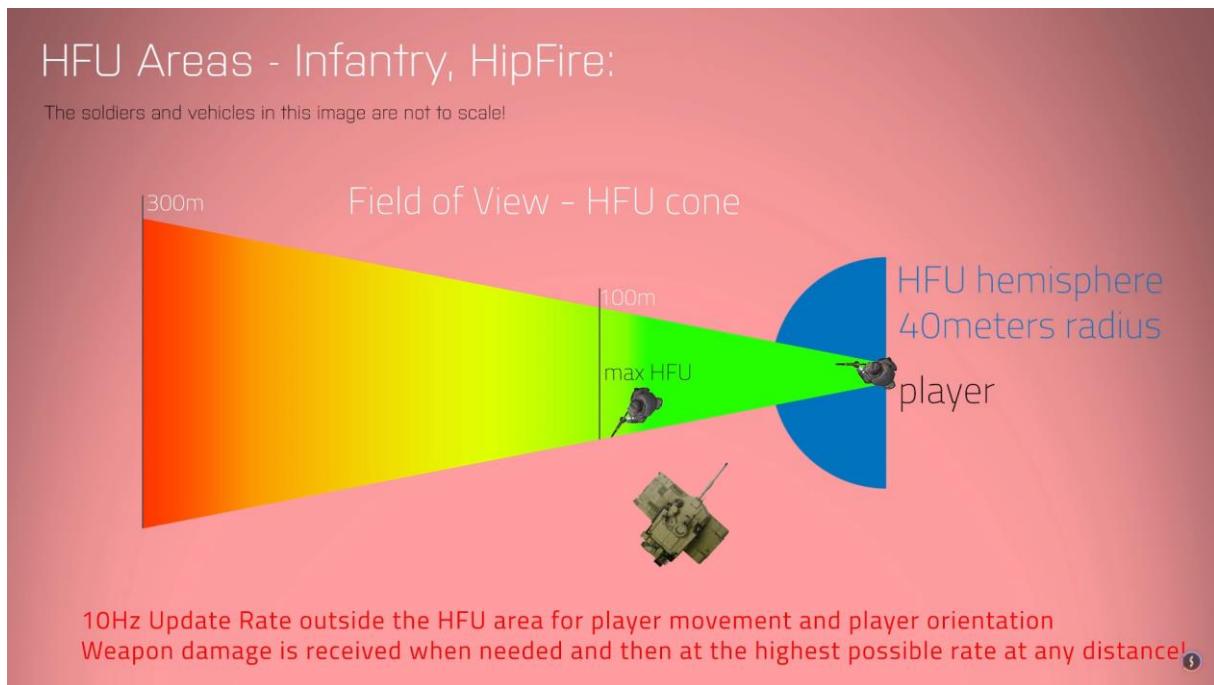
Slika 11 Mrežna opterećenost trenutno popularnih igara [20].

Jedina mana ovog svijeta je manjak fizičkih objekata, to su samo vozila, koja su fizički simulirana, te objekti koji se mogu pokupiti, što daje dojam dosta statičnog svijeta.

Ova igra predstavlja standard trenutne generacije, 100 igrača u interaktivnom svijetu povezanih sa brzim osvježavanjima.

### 3.2.2.2. Battlefield 4

Jedna od naprednijih igara u području tehnologije umreženja je Battlefield 4. Ta igra se sadrži od 64 igrača, u vrlo dinamičnom svijetu, sa uništivim terenom i objektima, fizički baziranim vozilima te projektilima. Koristila je i napredne tehnike poput visoko frekvencijske umrežene sfere, što je unaprijeđena tehnika replikacije bazirane na udaljenosti. Radi se o uzimanju u obzir blizine drugih likova, tako da se bliži likovi brže osvježavaju (ako su u određenoj udaljenosti), te se oni likovi koji su u vidljivom polju klijenta (eng. Field of view) također osvježavaju brže od onih koji su na primjer iza klijenta [21]. Ova polja su izrađena na bazi tri klase, pješak, automobil i avion, zbog njihovih različitih brzina i pogleda. Ova metoda je omogućila skaliranje sustava umreženja do 120 Hz osvježavanja, što drži korak sa kompetitivnim igrami kao Counter Strike Global Offensive.



Slika 12 Napredne tehnike umreženja u Battlefield 4 [22]

Jedine mane ove igre su broj igrača, koji ne prati trenutne trendove od 100+ igrača, ne dinamičnost uništenih objekata (uništivost objekata je zadana – mogu se uništiti na samo jedan način) te manjak umjetne inteligencije.

Ova igra predstavlja jednu od najnaprednijih konfiguracija umreženja, te će služiti kao dobra tehnološki cilj ovog rada.

### **3.2.2.3. DayZ Standalone**

DayZ Standalone je igra koja kroz razvoj služi kao savršen primjer za serverski autoritativnu logiku. Naime ova igra je započela život kao modifikacija na igru „ArmA 2“, te kasnije prešla na samostalnu (eng. Standalone) verziju, i to u okruženju za razvoj „Real Virtuality“ [23].

DayZ je kao igra veoma kažnjavajuća, provodi se u trajnom svijetu, te ako igrač ikada u procesu igranja umre, počinje od početka te gubi sav napredak. To motivira pojedince da traže načine za varanje, a pošto je „Real Virtuality“ kao okružje bilo vrlo klijentski orijentirano, to je bilo vrlo lagano.

Kada kažemo klijentski orijentirano, mislimo na to da se korisniku vjerovalo kada je serveru rekao nešto. Mogao mu je reći da je na jednom kraju svijeta, a u sljedećem trenutku, na potpuno drugom kraju, ili pak da nikad nije primio štetu, da ima bezbroj metaka i slično.

Zbog tih, a i mnogih drugih razloga, prešlo se na serverski autoritativan model te općenito, može se reći, na umreženo orijentirano razvojno okruženje „Enfusion“ [24]. Gdje se je kroz dugu kampanju reinženjeringa samog okruženja, ograničila većina metoda varanja [25]. Sada je DayZ vrlo impresivna umrežena igra, sa stotinu tisuća umreženih objekata, vozilima, puno AI te puno igrača (60-120 ovisno o konfiguraciji servera).

Zaključak koji možemo povući od DayZ je da od temelja trebamo raditi igre sa umreženjem na umu, te sa serverski autoritativnim modelom.

### **3.2.2.4. Planetside 2**

Planetside 2 je nešto starija igra, ali predstavlja konfiguraciju koncentriranu na maksimizaciju broja igrača [26]. To je skup povezanih servera u jedinstveno iskustvo, jedan server u ovom povezanom skupu zove se krhotina (eng. Shard). Na jednom od tih krhotina moguće je imati 2000 igrača, ovo predstavlja trenutnu gornju granicu trenutne generacije i jedno-serverske arhitekture. Treba spomenuti da su za postignuće ove metrike žrtvovane neke serverski autoritativne značajke, te se često dolazi do urušavanja sinkronizacije i zastajanja (eng. Lagg).

### **3.2.3.Specifikacije igara sljedeće generacije**

Dakle očito je su umrežene igre vrlo tehnički zahtjevne, ograničene su većinom zbog procesorskih performansi i trenutnom arhitekturom; te su dizajnirane oko tih ograničenja, često su samo prenamijenjene iz lokalnih igara u umrežene, bez previše obzira na posljedice. Rezultat su vrlo prorijeđene simulacije koje samo daju prividnost realizma, te pružaju ograničeni broj interakcija.

Što nas dovodi do ideje igara sljedeće generacije. Želimo se u potpunosti odvojiti od ograničenja arhitekture, želimo simulacije koje su detaljnije, trajne i bliže pravom svijetu, želimo više igrača u većem i trajnom svijetu. Ukratko želimo igre sa manje kompromisa ograničenjima računala.

Kao specifikaciju za igre sljedeće generacije, u umreženom smislu, koristiti ćemo prosječnu globalnu internetsku brzinu [27], koju procjenujemo na 100 Mbps. Sa serverske strane pretpostavljamo 10Gbps vezu [28]. Nadalje, igra mora biti serverski autoritativna, za eliminaciju mogućnosti varanja, te premašiti trenutni standard broja igrača i detaljnosti okruženja.

Da bi smo postigli više interakcija i veću detaljnost simulacije, moramo se odmaknuti od postavljenih ograničenja. To ćemo ostvariti kroz promjenu arhitekture, kao ekstenzija toga raditi ćemo na umreženoj fizici, te naposljetku osigurati pravila igre za što veću interakciju kroz funkcionalnost igre i meča.

Treba spomenuti da u neku ruku, ova ograničenja su prisutna i zbog limitiranog budžeta lokalnih računala, često konzolama prošle generacije, na primjer serija igara Battlefield poznata je po tome da radi na jako puno generacija, ali je zato i ograničena u dizajnu i detaljnosti simulacije. Ograničenja lokalnih računala, te grafičke optimizacije nisu tema ovog rada te se neće obrađivati, ali treba spomenuti da se na ovim problemima aktivno radi, te će UE5 sa svojim tehnologijama i metodama poput Lumen i Nanite otvoriti još više mogućnosti [29].

### **3.3. Umrežena simulacija fizike i stanje svijeta**

Fizika je iznimno bitan, temeljni dio svake simulacije, svijeta, te tako i svake igre, ona zapravo predstavlja stanje svijeta. Njena prisutnost može biti razlika između dosadnog statičnog svijeta te potpuno dinamičnog i interaktivnog.

Za potrebe ovog rada, najvažniji aspekt je preciznost simulacije, dakle svi objekti moraju na svim računalima imati istu lokaciju, fizika mora biti serverski autoritarna, te mora moći podržavati nekoliko tisuća objekata u jednoj instanci.

Drugi aspekt koji je vrlo bitan je neposredna kontrola fizičkih objekata, dakle vozila – koja su potpuno fizički simulirana i autoritarna preko server, trebaju imati trenutne kontrole za njena klijenta.

Srž problema je da igrač može primiti, te server slati ograničeni skup podataka, nadalje sve što igrač vidi je zapravo prošlost, proporcionalna odazivu naprema serveru, a cilj je održati točnost i trenutnost klijentske simulacije. Za ovo zapravo ne postoji idealno rješenje [30], jer ne postoji način predviđanja budućnosti, moguće su samo metode ublaživanja.

### **3.3.1. Tehnike umreženja**

U ovom poglavlju pregledati ćemo tri tehnike umreženja koje se trenutno koriste u industriji.

Temeljni izbor koji nam se predstavlja pri rješenju ovog problema je jednostavan, umrežiti unose svih likova ili umrežiti poziciju svih objekata. To možemo gledati kao izbor između slanja unosa u simulaciju, ili rezultata simulacije.

Ovaj izbor je važan jer njegove posljedice propagiraju kroz sve umrežene značajke.

#### **3.3.1.1. Deterministički diskretni korak – slanje unosa**

Tehnika deterministički diskretnog (urednog) koraka ( eng. Deterministic lockstep) je koncept da se svakoj simulaciji šalju unosi koji utječu na nju, dakle ako bi svaki korisnik izveo iste unose u isto vrijeme, rezultat bi bio beskonačno veliki broj objekata te sinkronizirana simulacija sa iznimno niskom cijenom [31].

Postoji nekoliko iznimno teških tehničkih problema sa ovom metodom:

- Simulacija mora biti deterministička. Deterministička simulacija je ona simulacija koja za iste ulaze uvijek daje iste izlaze. Dakle za slučajne brojeve moramo koristiti istu kontrolnu sumu (eng. Checksum)
- Zbog raznih metoda i tehnika za optimizaciju broja s pomičnim zarezom (eng. floating point number ) u raznim okruženjima, teško se garantirati da će svako računalo doći do istog rezultata, što ruši determinizam.
- Zahtijeva da se simulacija događa u diskretnim koracima, odnosno u vremenski fiksnim otkucanjima, zbog lakše sinkronizacije, industrijski standard je 60 koraka u sekundi.
- Da bi smo mogli simulirati sve unose, prvo ih trebamo primiti, dakle uvijek će se čekati unos igrača sa najsporijom vezom. To je iznimno pogubno za igre sa puno igrača jer šansa za mrežnom greškom linearno raste s brojem igrača [32].
- Trajni svjetovi zahtijevaju posebnu pozornost. Da bi se novi igrač mogao pridružiti svijetu koji konstantno traje, on treba primiti sve unose koji su se do tada dogodili, a broj pokreta u trajnom svijetu može biti neograničeno velik ( nekoliko gigabajta i slično).

Ovi problemi su tehnički rješivi, ali zahtijevaju da se cijelo okruženje za razvoj dizajnira oko njih. Dakle potreban je fizički sustav koji je potpuno determinističan preko više platformi

te radi u diskretnim koracima. Da bi se riješio problem zakašnjenja unosa, potreban je sustav premotavanja.

U ovoj tehnici i server i klijenti provode simulaciju fizike, klijenti ju provode zato da bi se održala trenutna kontrola i slika simulacije, dok ju server provodi kako bi održao sinkronizaciju te tako bio autoritet o simulaciji, te odobravao ili odbacivao unose (zbog prevencije varanja i slično) što je zapravo vrlo povoljno za serverske performanse.

Sve navedeno se zapravo implementira u novom fizičkom sustavu Epic Gamesa, zvanom Chaos fizikom. Dakle to je deterministički sustav sa mogućnošću premotavanja. Treba spomenuti da je UE4, te tako i UE5, arhitektурno optimiziran za fizičke simulacije, jer je orijentiran prema entitetima [33].

Jedini problem kojem se nije zasad dalo pozornosti je trajnost. Trajni svjetovi mogu se izvoditi satima, tjednima ili čak godinama, provođenje simulacije svih pokreta u toliko dugog vremena nije praktično. Jedno rješenje bilo bi periodično spremanje pozicije i brzine svih objekata u sustavu, te bi se tako spremilo njihovo stanje, bez spremanja i simuliranja potencijalno neograničeno velikih setova unosa. Ova metoda bi se mogla optimizirati tako da spremamo pozicije samo određeno velikih objekata ili samo onoliko objekata koliko želimo spremati i slično. Treba napomenuti da se na ovom problemu u Chaos sustavu za UE5 trenutno radi [34].

Ovu tehniku koriste većinom igre koje su ekskluzivne za jednu platformu ili RTS-ovi sa jako puno simuliranih jedinica kao Age of Empires [35], ali očito će se u budućnosti primjenjivati i na druge igre, jer Epic Games namjerava koristiti alate koje izrađuje za svoje igre.

Nadalje ova tehnika nema ograničenja povezanih sa preciznošću broja s pomičnim zarezom, odnosno svijet može biti velik i precizan koliko god ga definiramo, kontekst za ovaj problem biti će dan u pregledu sljedećih metoda.

Pogledajmo sada moguća ograničenja ovog sustava, s obzirom na danu specifikaciju od 100 Mbps. Dakle 100 Mega bita je 104857600 bitova u sekundi, to nam je budžet.

Prepostavljamo da se koristi kompleksan set kontrola za kretnje i interakciju, baziran na šest mogućih smjerova kretnje (eng. Six degrees of freedom) uz 20 dodatnih sposobnosti, što rezultira sa 32 moguća unosa. 32 unosa možemo prevesti na 32 bit-a odnosno binarne informacije o tome da li su trenutno aktivni ili ne.

Prepostavljamo da svaki lik šalje 30 puta u sekundi 32 bita informacija, što rezultira sa:  
 $30 * 32 \text{ bit} = 960 \text{ bit po sekundi po liku}$

Teoretski rezultat je:

$$\frac{104857600}{960} = 109226 \text{ unosa po sekundi}$$

koje klijent može primiti.

Sa serverske strane, to znači da on mora slati 100 Mega bita 109226 puta, što nas dovodi do:

$$100 \text{ Mbit} * 109226 = 10922600 \text{ Gbit}$$

što je, prema našoj specifikaciji daleko previše, dakle očito je ograničenje na serverskoj strani, te moramo ovaj problem gledati sa stajališta servera.

10 Gigabita (10 000 000 000 bitova) prometa daje nam mogućnost slanja:

$$\frac{10\ 000\ 000\ 000 \text{ bit}}{960 \text{ bit}} = 10\ 416\ 666 \text{ pokreta}$$

Broj igrača koji bi teoretski mogli podržati je korijen ovog broja, odnosno:

$$\sqrt{10\ 416\ 666} = 3227 \text{ igrača}$$

Dakle svakom klijentu šaljemo 3227 pokreta po 960 bitova, i to radimo 3227 puta (šaljemo ih svakom igraču), što nas dovodi od 10 Gigabita. Klijentsko opterećenje bilo bi:

$$3227 * 960 \text{ bit} = 3.1 \text{ Megabit po sekundi}$$

Sa 100 Gigabita prometa, imamo mogućnost slanja 104 166 666 pokreta. Broj igrača koji bi mogli podržati je dakle 10 206, sa klijentskim opterećenjem od 9.8 Mbps.

Pri slanju 60 pokreta u sekundi, što je preporučena metrika [36], i 10Gbit prometa, dolazimo do paketa od 1920 bitova, što nas dovodi do 2282 igrača.

Prednosti ove tehničke su:

- Svi igrači su odmah svjesni cijele simulacije

- Simulacija u sebi može imati neograničeno velik broj fizičkih objekata, odnosno biti neograničeno detaljna
- Veličina paketa po klijentu je iznimno mala.
- Nismo ograničeni preciznošću broja s pomičnim zarezom, možemo imati vrlo velike svjetove

Nedostatci ove tehnike su:

- Teško se skalira sa brojem igrača, pošto smo ograničeni sa serverskom propusnošću.
- Za održavanje sinkronizacije potrebni svi unosi, bez obzira na njihovu prostornu poziciju, jer bi se inače srušila sinkronizacija, zato je ograničena propusnošću mrežnog prometa.
- Pošto se simulacija mora provoditi i na klijentu, to može utjecati na njegove lokalne performanse računala, te bi detaljnost fizičke simulacije bila ograničena nižom granicom hardverskih zahtjeva, ako se ne optimizira u prostornom smislu.
- Razvoj ove tehnike, te samog razvojnog okruženja koje ju koristi je iznimno zahtjevno.

Teoretski, ako bi ovu tehniku prostorno optimizirali, broj igrača u jednom svijetu bio bi neograničen, u svijetu sa neograničeno velikim brojem trenutno sinkroniziranih objekata.

### 3.3.1.2. Interpolacija slike stanja – slanje pozicije

Tehnika interpolacije slike stanja je koncept u kojem se šalju pozicije svih fizičkih objekata. U ovoj tehnici samo server provodi fizičku simulaciju dok korisnik pri primanju pozicijskog ažuriranja, provodi samo lokalnu interpolaciju ( pomicanje objekta ) između zadnje lokalno poznate pozicije te nove pozicije [37]. Rezultat je da klijenti zapravo dobivaju rekonstruirane slike stanja prave simulacije.

Ovo je tehnika koju koriste igre u poglavljima tehničkih ograničenja trenutne generacije.

U većini slučajeva ova tehnika ne ovisi o diskretnim koracima, simulacija ne treba biti deterministička, ne ovisi o najsporijem klijentu, te je zato tehnički puno lakše izvediva i odlično podržava trajne svjetove. Nadalje velika prednost joj je skalabilnost, jer može raditi bez obzira na broj ažuriranja u sekundi.

Pozicijski podaci sastoje se vektora za poziciju ( 96 bitova ) i kvaterniona ( engl quaternion ) za rotaciju ( 128 bitova ) što je ukupno 224 bitova [38].

Ovo je očito neprihvatljivo te se mora optimizirati.

Orijentacija se optimizira tehnikom poznatom pod nazivom najmanja tri, ona se bazira na znanju da dužina kvaterniona mora biti ukupno 1, odnosno:

$$x^2 + y^2 + z^2 + w^2 = 1$$

Dakle možemo poslati indeks najdužeg elementa (dva bita) te ostale tri najmanje komponente. Nakon daljnje kompresije tri najmanjih komponenti na 9 bitova dolazimo do rezultata:

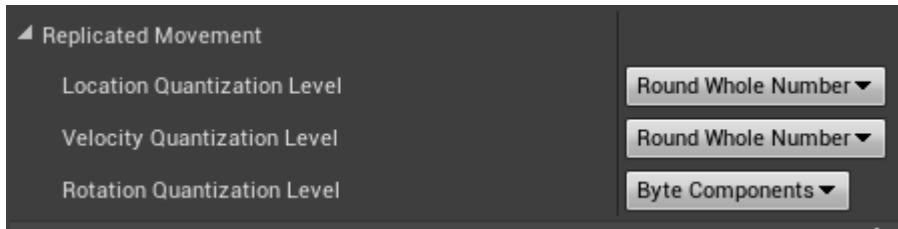
$$2 + 9 + 9 + 9 = 29 \text{ bitova}$$

Optimizacija pozicije ovisi o tome koliko želimo precizno prikazati pozicijske promjene, generalno prihvatljiva preciznost je ona od 2mm. Ukratko to zahtijeva 18 bitova po osi, odnosno 54 bita.

Dakle rezultat je:

$$29 + 54 = 83 \text{ bitova}$$

Ovakav tip kompresije dostupan je i u UE4, u izborniku za replikaciju kretanja.



Slika 13 Kompresija vektora i rotacije u UE4

Moguće je ovo dalje optimizirati delta kompresijom, koja zapravo samo šalje promjene relativne zadnjem ažuriranju. Za ovu optimizaciju potrebni su paketi priznavanja (eng. acknowledge). Generalno nije moguće procijeniti cijenu ove tehnike nakon ove optimizacije, jer ovisi o ponašanju sustava, ali možemo ju procijeniti na 20 ažuriranja po sekundi.

Kao posljedica dizajna arhitekture, broj objekata za koje možemo slati pozicijska ažuriranja je ograničen, zbog veličine pozicijskih podataka, to ne bi bilo praktično. Nadalje, pošto moramo slati što manje pozicijske podatke, veličina broja s pomičnim zarezom je ograničena, što rezultira manjim svijetom sa manjom preciznošću.

### **3.3.1.3. Sinkronizacija stanja**

Većina igara slanje pozicijskih ažuriranja dalje optimizira tehnikom sinkronizacije stanja. Ovdje je ideja da se održi trenutna kontrola provođenjem lokalne simulacije unosa za važnije objekte poput igrača (poput deterministički diskretnog koraka), dok se za ostale objekte selektivno koristi sinkronizacija stanja.

Za selektivnu sinkronizaciju stanja koristi se tehnika akumulatora prioriteta koja stavlja potrebna ažuriranja pozicije u polje te ih onda na bazi njihove važnosti šalje korisnicima, ovo logika može biti podosta zahtjevna za server, te je jedan od glavnih faktora za ograničenu veličinu trenutnih igara kao onih koje smo obradili u poglavljiju tehničkih ograničenja trenutnih igara [39].

Ujedno ovo je jedna od glavnih prednosti ove tehnike - da se količina prometa preko mreže može ograničiti na ciljanu cifru, većinom je to oko 350 kbps [40].

Nadalje tehnički ne postoji ograničenje broja fizičkih objekata zbog optimizacija, ali tu dolazi i do najvećeg nedostatka ovih tehnika; to što klijent nikada zapravo ne zna koje je stanje simulacije. Kako smo vidjeli u poglavljju tehničkih ograničenja trenutnih igara, prihvatljivo je da se simulacija ne može razumno lokalno provoditi sa manje od 10 ažuriranja u sekundi, a za uvjerljivo iskustvo potrebno je 20.

Primjenu ove tehnike možemo vidjeti u UE4 koji koristi kombinaciju sinkronizacije stanja uz mogućnost vraćanja za svoje likove – kako bi se održala trenutnost kontrole, te interpolaciju slike stanja za sve ostale fizičke objekte [41].

Pogledajmo ovaj problem iz perspektive implementacije u UE4. Dakle u UE4 se na klasnoj bazi se zadaje maksimalna udaljenost na kojoj bi se objekt te klase replicirao, te u koji prioritet da se stavi. Ovo je zapravo srž implementacije umreženog drivera za UE4. Ta tehnika se zove umreženo izbacivanje po kvadratnoj udaljenosti (eng. Net Cull Distance Squared).

Na primjer u sljedećem video zapisu [42] u igri Hell Let Loose objekti koji se mogu izgraditi konfiguirirani su tako da imaju manji domet od igrača, to inicijalno ne zvuči problematično dok se ne dogodi situacija gdje je igrač možda iza objekta koji se može izgraditi, kao na videu.



Slika 14 Hell Let Loose – objekt koji se može izgraditi nije repliciran [42]

Nadalje, u slučaju pre agresivne optimizacije pomoću akumulatora prioriteta i velikog opterećenja može doći do gotovo potpune degradacije simulacije, u sljedećem primjeru vidljivi su plutajući igrači sa potpuno desinkroniziranim pozicijom, ovo je dio testa sa 4138 igrača (od planiranih 5000) u igri Scavengers [40]. Dakle prostorne optimizacije ne mogu nam pomoći u slučaju kada su svi igrači na jednom mjestu, odnosno degradacija iskustva je tada velika.



Slika 15 Scavengers desinkronizacija pozicije igrača, vlastita izrada

Možemo zaključiti da je za optimizaciju ove tehnike potrebno jako puno konfiguriranja da bi funkccionirala bez pre velike lokalne de-sinkronizacije i degradacije iskustva.

Ovu metodu, sa diskretnim koracima za logiku igre i fiziku, koristi i igra Apex Legends, gdje se cijelo stanje svijeta na svako osvježavanje replicira svakom klijentu. Posljedica ovog dizajna je da bi se simulacija održala, paketi moraju stići do klijenata, te se zato serversko osvježavanje limitira na 20Hz, ali treba spomenuti da je ovo vrlo dobar rezultat sa skoro trenutnim ponašanjem simulacije - 75 mili sekundi zakašnjenja, što je gotovo neprimjetno [43].

Prednosti pozicijskih tehnika su:

- Fleksibilnost s obzirom na ciljeve mrežnog prometa
- Klijenti ne provode fizičku simulaciju
- Može se prostorno optimizirati

Nedostaci pozicijskih tehnika su:

- Ograničeni broj fizičkih objekata
- Klijenti nikada ne znaju pravo stanje cijelog sustava
- Optimizacije mogu biti skupe za serversku performansu, te ga to ograničava na jedno-serverskoj arhitekturi
- Optimizacije ne mogu pomoći u slučaju da su svi igrači na jednom mjestu te dolazi do primjetne degradacije iskustva
- Veličina i preciznost svijeta je ograničena na preciznost broja s pomičnim zarezom

### **3.4. Više-serverska arhitektura**

Da bi mogli provoditi više interakcija u simulaciji, potrebno je izvesti ili višedretvenost, ili više-serverska arhitektura, ovo predlaže i Dave Ratti u diskusiji o umreženju u UE4 [44].

Dok bi višedretvenost sigurno utjecala na performanse servera, nije toliko skalabilna kao više-serverska arhitektura, jer bi još uvijek bili ograničeni brojem dretvi, dok je broj servera koje međusobno možemo umrežiti teoretski neograničen.

Kada govorimo o više-serverskoj arhitekturi, mislimo na proces obrade jedne simulacije od strane više računala, ovo se još naziva distribuirana simulacija. Ovo je odvojeni pojam od tehnologija baziranih na krhotinama (eng. Shard) gdje povezujemo više samostalnih simulacija.

Postoji nekoliko igara koje koriste ovu arhitekturu, poput Star Citizen, njihova inačica ove arhitekture zove se „server meshing“ [45]. Nadalje postoji igra The Elder Scrolls Online sa svojom „Megaserver“ tehnologijom [46] koja je služi za skaliranje broja igrača, ali u više svjetova. Naposljetku igra Mortal Online 2 [47] ju planira koristiti za skaliranje broja igrača u jednom svijetu.

Postoji nekolicina javno dostupnih tehnologija, jedno od njih, zvano Aether engine [48] nekolicinu prednosti poput dinamično skaliranje broja dostupnih servera ali trenutno nije spremno za produkciju.

### **3.4.1. Spatial OS**

U ovom radu koristi će se javno dostupna tehnologija Spatial OS [5], tvrtke Improbable, jer ima već stabilnu i produkcijski spremnu inačicu.

#### **3.4.1.1. Temeljni pojmovi**

Sada ćemo objasniti temeljne pojmove i koncepte vezane uz njegovu funkciju.

Radnici (eng. Worker) [49] izvode operacije vezane uz Spatial OS sustav, mogu biti serverski ili klijentski. Serverski radnik jednak je klasičnom serveru igre, dok je klijentski radnik jednak klasičnoj klijentskoj instanci igre.

Spatial OS simulaciju percipira kroz tri koncepta [50]:

- Svijet
- Entiteti
- Komponente

Svijet je izvor kanonske istine za cijelu simulaciju. Sve podatke koje želimo dijeliti između radnika moramo spremiti u entitete, to su svi objekti u igri [51]. Svi entiteti se sastoje od komponenata koje spremaju podatke [52].

Ukratko logiku pišemo na radnicima koji koriste podatke iz komponenata na entitetima. Ovakva arhitektura je potrebna jer želimo da više radnika može pristupiti podacima, bez međusobnog komuniciranja.

Ovo je zapravo proširenje uzorak dizajna zvan entitet, komponenta, sustav (eng. Entity, component, system – kratica ECS) ali se još uvijek jako dobro mapira na UE4 sustav koji se centririra oko entiteta, koji su povoljni za fizičke simulacije, ali treba spomenuti da dobro radi i sa sistemski centriranim sustavima poput Unity, koji su povoljni za AI [33].

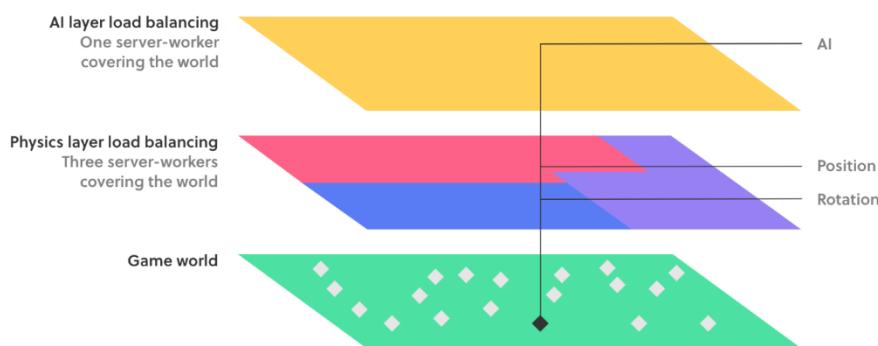
Naposljetu, treba spomenuti da za spremanje stanja svijeta koristi se visoko propusna baza podataka sa niskom latencijom [5].

### 3.4.1.2. Distribucija simulacije i povezani koncepti

Spatial OS pruža dva koncepta za distribuciju simulacije:

- Slojevitost (eng. layers)
- Prostorno uravnoteženje opterećenja ( eng. Load balancing )

Slojevitost je koncept u kojem se grupe komponenata organiziraju po radnici koji simuliraju svijet [53]. Na sljedećoj slici prikazan je primjer gdje na vrhu imamo sloj koji je odgovoran za komponentu upravljača umjetnom inteligencijom, te jedan sloj koji je zadužen za fizičku simulaciju.



Slika 16 Slojevitost [53]

Prostorno uravnoteženje opterećenja je podjela jednog sloja po njegovom prostornom odnosno geografskom načelu, dakle jedan radnik brine se samo za komponente na određenom dijelu svijeta [54]. Treba napomenuti da prostorno uravnoteženje opterećenja kao značajka još nije spremno za produkciju, te se zato ne koristi u ovom radu, ali njena primjena zahtijevala bi minimalne modifikacije.

Sada ćemo objasniti neke od povezanih koncepata vezanih za distribuciju simulacije.

Autoritet je koncept u kojem samo jedan radnik može imati pravo pisanja podataka o komponenti u jednom trenutku [55]. Ovo je važno za prostorno uravnoteženje opterećenja jer se tamo autoritet mijenja, taj koncept se zove predaja (eng. handover) [56].

Interes je koncept u kojem jedan radnik zahtjeva čitanje podataka o komponenti [57]. Ovaj koncept omogućava klijentima da primaju samo informacije relevantne za njih.

Na sljedećem izvoru vidimo primjer gdje je punom crtom označeno područje gdje je radnik autorativan, a iscrtano je označeno područje interesa, ona se preklapaju zbog predaje autoriteta nad entitetom [58].

Trajnost (eng. persistance) [59] je koncept u kojem se podaci svih komponenti spremaju u slike sustava (eng. Snapshot) kako bi se svijet kasnije mogao obnoviti [60].

### 3.4.1.3. Ograničenja i maksimalni broj igrača

Pojam operacija je iznimno važno ograničenje za Spatial OS sustav, to je bilo koje pisanje ili čitanje podataka [58]. U 2019.g. postavljena je metrika od 6 milijuna operacija [61], što je rezultiralo sa 6000 igrača, ovo je postignuto sa UE4, odnosno sa pozicijski orientiranim umreženjem. U pravom okruženju, već prije spomenute igre Scavengers, postignuto je 4138 igrača na jednom mjestu, ali sa vrlo malo interakcije, odnosno bez oružja i sličnog, treba napomenuti da u 2021.g. dana nova cifra, od 250 milijuna operacija, što je rezultiralo sa 10,000 igrača na 30Hz [40].



Slika 17 Operacije [61]

Naposljetu, treba zamijetiti da je trenutno postavljeno ograničenje na kupljenim serverima, od 200 igrača, te je ovo metrika koja će nas voditi u dizajnu funkcionalnosti igre i meča [62].

#### **3.4.1.4. Dodatna rješenja za distribuirane svjetove**

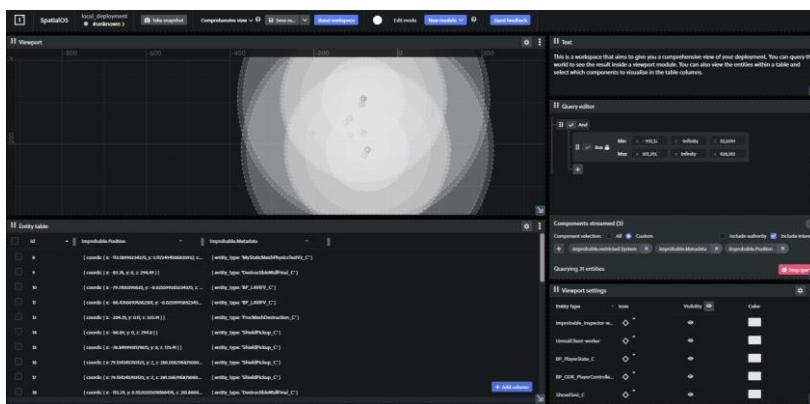
Spatial OS je riješio i neke od najtežih problema distribuirane simulacije, kao distribucija fizičke simulacije, njihovo rješenje pokriva i slučajeve gdje dolazi do preklapanja autoriteta [63].

Nadalje, riješen je i problem velikih svjetova. To je problem sa tipom memorije za spremanje pozicije igrača na serveru. Najveći mogući broj s pomičnim zarezom od 32 bita [64] je 2,147,483,647 te zato je moguće definirati svijet velik samo otprilike 20 km kvadratnih. Postoje metode za ublažavanje ovog problema poput pomicanja centra svijeta (eng. World origin shift) ali one komplikiraju proces razvoja igre [65].

Ovaj problem se također rješava u Unreal Engine 5 pomoću koordinata velikih svjetova (eng. Large World Coordinates) kojom se uklanjuju ograničenja veličine svijeta [66] [67], te se zato ovaj problem neće obrađivati u ovom radu.

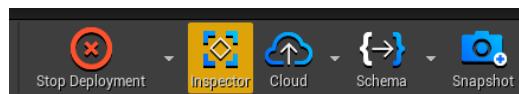
### **3.4.1.5. Inspektor svijeta**

Inspektor svijeta je web alat za pregled stanja Spatial OS svijeta. Pomoću njega možemo uživo vidjeti što se događa na postavljenom poslužitelju te svim povezanim radnicima [68]. Prikazuje sve entitete, komponente te radnike i njihove interese, kao i pod kakvим su opterećenjima.



Slika 18 Inspektor svijeta

Pokreće se klikom na gumb inspektora.



Slika 19 Izbornik Spatial OS u UE4

### 3.4.1.6. Istoimeni pojmovi u UE

Kod implementacije Spatial OS u UE, postoji još nekoliko pojmove na koje treba obratiti pozornost, odnosno kako se one prevode u Spatial OS pojmove [69], što je vidljivo u sljedećoj tablici:

Tablica 1 Istoimeni pojmovi UE i Spatial OS [69]

Unreal Engine	Spatial OS
Lik (eng. Actor)	Entitet (eng. Entity)
Replcirano svojstvo (eng. Replicating property)	Polje (eng. Field)
Klijentski/Serverski pozivi na daljinski postupak (eng. Client/Server Remote Procedure Calls – kratica RPC)	Komanda (eng. Command)
Umreženi multi-poziv RPC (eng. NetMulticast RPC)	Događaj (eng. Event)
Uvjetna replikacija (eng. Replication Condition)	Dizajn komponente (eng. Component design)

Važno je upoznati se sa pojmovima poziva na daljinski postupak te umreženi multi-poziv [70].

Poziv na daljinski poziv je funkcija koja se poziva lokalno, ali izvodi na daljinu, na drugom računalu.

Umreženi multi-poziv je tip RPC koji se poziva sa servera, i izvodi na njemu te na svim povezanim klijentima.

### 3.4.1.7. Kako iterirati u Spatial OS

Naposljetku, jako je važno znati kako iterirati pri izradi igre sa Spatial OS [71]. Kako je vidljivo na danom izvoru, ako smo napravili nekakve promjene koje bi utjecale na replicirana svojstva, moramo obnoviti shemu prije postavljanja igre na poslužitelja.

### 3.4.2. Upravljanje interesom klijenta igre

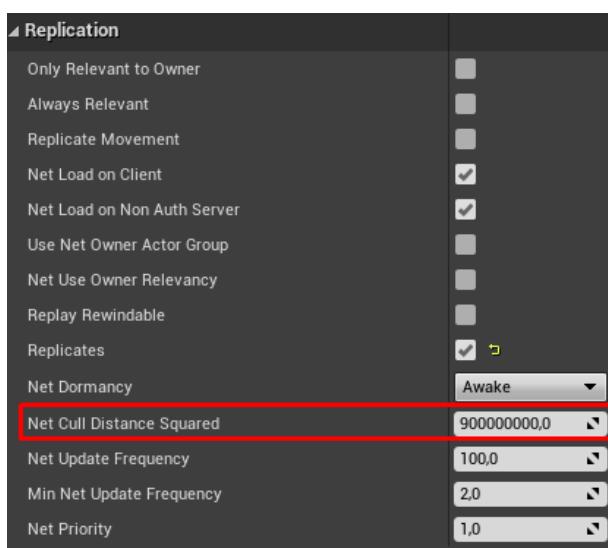
U ovom poglavlju objasniti ćemo zašto je potrebno upravljati interesom klijenata igre, te kroz koje metode to možemo ostvariti.

Kao posljedica pozicijskih ažuriranja, ne možemo preuzeti podatke o svim komponentama odjednom, jer bi to očito bilo previše podataka, uostalom nešto što se događa na potpuno drugoj strani mape od igrača vjerojatno mu nije relevantno.

#### 3.4.2.1. Umreženo izbacivanje po kvadratnoj udaljenosti

Ovaj slučaj nas dovodi do prve metode koju možemo koristiti to je već prije spomenuto umreženo izbacivanje po kvadratnoj udaljenosti [72]. Dakle na klijentskog radnika se stavlja sfera interesa određenog radijusa, u kojoj se iskazuje interes za neki entitet. Definira se na razini entiteta odnosno klase u UE4, u replikacijskom izborniku. Pogledajmo sad jedan primjer.

Na sljedećoj slici vidljiva je konfiguracija entiteta uništivog terena, koji će se detaljnije kasnije obrađivati. Pošto želimo da se ovaj entitet replicira na 300 metara, potrebno je to pretvoriti u centimetre, odnosno 30000cm, te unijeti njegov kvadrat odnosno 900000000,0. Kao što smo prije spomenuli, ovo je srž UE4 umreženog drivera, ovaj upit se pretvara u Spatial OS upit interesa u umreženom driveru Spatial OS-a.



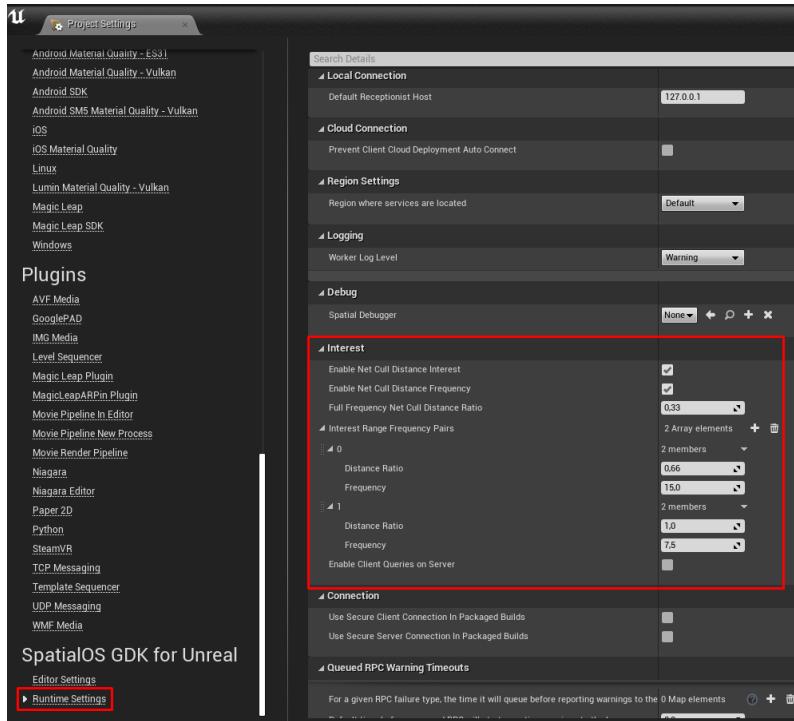
Slika 20 Konfiguracija umreženog izbacivanja po kvadratnoj udaljenosti

Nešto ljepeš prikazan primjer dan je u samoj dokumentaciji ove metode. Dakle na izvoru je vidljiva udaljenost za igrače je postavljena na 200 metara, dok je za vozila postavljena na 400 m, igrač kojeg promatramo nalazi se u centru.

### 3.4.2.2. Umreženo izbacivanje po udaljenosti sa modifikacijom frekvencije

Prema izvornoj izvedbi u UE4, entiteti unutar zadane udaljenosti se klijentu šalju punom zadanom frekvencijom, to je nepoželjno zbog veličine mrežnog prometa, te utjecaja na performanse procesora. Nadalje, očito je da su entiteti bliže klijentu važniji od onih dalnjih, te zato od njih treba primati više pozicijskih ažuriranja. Za ovo nam služi metoda umreženog izbacivanja po udaljenosti sa modifikacijom frekvencije (eng. Net Cull Distance Frequency) [73]. Treba napomenuti da je ovo metoda koja, ako je uključena, tada se primjenjuje na sve upite, te se ne može konfigurirati na razini pojedinog upita.

Da bi uključili ovu metodu, moramo u UE4 odabrati  
Edit > Project Settings > Spatial GDK for Unreal > Runtime Settings > Interest.  
Ovdje stavljamo kvačicu na Enable Net Cull Distance Frequency upit.



Slika 21 Uključivanje umreženog izbacivanja po udaljenosti sa modifikacijom frekvencije u postavkama projekta

Nadalje je potrebno konfigurirati ovu metodu. Prvo definiramo u kojem omjeru udaljenosti želimo punu frekvenciju (eng. Full frequency net cull distance ratio), dakle na

vrijednosti 0,33, u punoj frekvenciji dobivati ćemo ažuriranja onih entiteta koja su udaljena 1/3 od postavljene udaljenosti. Ovo je dalje moguće konfigurirati u izborniku za parove udaljenosti i frekvencije (eng. Interest range frequency pairs).

U ovom primjeru smo zadali da se na 2/3 udaljenosti dobiva 15 ažuriranja u sekundi, a na punoj udaljenosti 7,5 ažuriranja u sekundi.

### **3.4.2.3. Entiteti nad kojima uvijek imamo interes**

Za neke entitete želimo uvijek primati ažuriranja, bez obzira na njihovu poziciju, to se izvodi pomoću metode zvane „Uvijek zainteresirani“ (eng. AlwaysInterested) [74]. Za to se koristi UE4 „UPROPERTY“ specifikator „AlwaysInterested“, to mora biti objektna referenca (AActor ili UObject) ili imati specifikator „Replicated“ ili „Handover“. Primjer ove metode bi mogli izvesti nad zdravljem naših suigrača, za koje uvijek želimo znati status.

### 3.4.2.4. Komponenta interesa za lika

Ovo je UE4 komponenta koja se može dodati bilo kojem liku, sastoji se od:

- Liste upita
- Prekidača za umreženo izbacivanje po kvadratnoj udaljenosti

Ona pruža način za dubinsko definiranje interesa. Kada klijent posjeduje lika sa tom komponentom, lista upita unutar nje definira podatke koje taj klijent prima, te se zato neće koristiti prije navedene metode, nego samo ova. Treba napomenuti da jedan lik može imati samo jednu ovakvu komponentu odjednom. Dakle, za razliku od Unreal drivera koji definira veze od objekta prema korisniku samo jednom, ovaj sustav nam dopušta definiranje veza od korisnika prema objektima, i to više puta [75].

Upiti mogu biti:

- Jedno ograničenje
- Više ograničenja povezanih sa logičkim operatorima „OR“ ili „AND“.

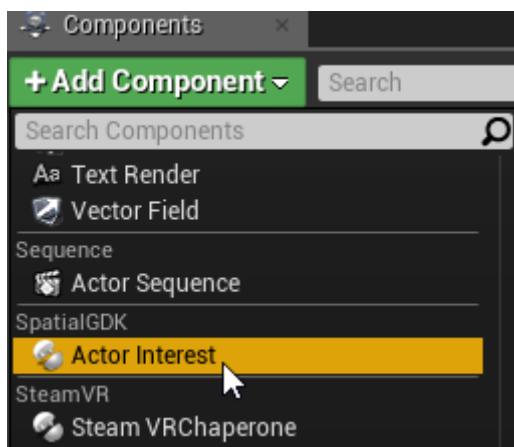
Sljedeća tablica pokazuje listu mogućih ograničenja:

Tablica 2 Opis ograničenja [57]

Ograničenje	Opis
UOrConstraint	Zadovoljeno ako su zadovoljena neka od njegovih unutarnjih ograničenja.
UAndConstraint	Zadovoljeno ako su zadovoljena sva njegova unutarna ograničenja.
USphereConstraint	Uključuje sve likove unutar sfere usredotočene na određenu točku.
UCylinderConstraint	Uključuje sve likove unutar cilindra usredotočenog na određenu točku.
UBoxConstraint	Uključuje sve likove u okviru koji ograničava okvir usredotočen na navedenu točku.
URelativeSphereConstraint	Uključuje sve likove unutar sfere usredotočene na liku koji ima komponentu interesa za lika.

URelativeCylinderConstraint	Uključuje sve likove unutar cilindra usredotočenog na liku koji ima komponentu interesa za lika.
URelativeBoxConstraint	Uključuje sve likove unutar ograničenog okvira usredotočenog na liku koji ima komponentu interesa za lika.
UCheckoutRadiusConstraint	Uključuje sve likove klase ili izvedene klase unutar cilindra usredotočenog na liku koji ima komponentu interesa za lika.
UActorClassConstraint	Uključuje sve likove klase. Po želji možete uključiti izvedene razrede.
UComponentClassConstraint	Uključuje sve likove s komponentom lika određene klase. Po želji možete uključiti izvedene razrede.

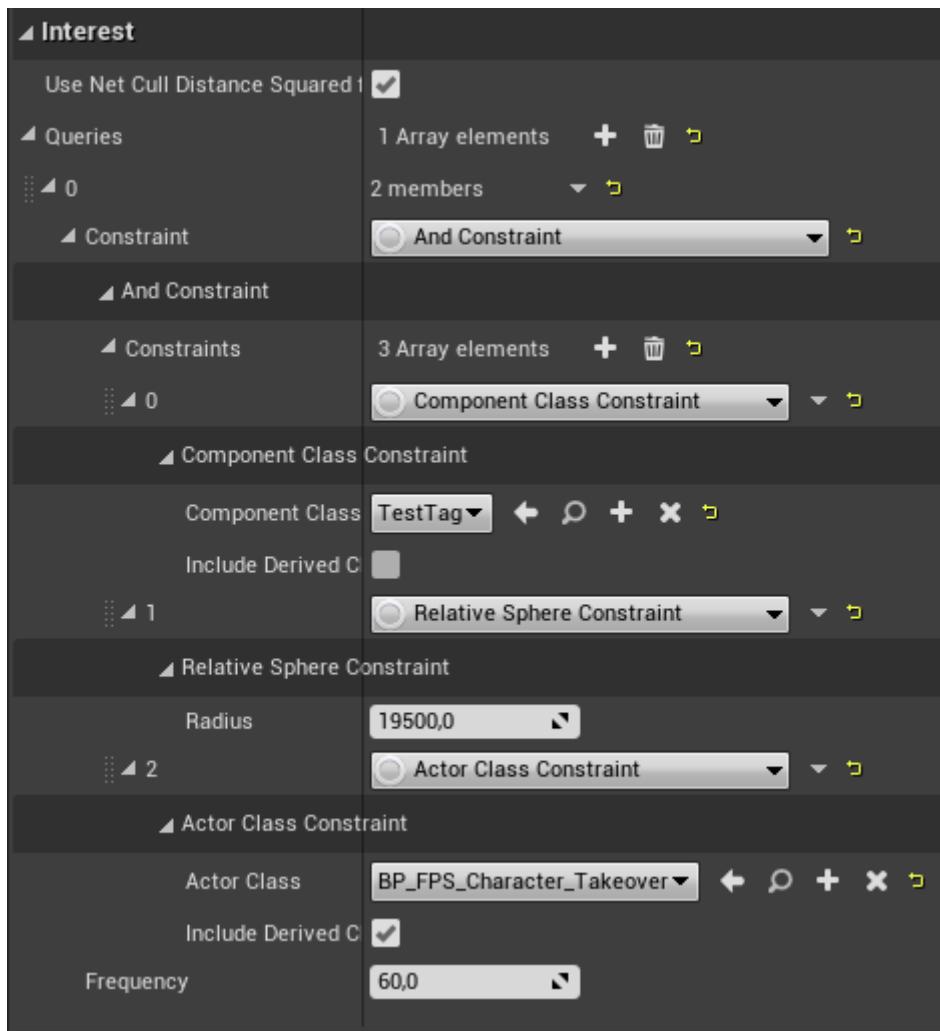
Komponenta se na lika dodaje otvaranjem nacrtu lika, pod izbornikom komponenti, odabirom navedene komponente te pritiskom tipke za dodavanje.



Slika 22 Komponenta interesa za lika u UE4

Ova komponenta omogućava nam vrlo slične modifikacije kao one izrađene za Battlefield 4. Dakle možemo stvoriti zasebne upite za aute, avione i pješadiju.

Na sljedećoj slici konfigurirali smo komponentu na upravljaču lika (eng. Player Controller) tako da se primaju ažuriranja od samo onih likova (tipa pješadije) koji imaju komponentu „TestTag“ i unutar su sfere od 19500 centimetara.



Slika 23 Konfigurirana komponente interesa za lika

Ova komponenta razvijena je zbog toga jer se neke pojedinosti lika ne stavljuju u komponente, nego u varijable. Na primjer tim kojem lik pripada je jednostavna varijabla sa identifikatorom tima, ali varijable komponenata ne možemo koristiti za njihovu identifikaciju u ograničenjima odnosno u shemi. Dakle komponenta „Tag“ ili oznaka, rješava ovaj problem jer se komponente mogu mijenjati u živom okruženju, dodavati ili ukloniti sa bilo kojeg lika, te se koristiti za identifikaciju interesa.

### 3.4.3. Dinamično upravljanje interesom klijenta igre

Dok je komponenta za upravljanje interesom lika vrlo korisna, nije dinamična, odnosno nije moguće ju modificirati tijekom igranja. Zbog ovog ne bi bilo na primjer moguće promjeniti interes na neprijateljski tim ako bi promjenili tim, nadalje optimizacije za vidljivo polje također ne bi bile moguće.

#### 3.4.3.1. Istraživanje umreženog drivera

Kako bi smo mogli dinamički mijenjati interes našeg klijentskog radnika potrebno je prvo proučiti kako se interes inače inicijalizira. Ovo je bio dug proces proučavanja rada samog koda te se dubinski provjeravala funkcionalnost svakog dijela Spatial OS.

Rješenje leži u dvije klase, to su:

- USpatialNetDriver
- USpatialSender

Samo rješenje izgleda ovako:

```
PlayerBubbleAnd =  
CreateDefaultSubobject<UAndConstraint>(TEXT("PlayerBubbleAnd"));  
  
PlayerBubbleComponent = CreateDefaultSubobject<UComponentClassConstraint>  
(TEXT("PlayerBubbleComponent"));  
PlayerBubbleComponent->ComponentClass = UTagComponent::StaticClass();  
PlayerBubbleComponent->bIncludeDerivedClasses = false;  
  
RelativeSphere = CreateDefaultSubobject<URelativeSphereConstraint>  
(TEXT("Sphere1"));  
RelativeSphere->Radius = 19500;  
  
PlayerBubbleAnd->Constraints.Empty();  
PlayerBubbleAnd->Constraints.Add(PlayerBubbleComponent);  
PlayerBubbleAnd->Constraints.Add(RelativeSphere);  
  
ActorInterestComponent->Queries[0].Constraint = PlayerBubbleAnd;  
  
USpatialNetDriver* driver = (Cast<USpatialNetDriver>  
(GetWorld()->GetNetDriver()));  
USpatialSender* sender = driver->Sender;  
sender->UpdateInterestComponent(this);
```

Dakle, na klijentskom radniku, prvo moramo definirati ograničenja („PlayerBubbleAnd“, „PlayerBubbleComponent“ i „RelativeSphere“) te ih dodati u komponentu interesa. Nadalje moramo dobaviti Spatial OS umreženi driver iz svijeta te dobiti vezu na samog radnika, to je u ovom slučaju varijabla „sender“, ovdje možemo pozvati funkciju „UpdateInterestComponent“ koja nam omogućava ažuriranje interesa. Treba napomenuti da ova funkcija nije nigdje dokumentirana, te se nigdje drugdje ne koristi. Ovo sve se mora odraditi na serveru, odnosno ako bi htjeli aktivirati promjenu sa klijenta, trebamo obaviti RPC.

Nadalje, tijekom izvršavanja ovih testiranja otkrivena je greška pri ograničenju tipa „UComponentClassConstraint“, koje zapravo nije radilo jer se shema nije dobro obrađivala, konkretno radi se o promjenama nad klasama „SpatialActorChannel“ te „SpatialClassInfoManager“ [76].

### 3.4.3.2. Interes na bazi vidljivog polja klijenta

Da bi izveli interesne upite bazirane na vidljivom polju klijenta, trebamo ispred klijenta stvoriti interesnu sferu, te ju micati s obzirom na njegov pogled, sfera treba biti dovoljno velika da prekrije cijeli vidokrug klijenta.

Idealno mjesto za implementaciju ovoga je u definiciji samog upravljača lika, odnosno u klasi „GDKPlayerController“. Koristiti ćemo funkciju „SetControlRotation“, ona se poziva svaki puta kada se na serverskoj strani dobiva ažuriranje rotacije, što nam ušteđuje jedan bespotreban RPC. Treba napomenuti da smo ovom modifikacijom izašli iz sustava drivera, te ju je potrebno optimizirati, ovo radimo jednostavnim brojačem koji izvodi funkciju za promjenu interesa tek nakon 60 otkucaja, ovo se radi jer je promjena interesa zahtjevna operacija za procesor.

```
436 void AGDKPlayerController::SetControlRotation(const FRotator& NewRotation)
437 {
438     Super::SetControlRotation(NewRotation);
439     updateCounter++;
440     if (GetLocalRole() == ROLE_Authority && updateCounter%60==0)
441     {
442         // Networked client in control.
443         QueryTest();
444         updateCounter = 0;
445     }
446 }
```

Slika 24 Funkcija postavljanja rotacije lika u UE4

Implementacija se sastoji od inicijalizacije ograničenja u konstruktoru, te ažuriranja istih u funkciji „QueryTest“.

Prvo moramo izračunati centar sfere interesa, odnosno kamo korisnik gleda. Treba spomenuti da ovo ne uzima u obzir korisnikovu rezoluciju, te je to nešto što bi se trebalo ažurirati u dalnjim iteracijama.

```
314 void AGDKPlayerController::QueryTest()
315 {
316
317     FVector vector;
318     FRotator rotator;
319     FVector end;
320
321     GetPlayerViewPoint(vector, rotator);
322
323     //1. FOV 100
324     end = vector + (rotator.Vector() * 10000.f);
325
326     Sphere1->Center = end;
```

Slika 25 Funkcija za ažuriranje interesa

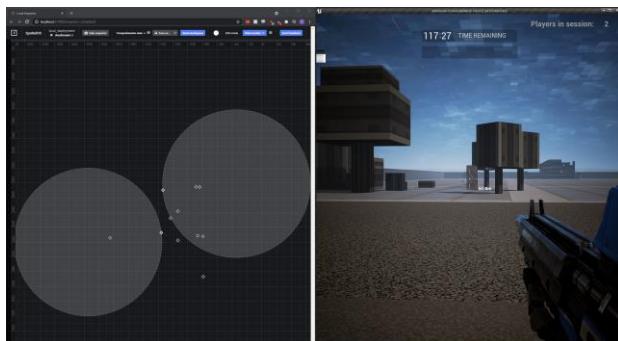
Nakon toga moramo ažurirati ograničenje te ga dodati komponenti interesa, radi se o jednostavnom sfernom te klasnom upitu unutar „AND“ upita.

```
328     FOVConstraint1->Constraints.Empty();
329     FOVConstraint1->Constraints.Add(Sphere1);
330     FOVConstraint1->Constraints.Add(Actor1);
331
332     ActorInterestComponent->Queries[6].Constraint=FOVConstraint1;
333 }
```

Slika 26 Postavljanje ograničenja za vidljivo polje

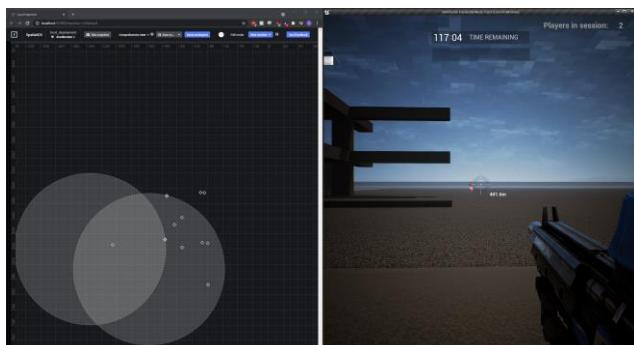
Naposljetku pozivamo već prije prikazano ažuriranje komponente interesa. Rezultat je vidljiv u navedenom video zapisu [77].

Na lijevoj slici je vidljiv inspektor koji prikazuje interes pojedinog klijenta, vidljiva je interesna sfera koja prekriva vidljivo polje klijenta unutar 100 metara od njegove lokacije.



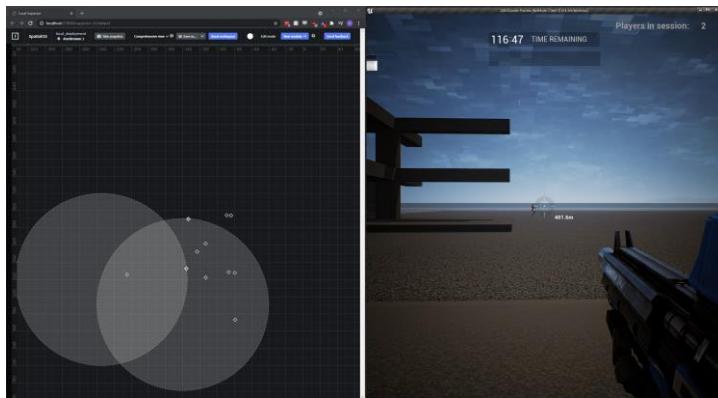
Slika 27 Inspektor koji prikazuje interes pojedinog klijenta

Kada se klijent okreće prema drugom klijentu, on se replicira, u sljedećoj slici vidljiv je proces stvaranja lika.



Slika 28 Stvaranje lika pri replikaciji

Na sljedećoj slici prikazan je potpuno repliciran klijent.



Slika 29 Potpuno repliciran lik

### 3.4.3.3. Buduće iteracije

Ideja je iskoristiti sve prikazane metode kako bi izradili najefikasniji model replikacije. Podijeliti ćemo interes svakog klijenta prema sljedećim aspektima: vidljivo polje i blizina (sfera oko klijenta). Vrijednosti su zadane sa namjerom da se održi što bolje iskustvo za klijenta, tako da se najbliži entiteti osvježavaju što brže, a najbrže oni koji su u vidljivom polju, kao što je izvedeno u Battlefield 4.

Tablica 3 Ažuriranja po polju

Udaljenost (eng. Distance)	Vidljivo polje (eng. FOV) - Hz	Blizina (eng. Vicinity) - Hz
100m	20	8
350m	15	6
700m	5	2
2000m	2	1
6000m	1	0 (ne obrađujemo)
19000m	1	0 (ne obrađujemo)

Nadalje slijedi tablica sa prosječnim brojem klijenata po polju. na ove vrijednosti ne možemo utjecati, one su rezultat kretanja klijenata kroz svijet. Ovo je samo procjena bazirana na iskustvu igranja, pretpostavlja se da je broj igrača veći što je veća udaljenost te da je više igrača u blizini klijenta nego u vidljivom polju.

Tablica 4 Prosječni broj klijenata po polju

Udaljenost (eng. Distance)	Vidljivo polje (eng. FOV) - klijenata	Blizina (eng. Vacinity) - klijenata
100m	16	32
350m	32	64
700m	64	128
2000m	128	256
6000m	256	0 (512)
19000m	512	0 (1024)

Dobivamo da je u vidljivom polju i blizini klijenta prosječno 1488 drugih igrača. Kada pomnožimo ove dvije tablice, dobivamo ukupan broj interakcija po klijentu.

Tablica 5 Broj interakcija po polju

Udaljenost (eng. Distance)	Vidljivo polje (eng. FOV) - interakcija	Blizina (eng. Vacinity) - interakcija
100m	320	256
350m	480	384
700m	320	256
2000m	256	256
6000m	256	0
19000m	512	0

Dobivamo ukupno 3296 interakcija po klijentu, ovaj broj može se još smanjiti tako da neprijateljski tim ažuriramo pod punim opterećenjem zadanim u tablici za ažuriranja po polju, dok naš tim ažuriramo recimo upola manje puta. Ovo se izvodi sa prije izvedenim „Tag“ sistemom.

$$(AžuriranjaNeprijatelja * 0,5 + AžuriranjaPrijatelja * 0,5) * Interakcija$$

$$(1 * 0,5 + 0,5 * 0,5) * 3296 = 2472$$

Dolazimo do 2472 interakcije po klijentu, ako uzmemo u obzir mogućih 6 milijuna operacija u sekundi, možemo doći do maksimalnog broja igrača.

$$\frac{6000000 \text{ operacija}}{2472 \text{ operacija po igraču}} = 2427 \text{ igrača}$$

Treba spomenuti da ovo predviđanje ne uzima u obzir projektile, fizičke objekte i slično, s druge strane operacije nisu baš idealna metrika jer to je prosječno mogući broj operacija iz testiranja tvrtke Improbable, što nije na njihovu krivicu, jer je općenito teško procijeniti prosječne mogućnosti dinamičkih svjetova, nadalje ona ovisi o dostupnom broju servera te performansi njihovih procesora.

Moguća je daljnja optimizacija, gdje bi dinamički izmjenjivali stopu ažuriranja s obzirom na broj igrača ili fizičkih objekata sa kojima je pojedinačni igrač u interakciji, ali treba spomenuti da se ta stopa izražava u cijelim brojevima, odnosno nije moguće osvježavati manje od jednom u sekundi (brojke između 0 i 1), ali je moguće ne osvježavati određeno polje (0 puta u sekundi).

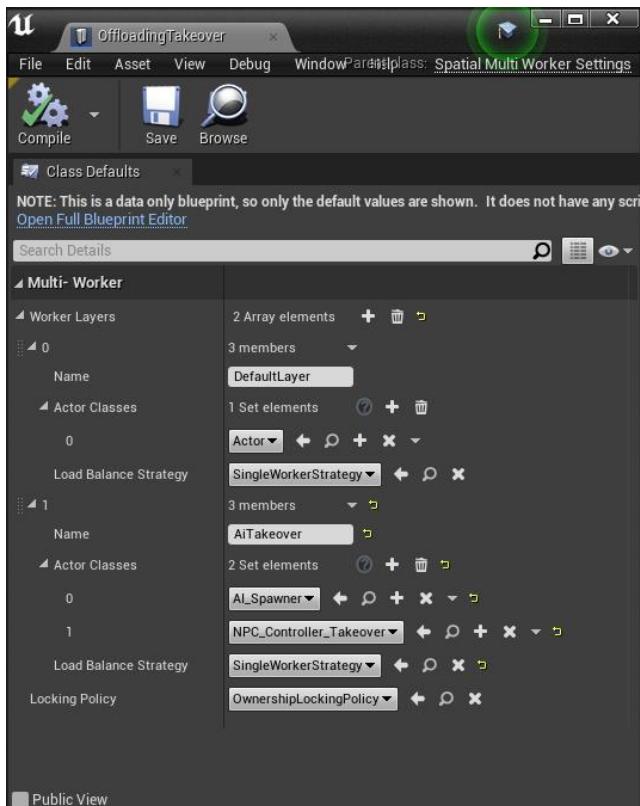
### 3.4.4. Uspostavljanje sloja za umjetnu inteligenciju

Umjetna inteligencija je odličan kandidat za postavljanje na svoj sloj, pošto je vrlo skup za procesiranje, ali latencijski tolerantan [78].

Da bi smo konfigurirali naš svijet (mapu) sa ovim slojem, potrebno je stvorimo konfiguraciju, odnosno novu klasu koja nasljeđuje „SpatialMultiWorkerSettings“, u ovom slučaju ona se nalazi u putanji „Content/TakeoverPrototype/Offloading Takeover“. Sada možemo konfigurirati naš svijet.

Konfigurirati ćemo ga u dva sloja, prvi sloj „DefaultLayer“ biti će konfiguriran za obradu svih klasa koje bi se mogle pojaviti u igri, odnosno klase „Actor“, zasad, zbog nedostupnosti prostornog uravnoteženja opterećenja, koristimo strategiju jednog radnika za sloj, odnosno „SingleWorkerStrategy“.

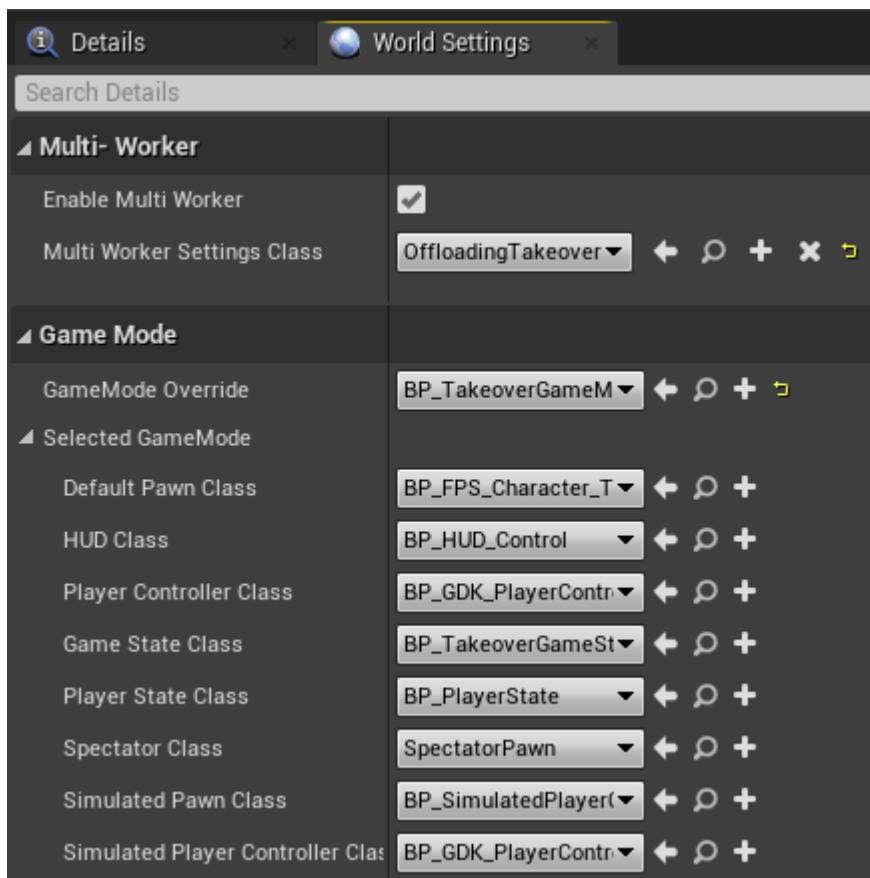
Drugi sloj, nazvan „AiTakeover“ koristit će se za obradu umjetne inteligencije, u sebi će sadržavati klasu stvaratelja umjetne inteligencije, odnosno „AI\_Spawner“, te klasu upravljača umjetne inteligencije, odnosno „NPC\_Controller\_Takeover“, sa prije navedenom strategijom.



Slika 30 Konfiguracija slojeva

Nakon kompilacije ove konfiguracije, možemo ju dodati u svijet odnosno mapu.

Otvorimo mapu koju želimo konfigurirati, dostupna na putanji „Content/TakeoverPrototype/Takeover\_Medium“. Te se prebacujemo na izbornik „World Settings“, ako on nije vidljiv, moguće ga je uključiti sa „Window/World Settings“.



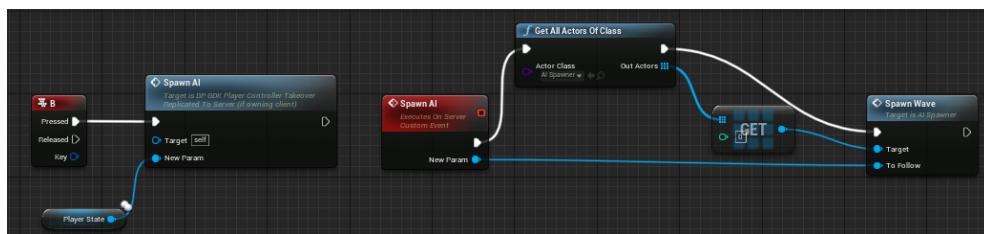
Slika 31 Dodavanje konfiguracije svjetu

Ovdje odobravamo korištenje više-serverske tehnologije odnosno slojeva pomoću „Enable Multi Worker“, te odabiremo konfiguraciju pod „Multi Worker Settings Class“, gdje stavljamo prije stvorenu klasu „OffloadingTakeover“.

### 3.4.4.1. Stvaranje umjetne inteligencije

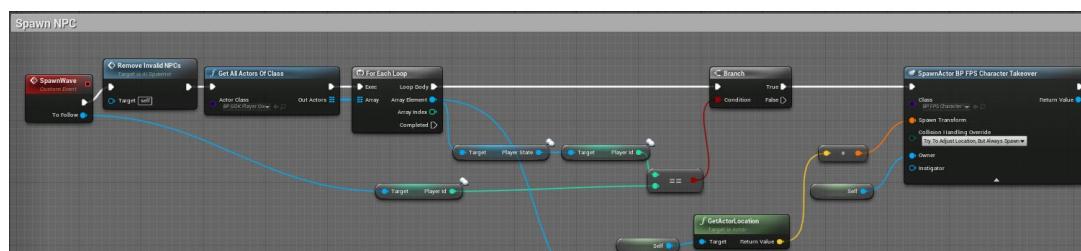
Da bi smo mogli pratiti rad AI koristiti ćemo prije navedenu klasu stvaratelja, odnosno klasu „AI\_Spawner“. Ona će se brinuti za sve funkcije sa serverske strane, dakle za stvaranje AI, za njeno preuzimanje te davanje naredbi.

Za potrebe ovog projekta stvaranje AI se radi na zahtjev klijenta, pritiskom na tipku „B“, što je vidljivo u klasi „BP\_GDK\_PlayerController“. Ovdje preko RPC na serverskoj strani pronalazimo sve stvaratelje (postoji samo jedan) te pozivamo njegovu funkciju za stvaranje. Šalje se i varijabla stanja igrača kako bi smo ga mogli jedinstveno identificirati.



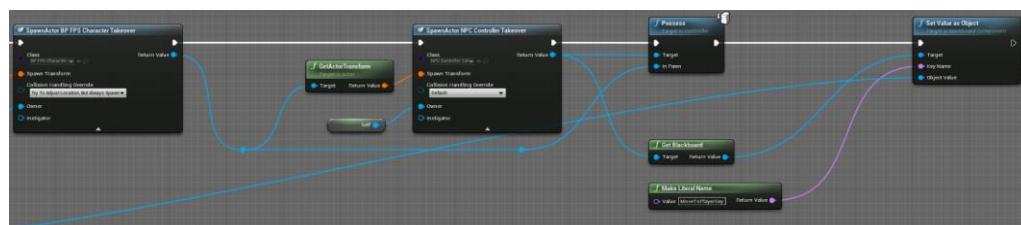
Slika 32 Zahtjev klijenta za stvaranje AI

U stvaratelju se poziva funkcija, koja pronalazi igrača te stvara AI u svijetu.



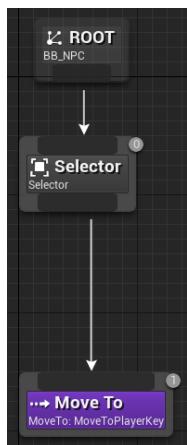
Slika 33 Funkcija za stvaranje AI u stvaratelju

Nadalje stvara se upravljač za AI, koji ga tada posjeduje, naposljeku zadaje se varijabla za kontroliranje ponašanja AI u drvu odluka, da on zna koga treba pratiti.



Slika 34 Stvaranje upravljača za AI

Drvo odluka je vrlo jednostavno, prati se zadani lik.



Slika 35 Jednostavno drvo odluka za AI

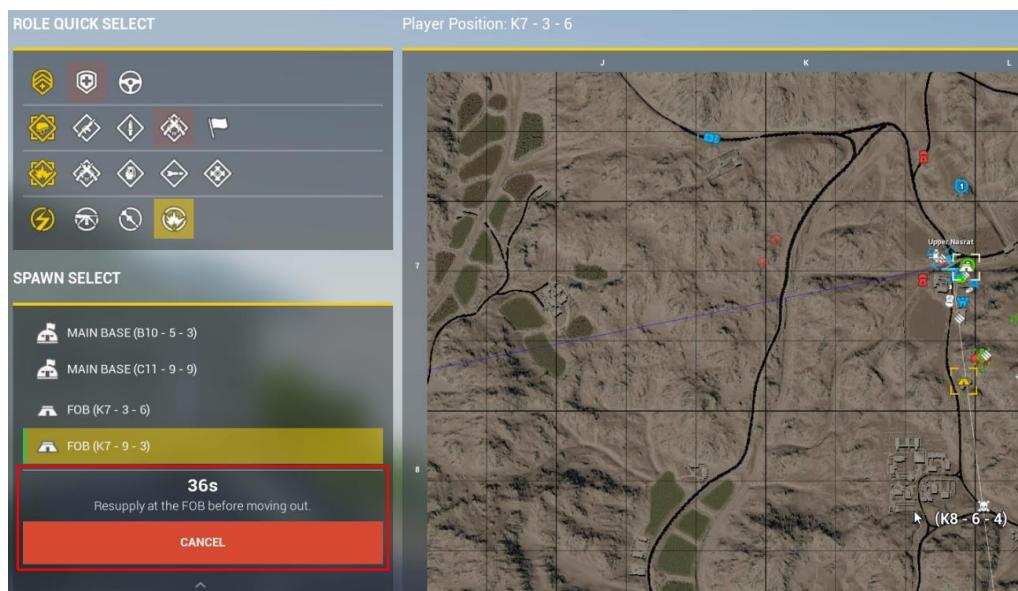


Slika 36 AI u igri

### 3.4.4.2. Preuzimanje kontrole nad umjetnom inteligencijom

Umjetna inteligencija nalazi se u igri samo za poboljšanje njenog tijeka, ne za povećanje interakcije direktno, jer interakcija sa AI ima puno manje značenje, za prave interakciju su odgovorni drugi igrači. Tijek igre se poboljšava preuzimanjem kontrole nad AI od strane igrača.

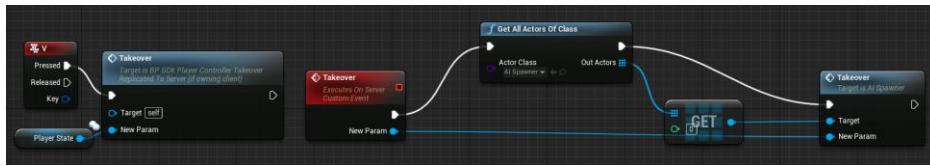
U većini drugih igara smrt znači čekanje za ponovo stvaranje, ovo može trajati od 30+ sekundi u Squad, do sekunde u Call of Duty. Svo ovo vrijeme je zapravo potrošeno na beskorisno čekanje, ljudi žele igrati igru, ne buljiti u ekran za ponovo stvaranje (eng. Respawn screen). U našoj igri nema mrtvog vremena, odmah se možemo vratiti u igru preuzimanjem kontrole nad AI.



Slika 37 Squad dugo vrijeme za ponovno stvaranje, vlastita izrada

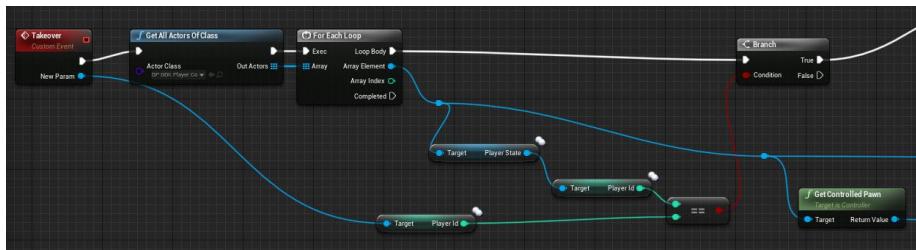
Ovo ćemo prvobitno postići dizajnom našeg sloja, možemo primijetiti da smo na sloj stavili samo upravljača AI, ne i njegovog lika. Važno je da lik bude na istom sloju kao i upravljač klijenta, jer tako izbjegavamo nepotrebne interne komunikacije između slojeva, makar su one zapravo vrlo sigurne i gotovo se odmah događaju [79], bolje je taj rizik staviti na stranu upravljača AI, nego na igrača, jer nam je njegovo iskustvo očito važnije.

Sam proces opet započinje od strane korisnika, koji pritiskom na tipku „V“ šalje RPC na server, gdje se ponovo traži odgovarajući stvaratelj AI te se aktivira njegova funkcija za preuzimanje kontrole nad AI.



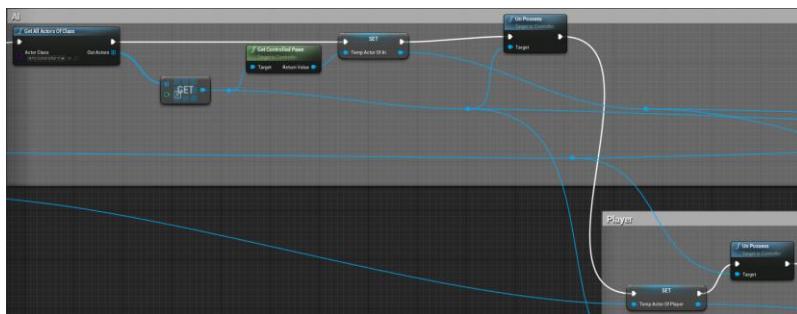
Slika 38 Klijent zahtijeva preuzimanje nad AI

U stvaratelju ponovo pronalazimo odgovarajućeg igrača koji je zatražio funkciju.



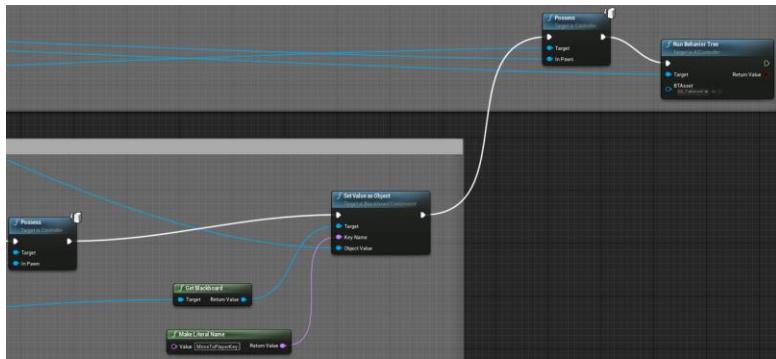
Slika 39 Pronalaženje klijenta u stvaratelju

Kada smo ga pronašli, pronalazimo prvi dostupnog lika AI-a, njegovom upravljaču mičemo kontrolu, te isto radimo kod igrača.



Slika 40 Pronalaženje AI u stvaratelju

Naposljeku samo zamijenimo upravljače, dakle igrač sada upravlja likom od AI, dok AI upravlja likom od igrača.



Slika 41 Zamjena upravljača

Na sljedećoj slici vidljivo je da je preuzeta kontrola nad AI, te da je lik klijenta (crveni) nad kontrolom AI.



Slika 42 Preuzeta kontrola nad AI

### **3.4.4.3. Buduće iteracije AI**

U sljedećim iteracijama AI bi trebalo unaprijediti sa drugim naredbama, osim praćenja igrača, te ih organizirati u odrede (eng. Squad). To bi bile komande za:

- Kreni do označene pozicije
- Zaustavi se
- Brani ovu poziciju
- Napadni ovu poziciju
- Sagradi
- Povuci se
- Regrutiraj (Ponovo stvori mrtve članove odreda)

Neke od ovih naredbi su se u prototipu izvele pomoću drva odluka, ali to je naizgled kompleksan način izvedbe, zbog ovoga se odustalo od ovog pristupa.

Mogući načini izvedbe su preko strojeva stanja, radi stabilnijeg ponašanja i lakšeg programiranja.

Nadalje moguće je navedeno izvesti pomoću hijerarhijske mreže zadataka (eng. Hierarchical Task Network) [80], sa još dubljim ponašanjima.

Naposljetku, ovismo o dizajnu igre, moguće je klijentu otvoriti modifikaciju ponašanja, preko sustava sličnom onome u Dragon Age Origins, zvanom napredni taktički sustav (eng. Advance Tactics System) [81] gdje on zapravo programira kako se ponašanje AI odvija u igri.

## 3.5. Umrežene značajke

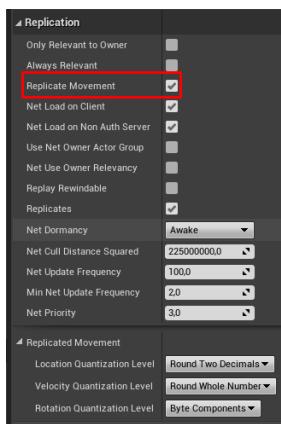
U ovom poglavlju ćemo implementirati umrežene značajke koje su kritične za dinamičan svijet, prikazati će se iteracije kroz koje smo prošli te dati prijedlog za buduće iteracije.

### 3.5.1. Vozila

Vozila su veoma izazovan slučaj umrežene fizike, te se zato obrađuju u ovom radu. Srž izazova je u osjećaju trenutne kontrole s obzirom na kašnjenje serverske obrade. Kroz iteracije će se prikazati nekoliko metoda za obradu ovog problema.

#### 3.5.1.1. Prva iteracija – Replikacija kretnje

Prva iteracija je obična replikacija kretnje koju UE4 koristi kao podlogu za sve fizičke objekte. Ona se uključuje preko izbornika replikacije preko prekidača za replikaciju kretnje (eng. Replicate Movement). Ovo je jedini izvorno dostupan način izvođenja vozila u UE4.

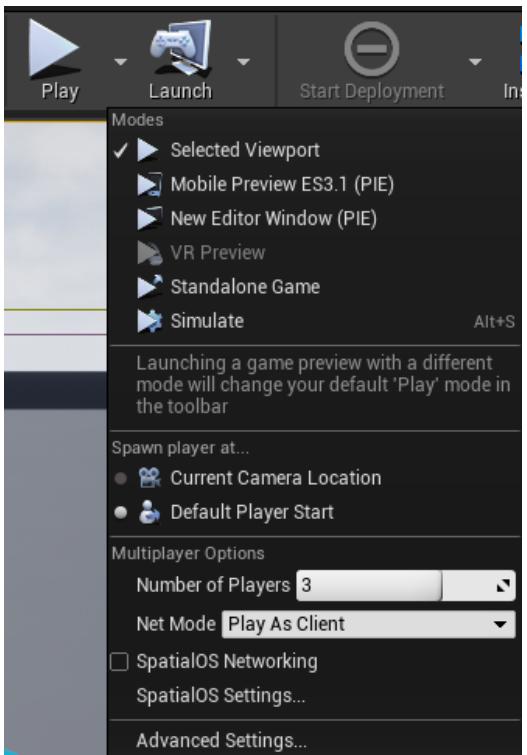


Slika 43 Replikacija kretnje u UE4

Ovo rješenje je jednostavno nezadovoljavajuće, zbog jednog razloga: manjka trenutne kontrole. Tijekom vožnje odmah primjećujemo da su kontrole ukočene, stalno nas se „vraća“ na prijašnje pozicije i slično.

Autori ekstenzije OWI Enhanced Vehicle Movement, koja je besplatno dostupna na Epic Games Store, predlažu modificiranje replikacijskih postavki projekta, ali one samo skrivaju problem, ne tretiraju ga [82].

Sami možete testirati ovu iteraciju navigacijom do „Content/OWIContent/Shared/Maps/DesertLandscape/Maps/DesertLandscape\_v1“ sa 2 igrača, bez Spatial OS networking, te igranje kao klijent.



Slika 44 Konfiguracija za pokretanje servera

Da bi simulirali kašnjenje paketa, kao u pravom okruženju, trebamo otvoriti konzolu pritiskom na gumb tilda (`), te unošenjem komande „NetEmulation.PktLag 100” za simulacije odaziva od 100 milisekundi.



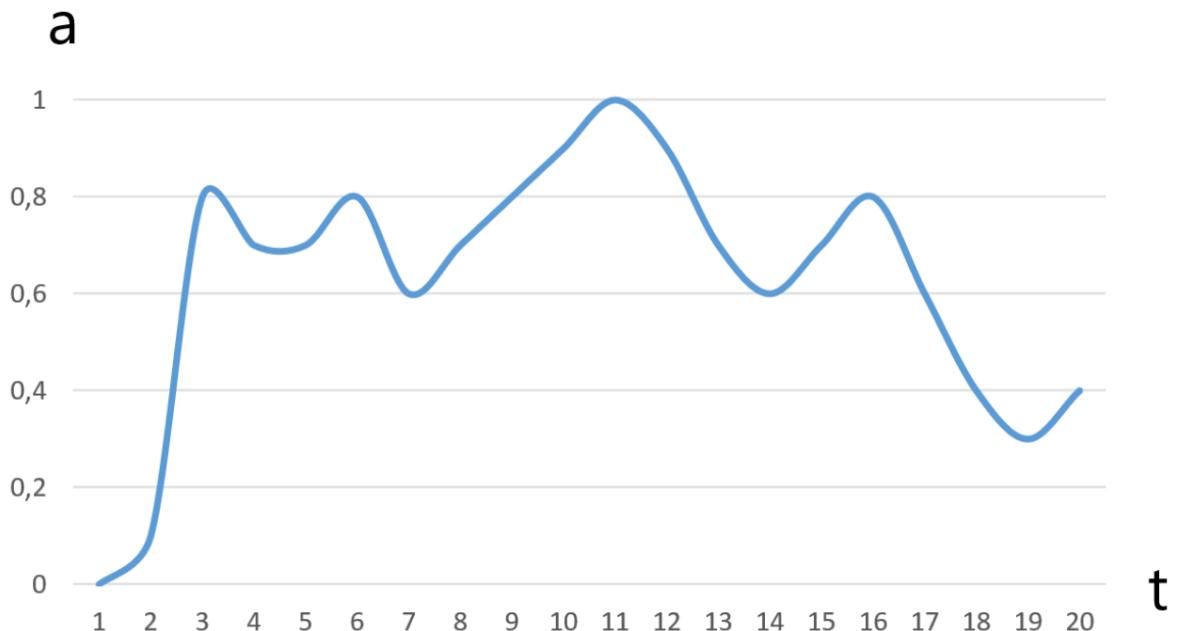
Slika 45 OWI Enhanced Vehicle Movement u igri

Da bi smo shvatili zašto se ovo događa, moramo pobliže promotriti funkciju samih vozila. Vozila se pomiču zbog svoje komponente za kretnju – klasa „WheeledVehicleMovementComponent“, odnosno zbog unosa koje ona prima. Unosi su dakle skretanje, gas (naprijed), kočenje, ručna kočnica te trenutna brzina. Ovi unosi rezultiraju povećanjem akceleracije vozila, te tako i njegovom kretnjom, odnosno promjenom pozicije. Replikaciju ovih varijabli možemo vidjeti i u funkciji „ServerUpdateState“.

```
605  |  /** Pass current state to server */
606  |  UFUNCTION(reliable, server, WithValidation)
607  |  void ServerUpdateState(float InSteeringInput, float InThrottleInput, float InBrakeInput, float InHandbrakeInput, int32 CurrentGear);
```

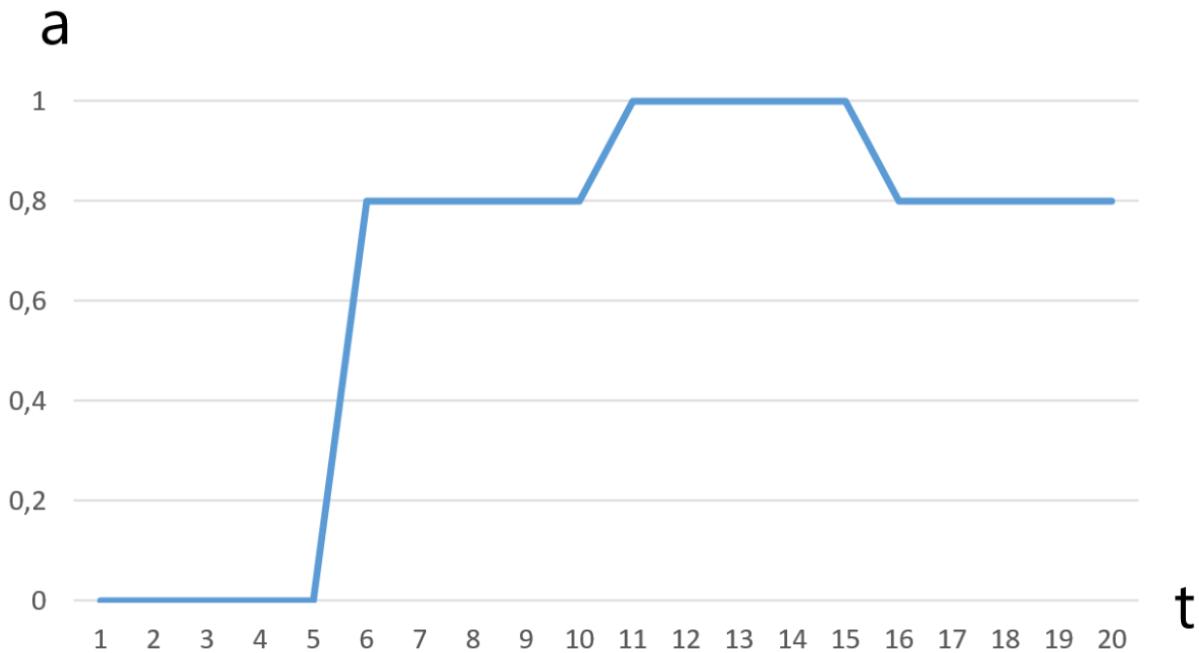
Slika 46 Funkcija za slanje unosa

Tu dolazimo do problema, zapamtimo, radimo u umreženom okruženju, posljedica toga je to što se replikacija klijentskih vrijednosti na server događa samo 20 puta u sekundi. Na temelju tih vrijednosti na serveru se izvodi simulacija. Na klijentskoj strani se simulacija također izvodi, ali sa trenutnim unosima, koji se mogu promjeniti bilo kada, ne samo pri njihovoj replikaciji. Pogledajmo sada jedan primjer ove posljedice, napravljen prema Sam Pattuzzi [83]. Dakle na sljedećoj slici vidljiv je rezultat raznih unosa na klijentskoj strani, odnosno akceleracija.



Slika 47 Rezultat unosa na klijentskoj strani, vlastita izrada

Pretpostavimo sada da server dobiva unose svakih 5 koraka. Rezultat je potpuno drugačiji izračun akceleracije jer server mora prepostavljati da je akceleracija konstanta između ažuriranja.



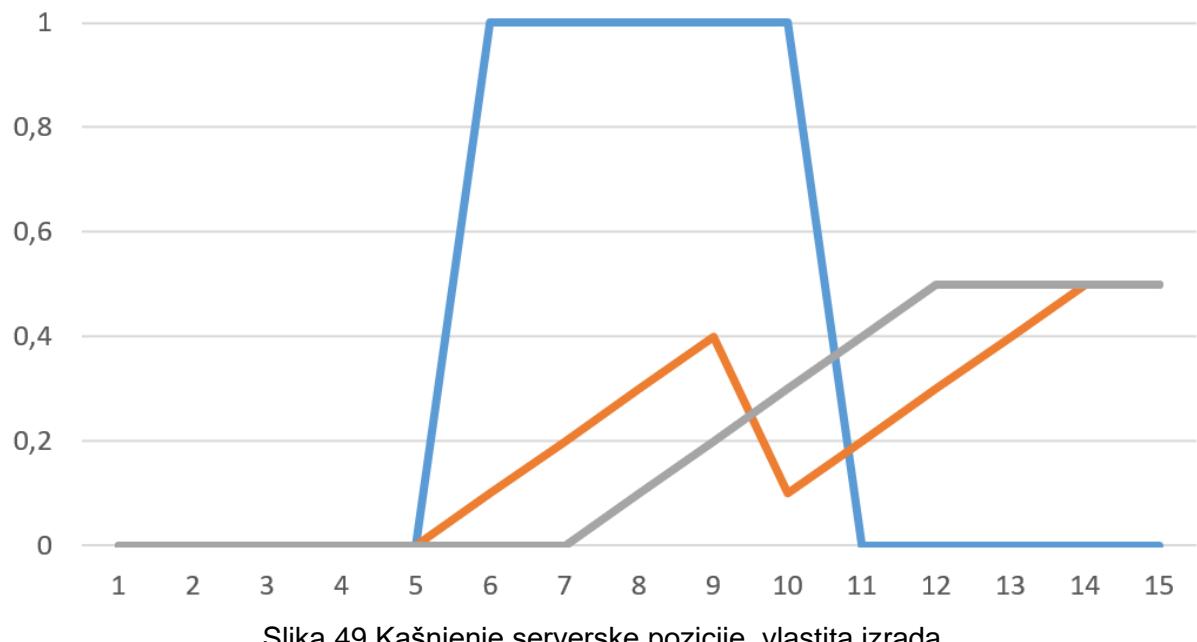
Slika 48 Rezultat unosa na serverskoj strani, vlastita izrada

Rezultat ovoga postupka je drugačija pozicija na serveru, a pošto server mora biti autoritativan, njegova pozicija se uzima kao pravilna.

Nadalje, ako bi riješili prije naveden problem, još uvijek dolazimo do problema sa sinkronizacijom pozicija servera. Dakle, pošto serverska pozicija klijentu kasni proporcionalno njegovom punom odazivu, klijenta će se uvijek „vraćati“ unazad, rezultat je ne trenutna kontrola.

Na sljedećoj slici možemo vidjeti zašto se ovo događa.

- Plava linija predstavlja akceleraciju
- Narančasta linija predstavlja klijentsku poziciju na X osi
- Siva linija predstavlja serversku poziciju na X osi



Dogodilo se sljedeće:

1. Na klijentskoj strani se u trenutku 5 po pritisku gumba naprijed povećava akceleracija
2. U 7. trenutku server prima replikaciju unosa te izvodi svoju simulaciju, te replicira novu poziciju klijentu
3. U 9. trenutku klijent prima pozicijsko ažuriranje te ga lokalno modificira, što rezultira vraćanjem unatrag na staru poziciju

### 3.5.1.2. Druga iteracija - INetworkPredictionInterface

Druga iteracija je ručno izrađena implementacija sučelja za umreženje INetworkPredictionInterface nad PhysX vozilima. Ovo je sučelje namijenjeno rješavanju prije navedenih problema. Implementacija radi na bazi umrežene uloge [84]. Lik može imati tri uloge ovisno o tome tko ga kontrolira i gdje, uloge su prikazane u sljedećoj tablici.

Tablica 6 Opis umreženih uloga [84]

Umrežena uloga	Opis
Autonomni zastupnik (eng. Autonomuos Proxy)	Lika kontrolira klijent na klijentskom računalu.
Autoritet (eng. Authority)	Lik postoji na serveru.
Simulirani zastupnik (eng. Simulated Proxy)	Lik postoji na klijentskom računalu, ali kontrolira ga se na daljinu.

Dakle na svakom od ta tri lika odvija se zasebni proces, u sljedećih nekoliko tablica opisati će se taj proces.

Tablica 7 Opis koraka na zasebnim likovima [84]

Korak	Opis
Autonomni zastupnik	
1	Klijent kontrolira autonomnog zastupnika lokalno. Funkcija „PerformMovement“ pokreće fizičku logiku kretanja iz komponente za kretnju.
2	Sastavljamo spremljene korake „FSavedMove_Character“ koji sadrže podatke o tome kako smo se upravo pomaknuli, te ih sprema u „SavedMoves“.
3	Slični spremljeni pokreti se kombiniraju te šalju na server sa „ServerMove“ RPC.
Autoritet	
4	Server prima „ServerMove“ i ponavlja pokret sa „PerformMovement“.
5	Server provjerava da li se njegova pozicija podudara sa klijentskom.
6	Ako se pozicije podudaraju, šalje se signal da je pokret ispravan. Ako se ne podudaraju, pokret se ispravlja sa „ClientAdjustPosition“ RPC.
7	Server šalje svoju lokaciju, rotaciju i trenutno stanje simuliranog zastupnika ostalim klijentima preko „ReplicatedMovement“ strukture.
Autonomni zastupnik	

8	Ako smo zaprimili ClientAdjustPosition, reproduciramo serverske pokrete i koristimo njegovu „SavedMoves“ listu da bi ponovo prošli njegove korake do konačne pozicije. Kada se pokreti riješe njih se miče iz liste.
Simulirani zastupnik	
9	Primjenjuje se replicirani lokacijski podaci, koristi se zaglađivanje (eng. Smoothing) za vizualnu reprezentaciju kretnje.

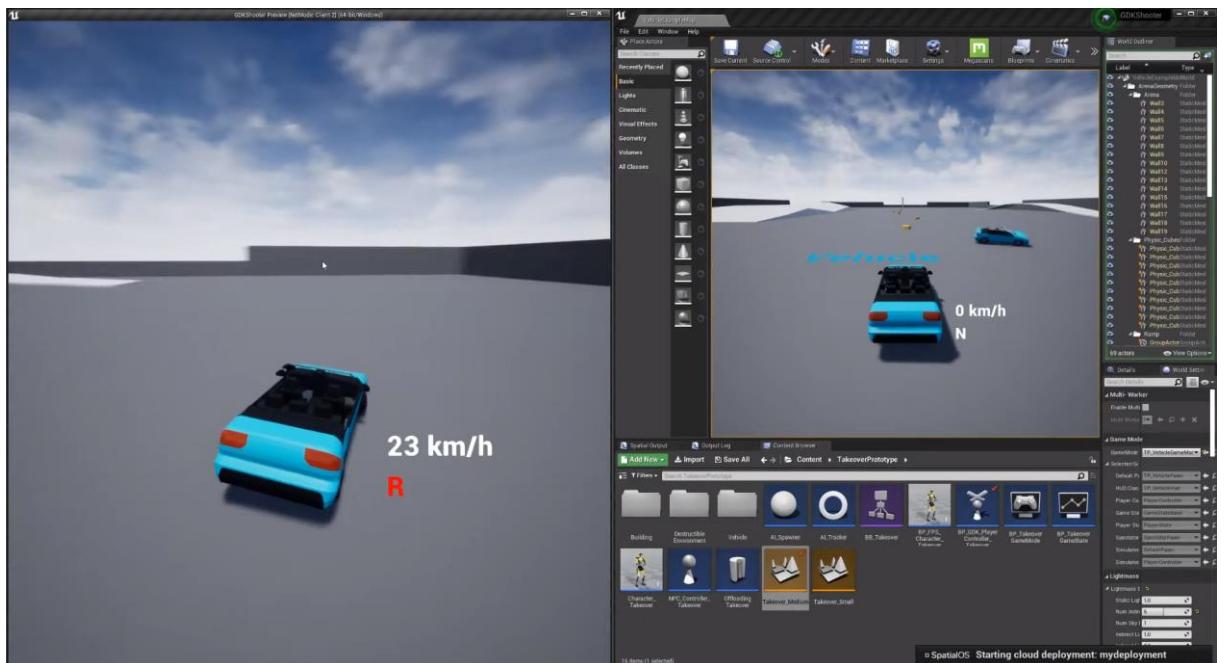
Rezultat, za likove, je trenutna kontrola sa serverski autorativnom pozicijom.

Kada primijenimo ovaj princip na PhysX vozila, odnosno na njegovu komponentu za kretnju, dolazimo do prividno dobrog rješenja, vozilo vizualno izgleda kao da se pravilno kreće bez prijašnjih artefakata.

Rezultat je vidljiv u klasi „C++

Classes/GDKShooter/NetPhysVehicleMovementComponent.cpp“. Implementacija inspirirana od strane korisnika „MazyModz“ [85].

Problem do kojeg dolazimo je ne trenutna kontrola, dakle zapravo se naš unos detektira tek nakon što ga je server potvrdio. Rezultat je vidljiv na navedenom video zapisu [86].



Slika 50 INetworkPredictionInterface u igri

Sami možete testirati kašnjenje pokretanjem svijeta u Content/VehicleCPP/VehicleMap\_V2 sa 2 igrača, bez Spatial OS networking, te igranje kao klijent. Da bi simulirali kašnjenje paketa, kao u pravom okruženju, trebamo otvoriti konzolu pritiskom na gumb tilda (`), te unošenjem komande „NetEmulation.PktLag 100“ za simulacije odaziva od 100 milisekundi.

Srž problema je u izvedbi simulacije na klijentu, jer zapravo sa PhysX ne možemo simulirati pokrete unaprijed ili unatrag, to je odvojeni proces koji se izvodi nakon otkucaja dretve igre.

Ovaj problem iskazan je i u temeljnoj dokumentaciji za Unreal Engine:

„Neka klijenti predviđaju ponašanje glumaca u vlasništvu klijenta na temelju unosa igrača; simulirajte ovo ponašanje prije nego što dobijete potvrdu od poslužitelja (i ispravite ako je potrebno). Ovaj model koristimo za kretanje piona i rukovanje oružjem, ali ne i za vozila, jer složenost spremanja i ponovnog prikazivanja fizičke simulacije nadilazi korist smanjenog kašnjenja za upravljanje vozilima, gdje se tipična kašnjenja internetskog odgovora ne razlikuju od tipičnog stvarnog svijeta kašnjenja u odgovoru upravljanja vozilom“ [87].

Zapravo smo implementirali nešto bolju metodu od replikacije kretnje, iskazanu u dokumentaciji za umreženo upravljanje fizičkim tijelima:

„Za vozila (PHYS RigidBody Actors), postoji sljedeći umreženi tijek:

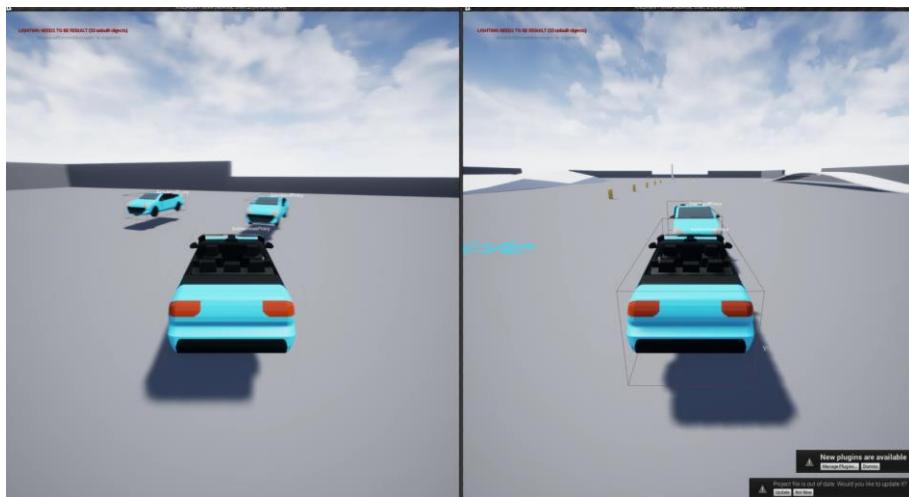
1. Pritisni tipku na klijentu
2. Pošalji unose (gas, skretanje) serveru – pozivom replikacijske funkcije ServerDrive
3. Generiraj izlaz (OutputBreak, OutputGas itd.); zapakiraj u repliciranu strukturu koja se može poslati klijentu – ProcessCarInput() se zove na serveru
4. Ažuriraj vozilo na serveru i klijentu, koristi izlaz (OutputBreak, OutputGas itd.) da bi primijenio sile/zakrete na kotače/vozilo – UpdateVehicle() se poziva na klijentu i serveru“ [88].

Treba napomenuti da je sa PhysX moguće ovo ostvariti koristeći Immediate mode kako bi importirali fizičku scenu te u nju uključili sve relevantne objekte [89]. Ovo je povoljna podloga za sljedeću iteraciju, ali zbog kompleksnosti izvedbe te opširnosti problema nije izведен.

### 3.5.1.3. Treća iteracija – Ručno izrađena fizika

Da bi smo demonstrirali da je ovo moguće izradili smo svoj model vozila, inspiriran izvedbom Sam Pattuzzi [83] koji koristi pojednostavljeni algoritam ali u srži identičan onome u Character Movement Component.

Temeljna razlika je da imamo svoj ručno izrađen fizički sustav koji obrađuje kretnju, te ga možemo pozivati neovisno o dretvi igre. Ovaj projekt dostupan je na sljedećoj poveznici [90], treba spomenuti da je minimalno modifcirana za potrebe pokretanja u novijim verzijama UE4.



Slika 51 Ručno izrađena fizika u igri

Rezultat je trenutna kontrola sa serverski autoritativnom pozicijom. Treba napomenuti da se nisu implementirala daljnje simulacije hidraulike i slično iz PhysX modela, te nije izrađena detekcija rotacije.

Sami možete testirati kašnjenje pokretanjem projekta sa 2 igrača, bez Spatial OS networking, te igranje kao klijent. Da bi simulirali kašnjenje paketa, kao u pravom okruženju, trebamo otvoriti konzolu pritiskom na gumb tilda (`), te unošenjem komande „NetEmulation.PktLag 100“ za simulacije odaziva od 100 milisekundi. U ovoj implementaciji možemo vidjeti da se bez obzira na veličinu odaziva održava trenutna kontrola.

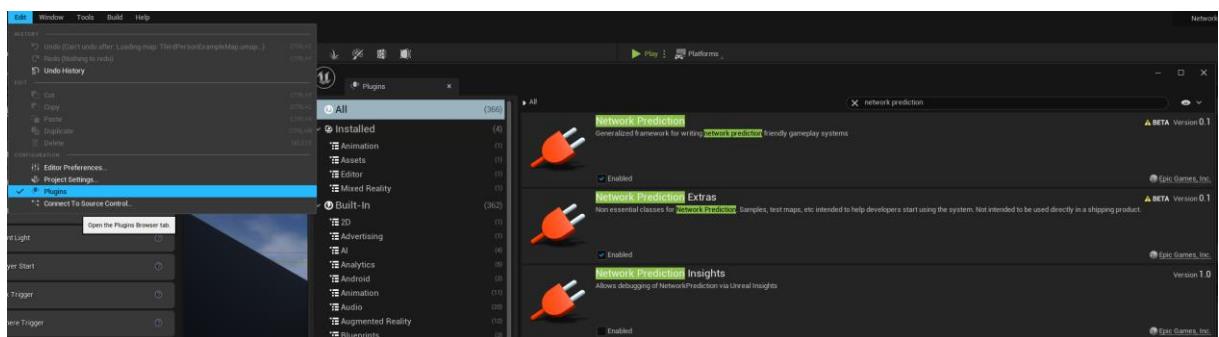
Inače postoje već izvedeni i spremni modeli koji implementiraju istu metodiku u ručno izrađenom fizičkom sustavu [91].

### 3.5.1.4. Četvrta iteracija – Chaos fizika

Gledajući u budućnost Chaos fizički sustav koristiti će sučelje na njegovoj razini, jednostavno nazvano „Network Prediction“, to znači da se svaki fizički objekt može trenutno ne samo kontrolirati nego i simulirati, odnosno predvidjeti, bez potrebe za čekanjem serverske pozicije, ali sa serverskim autoritetom [92].

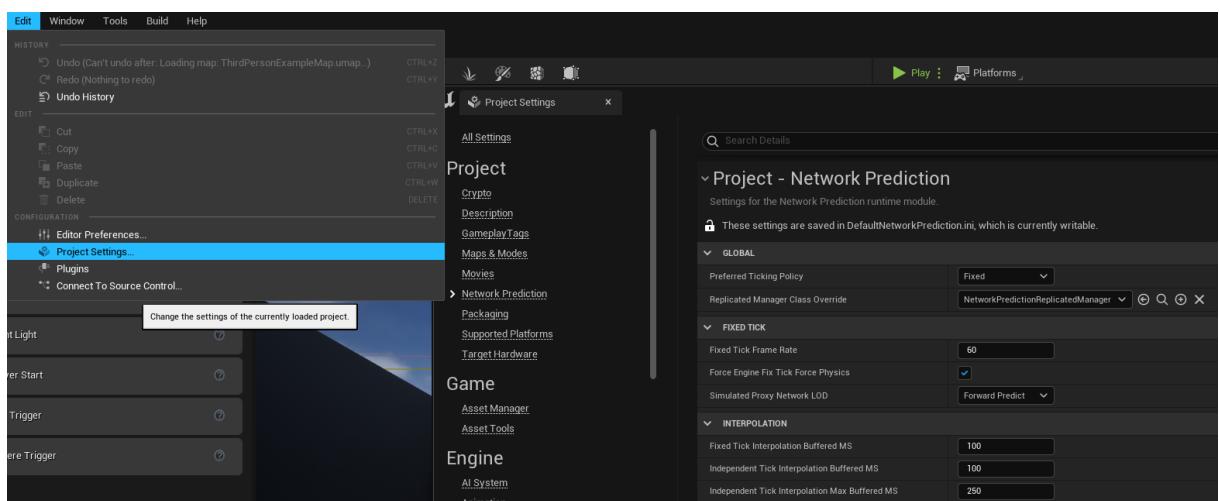
Dakle radi se o determinističkom sustavu koji samo prima unose drugih korisnika.

Njega se može pokrenuti skidanjem UE5 [93], te aktiviranjem proširenja „Network Prediction“ te „Network Prediction Extensions“.



Slika 52 Aktiviranje proširenja za predikciju

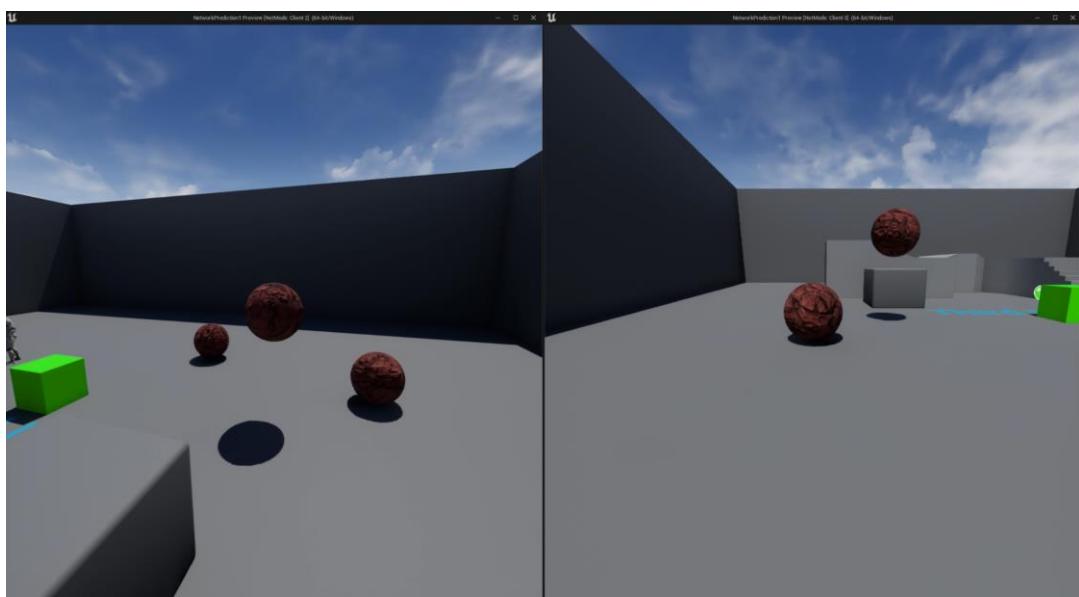
Nadalje potrebno je konfigurirati proširenje, to radimo u opcijama projekta, potrebno je aktivirati „Force Engine Fix Tick Force Physics“, to osigurava da se fizički sustav provodi u diskretnim koracima, odnosno 60 puta u sekundi.



Slika 53 Konfiguracija sustava za simulaciju fizike

Sada možemo u svoj svijet ugraditi navedeno sučelje, nasreću u dodatku proširenja imamo već pripremljene implementacije, samo je potrebno definirati zadanog lika (eng. Default Pawn) na „NetworkPredictionExtras\_ControllablePhysicsBall“. Treba spomenuti da se u C++ klasama nalazi još mapa sa drugim primjerima koji su dostupni na putanji „NetworkPredictionExtras Content/Maps“.

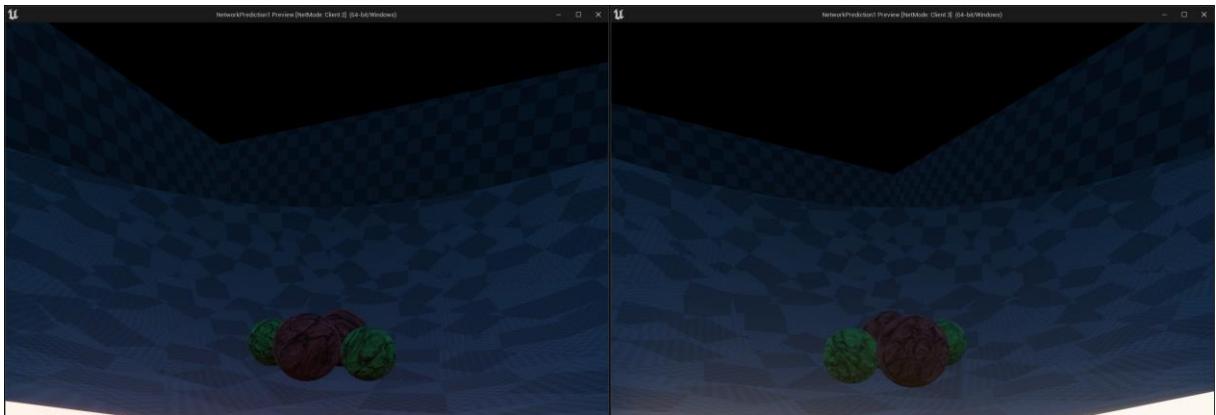
Ova fizička kugla je zapravo vrlo dobra reprezentacija vozila, dakle daje se neki unos, koji proizvodi neku silu, praktički je jedino potrebno implementirati sučelje koje samo na zadan način šalje te unose te se brine za premotavanje i slično. Rezultat je vidljiv sljedećem navodu [94].



Slika 54 Chaos fizika u igri [94]

Uključivanjem simulacije kašnjenja paketa, možemo vidjeti da se još uvijek održava trenutna kontrola, kada dobijemo zakašnjeli unos cijela simulacija se premotava, taj unos se simulira, te svi unosi koji su lokalno bili nakon njega. Rezultat je trenutna kontrola i trenutna slika svijeta, bez obzira na zakašnjenje veze.

Ovo je dobara podloga za testiranje determinističkog sustava, naime pokretanjem mape „TestMap\_PhysicsControllable\_Single“ vidjeti da zelene kugle na oba klijenta imaju iste pozicije. Treba zamijetiti da se pozicije zelenih kugli nikada nisu replicirale, samo unosi korisnika koji kontroliraju smeđe kugle.



Slika 55 Deterministička fizika u igri, vlastita izrada

Ako bi željeli promatrati proces premotavanja simulacije možemo koristiti dodatak „Network Prediction Insight“ koji daje pregled simulacije po otkucaju [95].

### **3.5.2.Umreženje simulacije fizike objekata**

Nekad bi željeli simulirati preciznu fizičku kretnju i rotaciju ogromnih objekata, koji mogu utjecati na tijek igre, poput padajućih zgrada koje mogu prekrivati veliku površinu mape i slično. Cilj je zapravo stabilan fizički objekt u umreženom okruženju. Također, da bi se spriječila ikakva moguća manipulacija objekata, ova simulacija se mora voditi na serverskoj strani.

#### **3.5.2.1. Algoritam**

Ovaj problem može se podijeliti u dvije faze, prva je determiniranje pozicije fizički simuliranog objekta, drugi je buđenje tog objekta da obavi simulaciju samo onda kada je to potrebno.

Algoritam za provođenje simulacije fizike je sljedeći; na serverskoj strani provodi simulacija te kada se dođe do finalne lokacije koja je barem 1 sekundu identična onoj prijašnjoj, ona se završava, jer se pretpostavlja da je objekt stao.

Ovaj pristup je potreban zbog ograničenja sinkronizacije stanja u UE4, jer bi se inače slale vrlo male promjene u poziciji objekta ( zbog kolizije ), što bi rezultiralo bespotrebnim slanjem paketa, dok simulacija ne bi bila znatno preciznija. Također je potreban jer bez njega dolazi do degradacije iskustva, jer se pojavljuje konstantni „jitter“ (malene kretnje objekta) zbog raznih faktora, kao osjetljivi algoritam za predviđanje sa korisničke strane, te nepreciznost float-a i prenesenog quanta, broj ciklusa provjere kolizije mora biti malen da bi se uštedjelo na cpu ciklusima. Iznimno je važno da se jitter riješi, jer će klijenti konstantno biti u interakciji sa ovim objektima ( hodati po njima ). Moguće je izbjegći ovaj problem povećanjem detaljnosti simulacije i substeppingom (metoda za provođenje simulacije više puta u jednom ciklusu), ali pošto simuliramo više tisuća fizičkih objekata ne možemo si to priuštiti.

#### **3.5.2.2. Prva iteracija – kucanje srca**

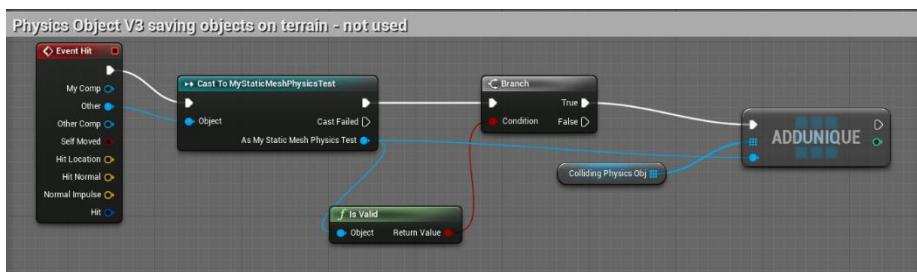
Prvi pokušaj – provodimo simulaciju, te ju ugasimo, svaku sekundu provodimo tehniku „kucanja srca“ (eng. Heartbeat) koji uključi fiziku kako ne bi plutali po zraku. Makar je prvidno rezultat precizan na svakom klijentu, još se uvjek razlikuje od servera, jer se izračuni kolizije uvek mijenjaju bez obzira na to što je objekt zapravo u mirovanju, te dolazi do bespotrebnog slanja paketa.

### 3.5.2.3. Druga iteracija – kanal za buđenje

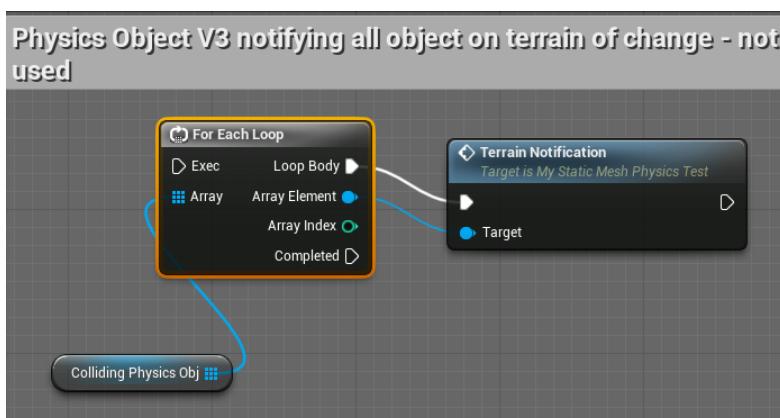
Sljedeći pokušaj sastojao se od primjene koncepta „Net dormancy“ iliti kanala za buđenje. Ukratko, prije nego što provedemo simulaciju probudimo kanal, što će poslati obavijest o rezultatu, nakon toga ugasimo kanal, što efektivno determinira rezultat simulacije. Problem ovog pokušaja sastojao se u buđenju, jer smo budili simulaciju na svaku novu koliziju, što ponovo rezultira buđenjem i provođenjem simulacije, te napoljetku i „jitterom“.

### 3.5.2.4. Treća iteracija – paljenje i gašenje simulacije

Treća iteracija sastoji se od provođenja simulacije, te kada smo determinirali zadovoljavajuću poziciju, gasimo simulaciju, da se pozicija više ne može mijenjati na serveru. Problem u ovom pristupu leži u buđenju objekta, u smislu, ako se on nalazi na dinamičkom terenu, koji se stalno pomiče, može doći do lebdenja objekta koji bi u tom trenutku zapravo trebao simulirati gravitaciju, ali mu je simulacija ugašena. Rješenje se nalazi u simulaciji buđenja, tako da pri dinamičkom terenu prikupljamo podatke o svim objektima koji se na njemu nalaze ( te koji imaju uključenu fiziku ), te nad njima ponovo pokrećemo postupak simulacije ako se teren promjenio.



Slika 56 Spremanje objekata koji se nalaze na terenu



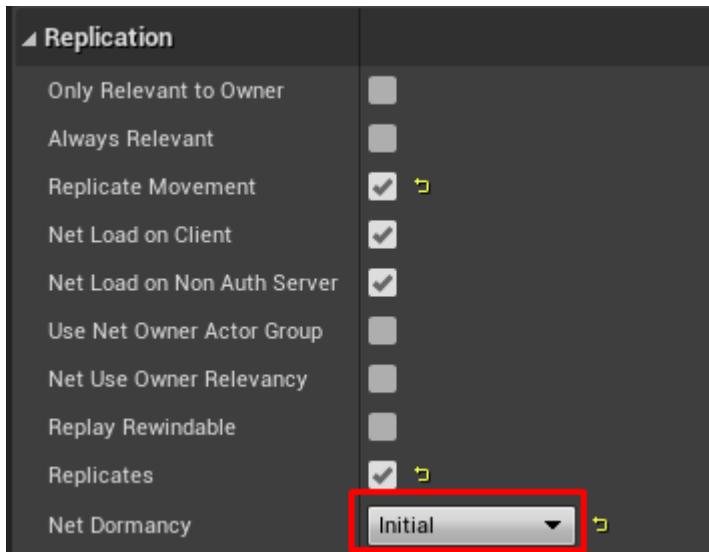
Slika 57 Notifikacija svih objekata na terenu o promjeni

Pri ovom rješenju se nikada ne događa degradacija iskustva, te je u velikoj većini slučajeva precizno i zbog toga jer su ovi objekti rijetki. Jedan rubni slučaj gdje bi moglo doći do lebdećih objekata je ako objekt padne, ali nikad ne dotakne teren, te tako nikad nije prepoznat, ali ovo je moguće riješiti sličnim praćenjem – koji objekti se nalaze na objektu (koji dodiruje tlo), makar, zbog rijetkosti ove pojave, te mogućih bespotrebnih izračuna neće se implementirati.

### 3.5.2.5. Finalna implementacija – razlika u poziciji

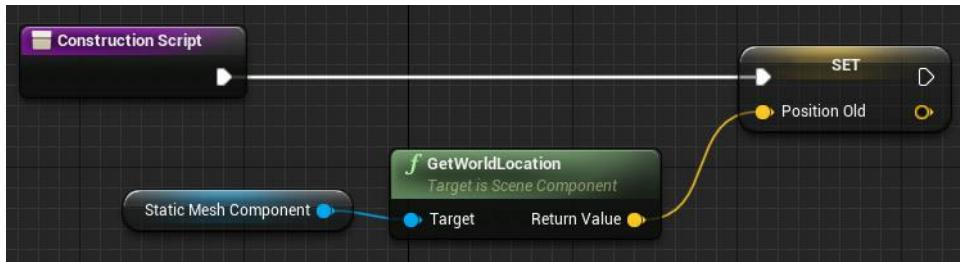
Nakon ovih pokušaja odlučeno je pogledati problem sa novog stajališta, jer očito je da prijašnja rješenja nisu skalabilna te bi moglo doći do curenja memorije.

Finalna iteracija je dakle sljedeća; fizički objekt se postavi kao inicijalno mrežno uspavan, dakle ne šalju se njegove promjene svim klijentima.

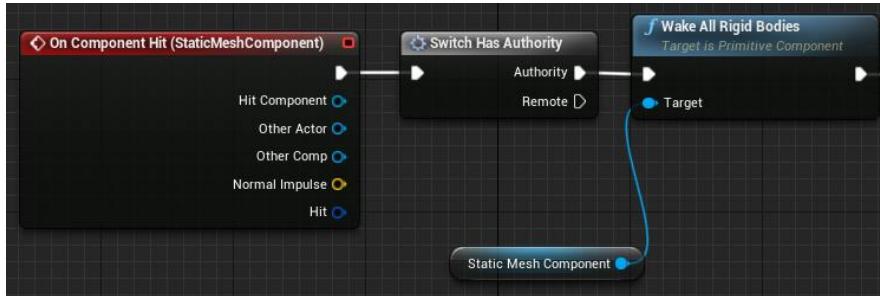


Slika 58 Objekt postavljen kao inicijalno mrežno uspavan

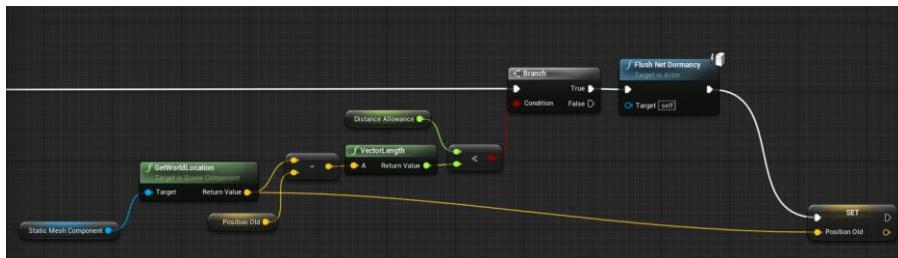
Kada se objekt konstruira, pamti mu se trenutna pozicija, kada mu se u tijeku igre dogodi bilo kakva kolizija, to tijelo se ispituje da li je razlika njegove pozicije i stare pozicije veća od one dopuštene (1 cm), ako je onda se mrežni kanal budi i šalje se promjena.



Slika 59 Konstruiranje objekta



Slika 60 Kolizija koja pokreće funkciju



Slika 61 Buđenje mrežnog kanala

Rezultat je savršeno sinkronizirana pozicija objekata, bez jittera te je iznimno jeftina na serverskoj strani. Rezultat je vidljiv na putanji „Content/ TakeoverPrototype/ DestructibleEnvironment/ StaticMeshPhysics/ MyStaticMeshPhysicsTestV3“ te u sljedećem navodu [96].

### 3.5.2.6. Buduće iteracije – zaključavanje Z osi

Moguće je nadalje optimizirati neke objekte za koje nam nije potrebna puna fizika, na primjer ako bi željeli da objekti padaju samo po Z osi, možemo ih ograničiti sa stezanjima te nekolicinom multi-poziva (pošto stezanja nisu replicirana). Ovaj efekt koriste igre poput Valheim da bi izbjegle pre kaotična fizička ponašanja, te repliciranje manje podataka.

### 3.5.3. Deformacija terena

Deformacija terena je ključna značajka za dinamičnost svijeta, daje dojam da okružje može reagirati na svaku pojavu. U klasičnim umreženim igrama teren ostaje statičan, niti najjače oružje ga ne može deformirati.

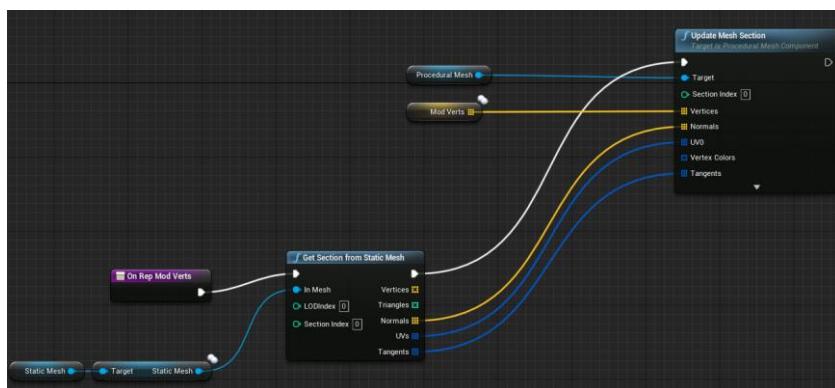
Ideja je da prilikom pogotka projektila jednostavno deformiramo teren oko mesta pogotka, i to proporcionalno šteti koju bi on učinio.

Treba spomenuti da je ipak izvedeno nekoliko puta u umreženim igrama koje su za to napravile budžet, primarno Battlefield serija te ArmA serija igara.

#### 3.5.3.1. Tehnologija i algoritam

Za izvođenje deformacije terena koristiti će se tehnologija proceduralnih oblika (eng. procedural mesh) [97], to je skup vrhova i bridova koji se mogu modificirati tijekom izvođenja. Nadalje koristi se algoritam za njihovo modificiranje, baziran na algoritmu koji je definirao korisnik „CodingWithRus“ [98], te se ovo poglavlje više bavi problemom umreženja.

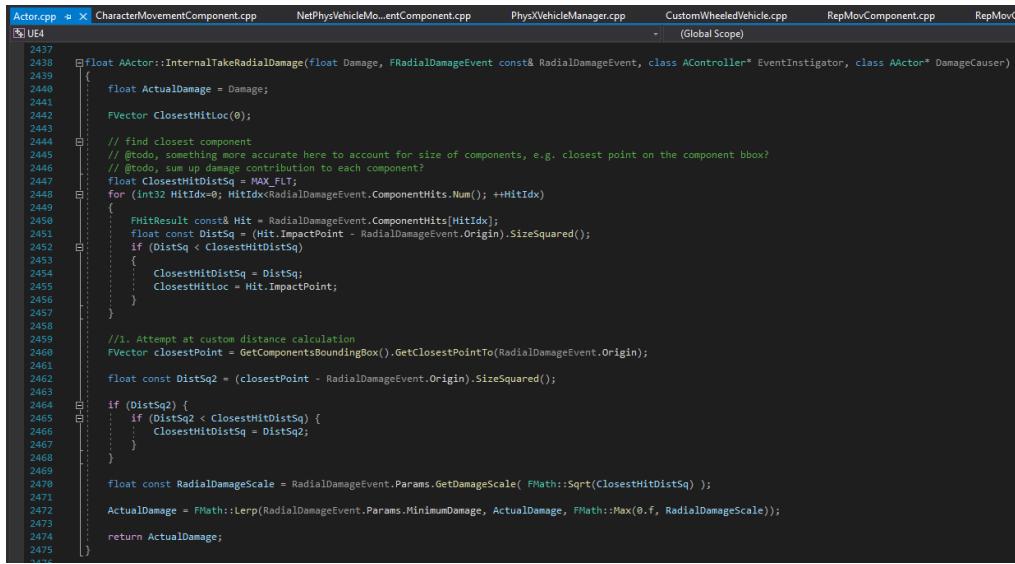
Algoritam započinje detekcijom događaja – primanja štete, zabilježimo lokaciju i količinu štete te pokrećemo modifikaciju oblika s obzirom na te parametre. Modifikacija oblika se provodi tako da prolazimo kroz svaki vrh terena, te ako je njegova udaljenost od mesta pogotka unutar radiusa (definiran kao varijabla terena koja zapravo predstavlja njegovu gustoću) tada nad njim primjenjujemo silu. Sila se izračunava kao odnos deformacijske snage (koliko multiplicirati štetu nad terenom) i udaljenosti od mesta pogotka. Ova sila se primjenjuje u smjeru Z osi, te naravno prema dolje, te se izglađuje sa varijablom „Smoothing“ za manje nazubljene vrhove. Kada smo tako prošli kroz svaki vrh, promjene se spremaju, te se tako modificira izgled terena i dolazi do replikacije klijentima (preko „On Rep Mod Verts“ metode koja se poziva svakom modifikacijom istoimenog polja vrhova), koji istu modifikaciju samo primjene na svoj lokalni teren, što rezultira sinkronizacijom.



Slika 62 On Rep Mod Verts metode koja se izvodi na klijentskoj strani

Treba napomenuti da je modificiran način primanja radikalne štete na UE4 razini.

Dakle, u postojećem sustavu projektila, pri pogotku, oštećenje se zadaje u centru objekta, što za naše potrebe nije dovoljno, jer bi htjeli da teren prima štetu točno na onim točkama gdje je i pogoden. Zbog toga implementiran je sustav koji izračunava točnu lokaciju pogotka na objektu, odnosno pronalazi točku na liku koja je najbliža točci pogotka. Ovo je i preporučeno od strane UE4 u komentarima same klase lika (Actor.cpp), te je i izvedeno:



```
2437
2438     float ActualDamage = Damage;
2439
2440     FVector ClosestHitLoc(0);
2441
2442     // find closest component
2443     // @todo, something more accurate here to account for size of components, e.g. closest point on the component bbox?
2444     // @todo, sum up damage contribution to each component?
2445     float ClosestHitDistSq = MAX_FLT;
2446     for (int32 HitIdx = 0; HitIdx < RadialDamageEvent.ComponentHits.Num(); ++HitIdx)
2447     {
2448         FHitResult const& Hit = RadialDamageEvent.ComponentHits[HitIdx];
2449         float const DistSq = (Hit.ImpactPoint - RadialDamageEvent.Origin).SizeSquared();
2450         if (DistSq < ClosestHitDistSq)
2451         {
2452             ClosestHitDistSq = DistSq;
2453             ClosestHitLoc = Hit.ImpactPoint;
2454         }
2455     }
2456
2457     //1. Attempt at custom distance calculation
2458     FVector closestPoint = GetComponentsBoundingBox().GetClosestPointTo(RadialDamageEvent.Origin);
2459
2460     float const DistSq2 = (closestPoint - RadialDamageEvent.Origin).SizeSquared();
2461
2462     if (DistSq2)
2463     {
2464         if (DistSq2 < ClosestHitDistSq) {
2465             ClosestHitDistSq = DistSq2;
2466         }
2467     }
2468
2469     float const RadialDamageScale = RadialDamageEvent.Params.GetDamageScale( FMath::Sqrt(ClosestHitDistSq) );
2470
2471     ActualDamage = FMath::Lerp(RadialDamageEvent.Params.MinimumDamage, ActualDamage, FMath::Max(0.f, RadialDamageScale));
2472
2473     return ActualDamage;
2474 }
2475 }
```

Slika 63 Actor.cpp modificado za potrebe deformacije terena

Generalno radimo sa terenom koji se sastoji od 100 vrhova te pokriva 100m kvadratnih, ali moguće je i izvesti teren iz bilo kojeg statičnog oblika (eng. Static mesh).

### 3.5.3.2. Prva iteracija – replikacija unosa

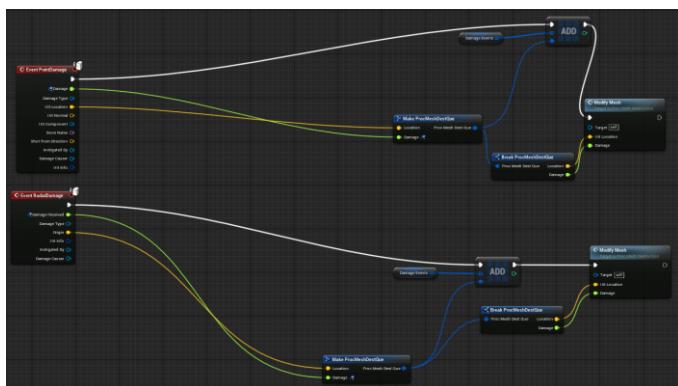
Prvi pokušaj sastojao se od simuliranja i na serveru, te na korisnicima. Dakle, ako imamo iste početne uvijete, te dajemo iste ulaze, istim redoslijedom, zagarantiran nam je isti rezultat.

Sve navedeno je točno ali dolazimo do zanimljivog problema, da bi smo očuvali redoslijed operacija moramo ih stavljati u polje te izvesti sve one operacije koje nisu izvedene na našem računalu. Problem je u tome što se ne može slati samo jedna promjena u polju, nego cijelo polje – ovo je efektivno jednako determinističkom pristupu, ali sa puno većim ulazom.

Serijalizacija u ovom slučaju izgleda kao dobra opcija. Ukratko serijalizacija osigurava da će svi paketi stići. Dakle, efektivno želimo poslati samo ulaze za simulaciju, ne cijeli niz - slanje jednog vektora i jednog broja sa pokretnim zarezom sigurno je sigurno manje od slanja rastuće liste vektora, koja bi eventualno rezultirala slanjem sve većih i većih paketa. Ali treba biti oprezan, jer dok serijalizacija garantira da će se svi pozivi obaviti, ne garantira slijed liste, dakle, moguće je da će doći do urušavanje sinkronizacije jer će početni uvjeti (pozicije vektora) biti različiti, što rezultira različitim rezultatom [99]. Alternativno moguće je na korisničkoj strani pratiti sekvencu preko nekog identifikatora, ali treba napomenuti da je lista događaja potencijalno beskonačno velika.



Slika 64 Struktura za spremanje događaja

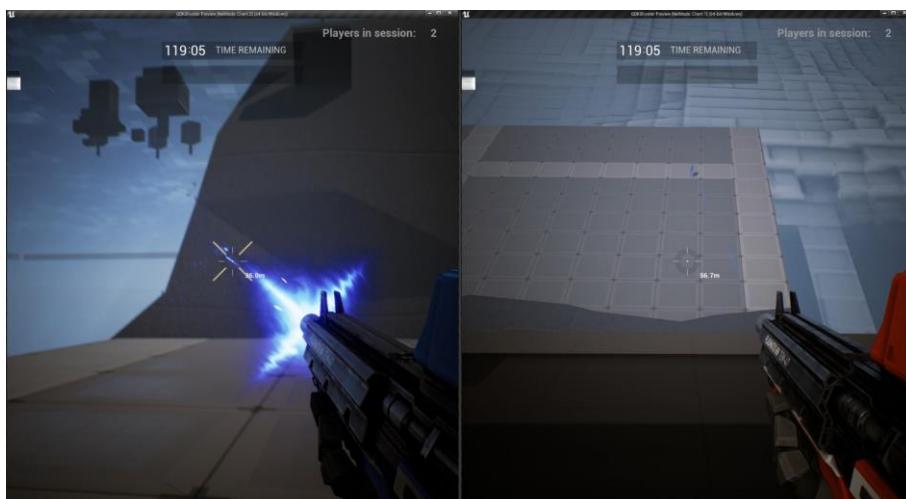


Slika 65 Obrada strukture

### 3.5.3.3. Finalna implementacija – serijalizirano polje vektora

Rješenje je dakle sljedeće. Simulacija se izvodi samo na serveru te se rezultat (dakle svih 100 vektora) sprema u replicirano polje vektora, ovo polje je automatski serijalizirano jer su vektori standardna struktura u UE4, dakle slati će se samo promjene potrebne za svakog korisnika zasebno (i to samo Z os, jer se samo ona mijenja).

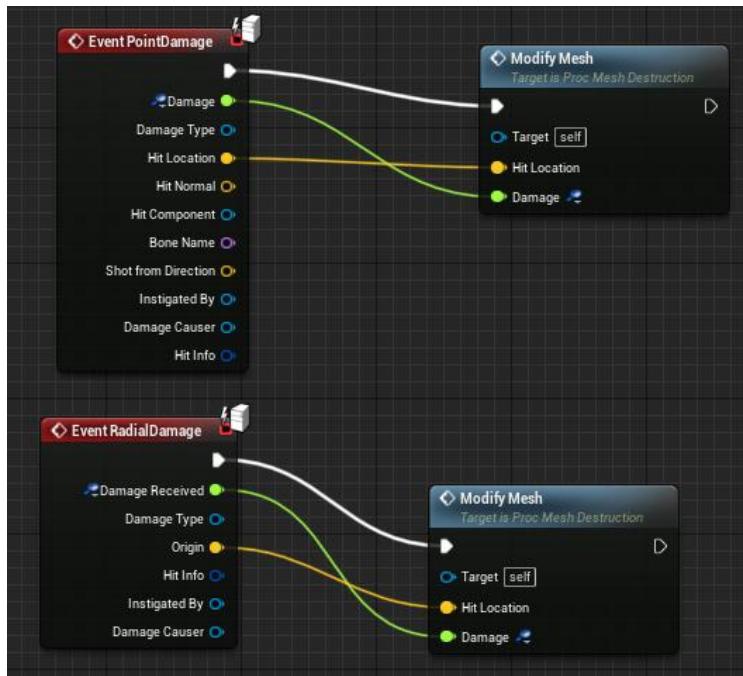
Ključ rješenja je da u ovom slučaju nije bitan poredak, jer će eventualno rezultat biti isti – sinkronizirano polje vektora. Cijena ovog rješenja je nešto veća, dakle pošto svaka promjena utječe na otprilike 8 vektora, dolazimo do relativno visoke cijene, efektivno jednake cijeni praćenja pokreta 3 igrača u jednom odazivu. Ali pošto garantiramo sinkronizaciju simulacije, te je ovo zasigurno manja cijena od prvog pokušaja, rezultat je zadovoljavajući.



Slika 66 Deformirani teren u igri

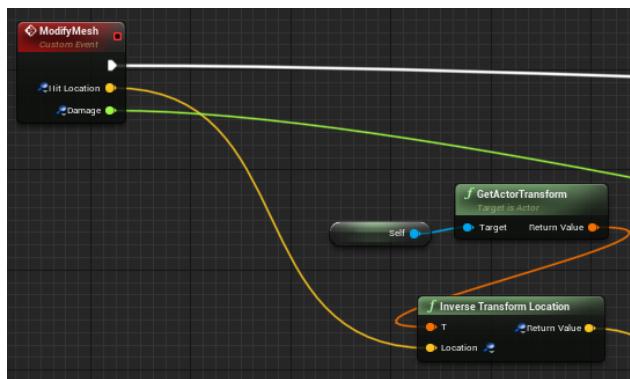
Rezultat je vidljiv na putanji „Content/TakeoverPrototype/ DestructibleEnvironment/ ProcMehsDestruction“, te na sljedećem navodu [100].

Za detekciju primanja štete koristimo događaje koji pokrivaju obije vrste štete, eksplozije i šteta bazirana na jednoj točci.



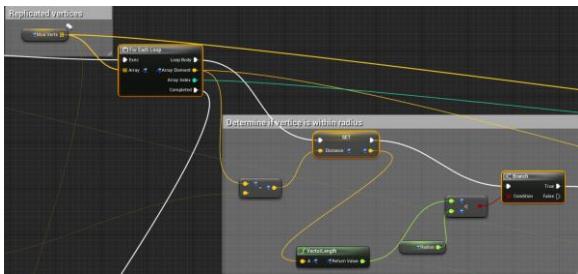
Slika 67 Događaji štete

Da bi smo mogli obraditi štetu, prvo moramo utvrditi gdje se dogodila na samom terenu, odnosno koordinate svijeta pretvoriti u koordinate na terenu, to radimo preko inverzne transformacije lokacije.



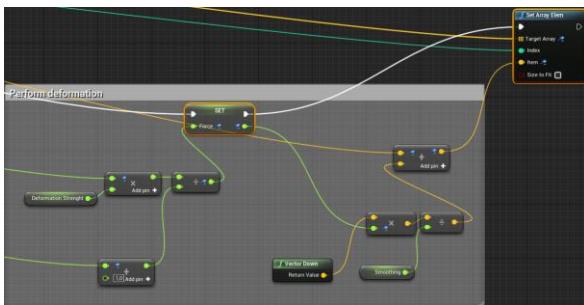
Slika 68 Inverzna transformacija lokacije

Za sve replicirane vrhove pronalazimo njihovu udaljenost od te lokacije te provjeravamo da li je unutar radijusa koji smo dopustili.



Slika 69 Prolaženje kroz sve vrhove

Nadalje provodimo deformaciju terena po vrlo jednostavnoj formuli koja je prije iskazana.



Slika 70 Deformacija terena

Naposljeku provodimo ažuriranje lokalnih vrhova.



Slika 71 Ažuriranje vrhova

### 3.5.3.4. Buduće iteracija

Ograničenje ove implementacije dolazi do izraza u njenoj veličini, dakle ako bi željeli pokriti cijeli svijet sa ovim terenom, tada imamo dvije opcije:

- Povećati teren da pokrije svijet
- U svijet staviti N instanci terena

Povećanje terena da pokrije cijeli svijet nije praktična opcija zbog više razloga. Prvobitno veliki teren bi se trebao osvježiti bilo kada bi se on modificirao, ako se radi o terenu od 64 km kvadratnih, što je standard velikih svjetova, dolazimo do neprihvatljivo velike operacije, dok bi se ovo moglo ublažiti korištenjem sekcija, postoji veći problem; Treba imati na umu da se svaki objekt replicira sa servera na klijenta s obzirom na njegovu udaljenost od tog objekta, točnije centra tog objekta (prisjetimo se sferne udaljenosti), dakle veliki teren ne bi skoro nikada bio dovoljno blizu da se osvježi, te tako ne bi izvodio svoju funkciju, a kada bi bio dovoljno blizu, količina podataka koja bi se slala klijentu bila bi neprihvatljivo velika:

$$64\text{km} * 1000 \text{ vrhova} = 64000 \text{ brojeva s pomicnim zarezom}$$

Dakle jedino senzibilno rješenje je da se svijet pokrije sa N instanci terena. Problem koji bi tada susreli je da bi susjedne terene trebali zajedno modificirati da rubni vrhovi ne „napuknu“ i stvore rupu u svijetu.

Jedno od manje očitih ograničenja ovog rješenja je da tuneli ne bi bili mogući, osim ako se ne naprave specifična ograničenja nad terenom.

Za dizajnere bi bilo povoljno i implementirati daljnja ograničenja poput mogućnosti definiranja vrhova terena koji se ne mogu micati, ili se mogu micati samo do određene vrijednosti na Z osi odnosno kuta naprema drugim vrhovima i slično, da bi se osigurao neki senzibilitet za dizajn svijeta tijekom njena uništavanja.

Nadalje, pošto UE5 prelazi na deterministički pristup, mogli bi se vratiti na prvu iteraciju ali bi trebali riješiti problem periodičnog spremanja stanja kako bi osigurali trajnost.

Naposljetku, treba spomenuti da se u svijetu proceduralnih oblika, specifično za UE4, u zadnje vrijeme pojavljuje mnogo inovacija i poboljšanja [101], te su mogućnosti za poboljšanje široke, te bi se mogle primijeniti i za uništive objekte. Slično izvedbi u igri Rainbow 6 Siege, gdje svaki metak dinamički modifcira oblik [102].

### 3.5.4. Izgradnja objekata

Izgradnja objekata pruža korisnicima izvor kreativnosti i slobode u oblikovanju svijeta. Omogućuje igračima da sagrade objekte sa različitim funkcionalnostima koja im mogu pomoći tijekom igranja, od stvaranja zaklona do mjesta za ponovno oživljavanje (eng. Respawn).

Želimo da se izgradnja događa u zasebnom pogledu, gdje se korisniku prvo prikazuje pregled kako bi objekt izgledao kada se sagradi, sa mogućnošću da se ih postavi više u jednom redu kao ponovljiv uzorak, slično kao u igri Company Of Heroes. Kada se oni postave, želimo mogućnost njihove izgradnje te rušenja.

#### 3.5.4.1. Prva iteracija – statički objekti

Kako bi ostvarili sve navedeno u specifikaciji, izraditi ćemo nekoliko klase za upravljanje raznim aspektima ove funkcionalnosti.

U klasi „Buildable“ definirati ćemo sve što čini objekt koji se može izgraditi. Potreban nam je oblik koji reprezentira pregled izgradnje, komponente za zdravlje objekta koja će primati podatke o izgradnji i rušenju te oblici za razne faze izgradnje.

```
9   // Sets default values
10  ABuildable::ABuildable()
11  {
12      // Set this actor to call Tick() every frame. You can turn this off to improve performance if your tick function is cheap enough.
13      PrimaryActorTick.bCanEverTick = true;
14      bReplicates = true;
15
16      TestTag1 = CreateDefaultSubobject<UTestTag>(TEXT("TestTag1"));
17
18      HealthComponent = CreateDefaultSubobject<UHealthComponent>(TEXT("Health"));
19      HealthComponent->SetIsReplicated(true);
20
21      RootComponent = CreateDefaultSubobject<USceneComponent>(TEXT("RootComponent"));
22      RootComponent->SetIsReplicated(true);
23
24      PreviewMesh1 = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("PreviewMesh"));
25      PreviewMesh1->SetupAttachment(RootComponent);
26      PreviewMesh1->SetVisibility(true);
27      PreviewMesh1->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
28      PreviewMesh1->SetIsReplicated(true);
29
30
31      BuildMesh1 = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("BuildMesh1"));
32      BuildMesh1->SetupAttachment(RootComponent);
33      BuildMesh1->SetVisibility(false);
34      BuildMesh1->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
35      BuildMesh1->SetIsReplicated(true);
36
37      BuildMesh2 = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("BuildMesh2"));
38      BuildMesh2->SetupAttachment(RootComponent);
39      BuildMesh2->SetVisibility(false);
40      BuildMesh2->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
41      BuildMesh2->SetIsReplicated(true);
42
43      BuildMesh3 = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("BuildMesh3"));
44      BuildMesh3->SetupAttachment(RootComponent);
45      BuildMesh3->SetVisibility(false);
46      BuildMesh3->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
47      BuildMesh3->SetIsReplicated(true);
48
49      collision.Init(false, 3);
50  }
```

Slika 72 Konstruktor klase Buildable.cpp

Dakle u konstruktoru klase definiramo sve komponente i oblike, treba napomenuti da se svi oblici skrivaju preko „SetVisibility“, te im se isključuje sva kolizija.

Kod kolizije dolazimo do zanimljivog problema, jer se ona ne replicira jer je tako dizajnirana zbog potreba UE4 [103], a mi je trebamo mijenjati tijekom igranja, jer treba biti ugašena za skrivene faze, te upaljena kada se do njih dođe. Ovo rješavamo replikacijom boolean liste, jedan boolean za svaku fazu, te korištenjem „On\_Rep“ funkcije koja se aktivira kada dođe do replikacije. U toj funkciji – „OnRep\_Collision“, se ovisno o postavljenim vrijednostima zadaje kolizijski profil.

```
59     void ABuildable::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const
60     {
61         Super::GetLifetimeReplicatedProps(OutLifetimeProps);
62
63         DOREPLIFETIME(ABuildable,collision);
64     }
```

Slika 73 Postavljanje repliciranih varijabli kolizije

```
85     void ABuildable::OnRep_Collision() {
86     if (collision[0]) {
87         BuildMesh1->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
88     }
89     else {
90         BuildMesh1->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
91     }
92
93     if (collision[1]) {
94         BuildMesh2->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
95     }
96     else {
97         BuildMesh2->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
98     }
99
100    if (collision[2]) {
101        BuildMesh3->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
102    }
103    else {
104        BuildMesh3->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
105    }
106 }
```

Slika 74 Funkcija koja se aktivira replikacijom kolizije

Sada možemo izvesti zadane funkcionalnosti. Za postavljanje objekta koristimo funkciju „Place“, u kojoj se mijenja vidljivost i kolizijski profil objekata, te zadajemo početno zdravlje, definirano kao 10% ukupnog zdravlja.

```

73     void ABuildable::Place() {
74         PreviewMesh1->SetVisibility(false);
75         BuildMesh1->SetVisibility(true);
76
77         BuildMesh1->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
78         collision[0] = true;
79
80         HealthComponent->GrantHealth((1 / 10) * HealthComponent->GetMaxHealth());
81         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("ABuildable::Place"));
82     }
83 }
```

Slika 75 Funkcija za postavljanje objekata

Treba napomenuti da je modificirana funkcija komponente zdravlja jer se u njenoj inicijalnoj implementaciji, pri konstruiranju, odmah zadalo puno zdravlje. Ovo smo ispravili postavljanjem nove varijable „startHealth“ kojom definiramo početno zdravlje (0).

```

13     UHealthComponent::UHealthComponent()
14     {
15         PrimaryComponentTick.bCanEverTick = true;
16
17         SetIsReplicatedByDefault(true);
18
19         if (startHealth == 0.f) {
20             startHealth = 100.f;
21         }
22         if (MaxHealth == 0.f) {
23             MaxHealth = 100.f;
24         }
25         CurrentHealth = startHealth;
26         if (MaxArmour == 0.f) {
27             MaxArmour = 100.f;
28         }
29         CurrentArmour = 0.f;
30     }
```

Slika 76 Komponente zdravlja

Sada nam samo ostaje obraditi slučajeve izgradnje i primanja štete. Izgradnja se događa u funkciji „Build“ gdje se samo zadaje zdravlje te ažuriraju kolizijski profili i vidljivost faza.

```

10     void ABuilable::Build(float value) {
11         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("ABuilable::Build"));
12         HealthComponent->GrantHealth(value);
13         HealthComponent->SetHealth(value);
14
15         void ABuildable::HealthUpdate() {
16             if (HealthComponent->GetCurrentHealth() >= 0.35 * HealthComponent->GetMaxHealth() && HealthComponent->GetCurrentHealth() < 0.75 * HealthComponent->GetMaxHealth()) {
17                 BuildMesh1->SetVisibility(true);
18                 collision[1] = true;
19                 BuildMesh2->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
20
21                 BuildMesh1->SetVisibility(false);
22                 collision[0] = false;
23                 BuildMesh1->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
24
25                 BuildMesh2->SetVisibility(false);
26                 collision[2] = false;
27                 BuildMesh2->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
28             }
29             else if (HealthComponent->GetCurrentHealth() > 0.75 * HealthComponent->GetMaxHealth()) {
30                 BuildMesh1->SetVisibility(true);
31                 collision[1] = true;
32                 BuildMesh2->SetCollisionProfileName(UCollisionProfile::BlockAll_ProfileName);
33
34                 BuildMesh1->SetVisibility(false);
35                 collision[0] = false;
36                 BuildMesh1->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
37
38                 BuildMesh2->SetVisibility(false);
39                 collision[2] = false;
40                 BuildMesh2->SetCollisionProfileName(UCollisionProfile::NoCollision_ProfileName);
41             }
42             else if (HealthComponent->GetCurrentHealth() == 0) {
43                 this->Destroy();
44             }
45         }
```

Slika 77 Ažuriranje kolizijskih profila

Slučaj primanja štete je zapravo identičan onome u liku, te je implementacija vrlo slična.

```

147 float ABuildable::TakeDamage(float Damage, const FDamageEvent& DamageEvent, AController* EventInstigator, AActor* DamageCausers)
148 {
149     TakeDamageCrossServer(Damage, DamageEvent, EventInstigator, DamageCausers);
150     return Damage;
151 }
152
153 void ABuildable::TakeDamageCrossServer_Implementation(float Damage, const FDamageEvent& DamageEvent, AController* EventInstigator, AActor* DamageCausers)
154 {
155     float ActualDamage = Super::TakeDamage(Damage, DamageEvent, EventInstigator, DamageCausers);
156     HealthComponent->TakeDamage(ActualDamage, DamageEvent, EventInstigator, DamageCausers);
157     HealthUpdate();
158 }

```

Slika 78 Funkcije za primanje štete

Da bi mogli koristiti ovaj objekt potrebno je izraditi komponentu za klijenta koja će organizirati pogled izgradnje i zvati funkcionalnosti objekta.

Ovo ćemo izvesti pomoću klase „BuildManagerComponent“, njen rad se obavlja u njezinom otkucaju, prvo se čeka uključivanje pogleda za izgradnju, sa tipkom „O“, nakon čega se korisniku prikazuje pregled objekta za izgradnju, takozvani „ToggleBuildMode“, koji je definiran u postavkama projekta. Pritiskom na srednji gumb miša korisnik potvrđuje početnu poziciju objekta za izgradnju (ponovo definirano u postavkama projekta), ovo poziva funkciju „RequestBuild“.



Slika 79 Postavke projekta

```

194 void UBuildManagerComponent::ToggleBuildMode() {
195     isBuilding = !isBuilding;
196     GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("ToogleBuildMode"));
197     if (!canBuild || !isBuilding) {
198         for (int i = 0; i < managedBuildables.Num(); i++) {
199             managedBuildables[i]->Destroy();
200         }
201     }
202     managedBuildables.Empty();
203 }

```

Slika 80 Funkcija za pokretanje pregleda za izgradnju

```

205 void UBuildManagerComponent::RequestBuild() {
206     if (canBuild) {
207         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("RequestBuild"));
208         previewMode = false;
209         currentBuildable->PreviewMesh1->ToggleVisibility();
210         plantingPoint = currentTrace;
211     }
212 }

```

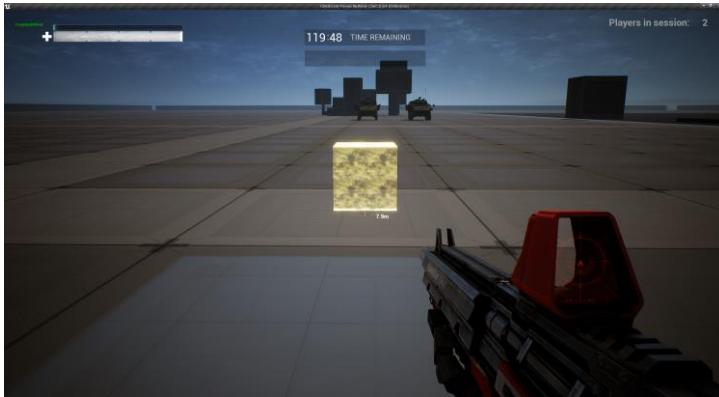
Slika 81 Funkcija za zahtjev izgradnje

Da bi se on mogao prikazati prvo se dobavlja kamera korisnika, i točka u koju on gleda, preko metode linijskog traga. Nakon što smo definirali poziciju, možemo stvoriti objekt, zapamtimo, ovo radimo samo lokalno, na korisničkoj strani, ne serverskoj. Treba napomenuti

da je kolizija za objekt pregleda isključena kako on ne bi bio u koliziji sam sa sobom dok se postavlja.

```
50     if (isBuilding) {
51         FHitResult HitResult;
52         float LineTraceDistance = 1800.f;
53         float HeadOffset = 150.f;
54
55         FRotator CameraRotation = playerCamera->GetComponentRotation();
56         FVector Start = playerCamera->GetComponentLocation() + (CameraRotation.Vector() * HeadOffset);
57
58         FVector End = Start + (CameraRotation.Vector() * LineTraceDistance);
59
60         FCollisionQueryParams TraceParams(FName(TEXT("InteractTrace")), true, NULL);
61         TraceParams.bTraceComplex = true;
62         TraceParams.bReturnPhysicalMaterial = true;
63
64         bool bIsHit = GetWorld()->LineTraceSingleByChannel(
65             HitResult,           // FHitResult object that will be populated with hit info
66             Start,              // starting position
67             End,                // end position
68             ECC_Visibility,    // collision channel
69             TraceParams);      // additional trace settings
70     );
71
72     currentTrace = HitResult.ImpactPoint;
73 }
74
```

Slika 82 Pronalaženje korisničkog pogleda



Slika 83 Pregled izgradivog objekta

U slučaju da se pozicija u koju korisnik gleda pomakne ( dok drži srednju tipku miša), tada se stvara još objekata između početne i trenutne točke, tako da između objekata nema praznog prostora. Treba napomenuti da se objekti mogu postaviti samo u ograničenoj udaljenosti od lika.

```
114     //For each full distance spawn a buildable and add it to managedBuildables
115     FActorSpawnParameters SpawnParams;
116     SpawnParams.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButAlwaysSpawn;
117     if (managedBuildables.Num() < amountOfCover) {
118         for (int i = managedBuildables.Num(); i < amountOfCover; i++) {
119             FVector additiveVector = plantingPoint + (distanceCalculator.GetSafeNormal() * (debugSize.Size() * i));
120             managedBuildables.Add(GetWorld()->SpawnActor<ABuildable>(BuildableFortification, additiveVector, distanceCalculator.Rotation(), SpawnParams));
121         }
122     }
123 }
```

Slika 84 Funkcija za postavljanje više objekata u redu



Slika 85 Postavljanje više objekata u redu

Kako bi izbjegli objekte koji probijaju druge objekte koristiti ćemo ponovo linijski trag, koji započinje od visine objekta za izgradnju, te se crta prema dnu (Z os).



Slika 86 Obrada slučaja probijanja

```

132     //Fix rotation in real time
133     if (managedBuildables.Num() > 0) {
134         for (int i = 0; i < managedBuildables.Num(); i++) {
135             FVector additiveVector = plantingPoint + (distanceCalculator.GetSafeNormal() * (debugSize.Size() * i));
136
137
138
139             //Shoot trace up
140             FVector endVectorCheck = additiveVector + (FVector(0, 0, 1) * 2000.f);
141             FHitResult HitResult1;
142
143             bool traceUp = GetWorld()->LineTraceSingleByChannel(
144                 HitResult1,           // FHitResult object that will be populated with hit info
145                 additiveVector,       // starting position
146                 endVectorCheck,      // end position
147                 ECC_Visibility,      // collision channel
148                 TraceParams          // additional trace settings
149             );
150
151
152             //Shoot trace down
153             FVector endVectorCheck2 = additiveVector + (FVector(0, 0, -1) * 10000.f);
154             FHitResult HitResult2;
155             FVector startVector2;
156             if (traceUp) {
157                 startVector2 = HitResult1.ImpactPoint;
158             }
159             else {
160                 startVector2 = endVectorCheck;
161             }
162             bool traceDown = GetWorld()->LineTraceSingleByChannel(
163                 HitResult2,           // FHitResult object that will be populated with hit info
164                 startVector2,         // starting position
165                 endVectorCheck2,      // end position
166                 ECC_Visibility,      // collision channel
167                 TraceParams          // additional trace settings
168             );
169
170             //If something is hit while going down, place the object there, else destroy it
171             FVector spawnVector;
172             if (traceDown) {
173                 spawnVector = HitResult2.ImpactPoint;
174                 FRotator finRotation = FRotator(spawnVector.Rotation().Pitch, distanceCalculator.Rotation().Yaw, spawnVector.Rotation().Roll);
175                 managedBuildables[i]->SetActorLocationAndRotation(spawnVector, finRotation);
176             }
177             else {
178                 managedBuildables[i]->Destroy();
179             }
180         }
181     }

```

Slika 87 Funkcija za obradu probijanja pomoću linijskih tragova

Kada smo zadovoljni sa postavljanjem objekata, puštamo srednju tipku miša, što aktivira funkciju „ReleaseBuild“, ta funkcija traži upravljača objekata koji će ih stvoriti na serverskoj strani, to radi preko funkcije „Server\_PlaceBuildable“.

```

36     UFUNCTION(Server, reliable, WithValidation, BlueprintCallable)
37         void Server_PlaceBuildable(FVector const& Location, FRotator const& Rotation);
38

```

Slika 88 Funkcija za postavljanje objekta na serverskoj strani

Ovdje se zapravo zadaje zahtjev za stvaranje ovih objekata, tako da se mogu prikazati na svim ostalim klijentima.

```

240     void UBuildManagerComponent::Server_PlaceBuildable_Implementation(FVector const& Location, FRotator const& Rotation) {
241
242         TArray<AActor*> FoundActors;
243         UGameplayStatics::GetAllActorsOfClass(GetWorld(), ABuildableManager::StaticClass(), FoundActors);
244         ABuildableManager* foundManager = Cast<ABuildableManager>(FoundActors[0]);
245
246         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("UBuildManagerComponent::Server_PlaceBuildable_Implementation"));
247
248         foundManager->SpawnRequest(BuildableFortification, Location, Rotation);
249     }

```

Slika 89 Implementacija funkcije za postavljanje objekta na serverskoj strani

To nas dovodi do klase „BuildableManager“ koja upravlja ovim objektima te se brine za njihovo stvaranje na serverskoj strani, ovo radi preko funkcije „SpawnRequest“.

```
28     void ABuildableManager::SpawnRequest(TSubclassOf<class ABuildable> BuildableFortification, FVector const& Location, FRotator const& Rotation) {
29         FActorSpawnParameters SpawnParams;
30         SpawnParams.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AdjustIfPossibleButAlwaysSpawn;
31
32         ABuildable* tempBuildable = GetWorld()->SpawnActor<ABuildable>(BuildableFortification, Location, Rotation, SpawnParams);
33         tempBuildable->Place();
34         //tempBuildable->Build(90);
35         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("ABuildableManager::SpawnRequest"));
36         managedBuildables.Add(tempBuildable);
37     }
38 }
```

Slika 90 Funkcija za stvaranje objekata u stvaratelju



Slika 91 Postavljeni objekti u igri

Naposljeku, kada smo postavili sve objekte, potrebno ih je izgraditi, za to nam je potrebno posebno oružje koje će obaviti tu funkciju. Ovo se obavlja pomoću klase „MyInstantWeapon“ koja nasljeđuje klasu „Weapon“ radi sličnih funkcionalnosti. Razlika je naravno da ovo oružje ne zadaje štetu, nego izgrađuje objekte ako su pogodeni, te naravno ako se mogu izgraditi.

```
174     void AMyInstantWeapon::DealDamage(const FInstantHitInfo& HitInfo)
175     {
176         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("DealDamage - MyInstantWeapon.cpp"));
177
178         GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, HitInfo.HitActor->GetClass()->GetFName().ToString());
179         if (HitInfo.HitActor->IsA(ABuildable::StaticClass())) {
180             Cast<ABuildable>(HitInfo.HitActor)->Build(25.f);
181             GEngine->AddOnScreenDebugMessage(-1, 15, FColor::Green, TEXT("DealDamage + HIT - MyInstantWeapon.cpp"));
182         }
183     }
```

Slika 92 Modificirano oružje za izgradnju objekata



Slika 93 Izgrađeni objekt

### **3.5.4.2. Buduće iteracije – fizički objekti**

Nedostatak prijašnje iteracije je u tome što smo zapravo stvorili statične objekte, koji ne reagiraju na fizičke događaje u svijetu. Trenutna implementacija je zapravo bila jedna od prvih značajki koja se implementirala, te se zato nije tako inicijalno izvela.

Dakle buduća iteracija trebala bi stvoriti fizički objekt, kao što smo definirali u umreženju simulacije fizike objekata, kojeg bi po izboru zaključali na Z os radi konzistencije zaklona, ali treba napomenuti da zaključavanje osi može narušiti ponašanje drugih fizičkih objekata.

Naposljetku, ovisno o dizajnu funkcionalnosti igre, bilo bi potrebno izraditi sustav konzumacije resursa za izgradnju, što bi ograničilo korisnike da ne stvaraju bezbroj mnogo objekata.

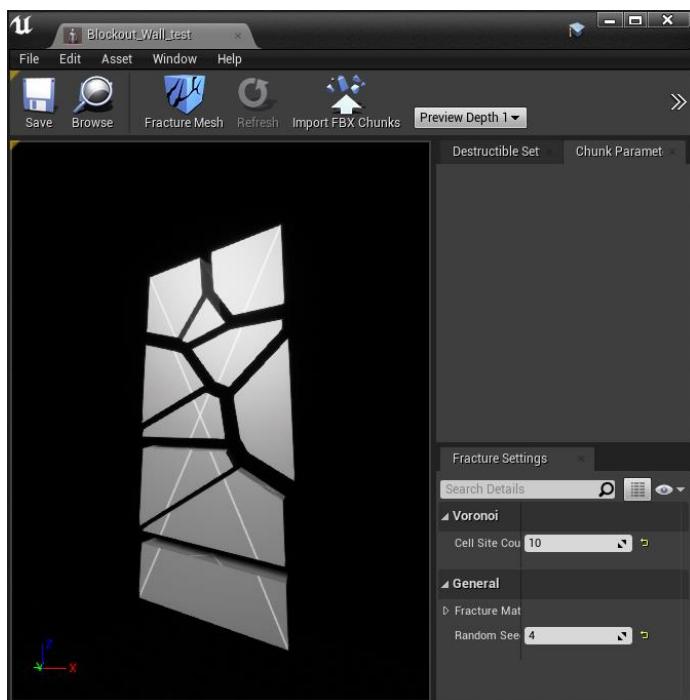
### 3.5.5. Uništivi objekti

Uništivi objekti okupiraju drugu stranu izgradnju objekata, ali imaju gotovo isti efekt na krajnjeg korisnika. Dakle pruža se duboka prilika za taktičku kreativnost i sloboda u kretanju kroz svijet.

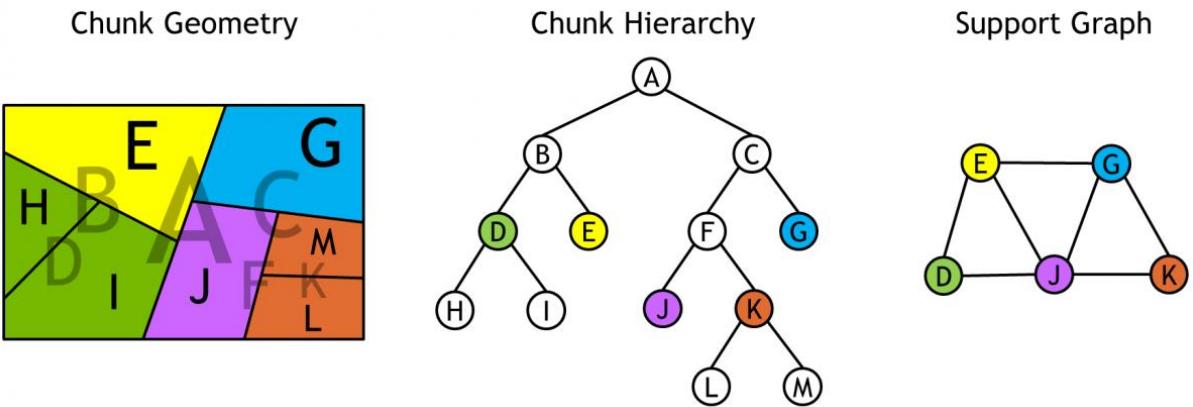
#### 3.5.5.1. Prva iteracija – PhysX APEX + DENT

PhysX APEX je trenutni sustav uništivih objekata, u potpunosti podržan u UE4 [104], te će se zato koristiti u ovom projektu.

On uzima bilo koji objekt te ga pretvara u skup krhotina (eng. Debris), koje su povezane kroz jednostavnu hijerarhiju i graf povezanosti, što zapravo definira kako se objekt može uništiti.



Slika 94 Objekt pretvoren u skup krhotina



Slika 95 Hijerarhija i graf povezanosti [105]

Zbog jednostavnije izvedbe ovog dijela rada, koristi se priključak DENT, dostupan na UE tržnici [106], on jednostavno replicira informaciju o frakturiranju pojedine krhotine.

Temeljni problem sa APEX je da je to zapravo odvojeni i većinom zatvoreni API nad kojim nemamo kontrole, te ga teško konfiguriramo jer je napisan u potpuno drugom jeziku i okruženju. Posljedica ovog dizajna je da je uništavanje vrlo ograničeno i teško se konfiguriра za umrežene potrebe.

Srž svih umreženih problema zapravo proizlazi iz manjka kontrole nad prije navedenim krhotinama. Naime, krhotine ne možemo pratiti kao obične objekte, te im zato ne možemo pratiti poziciju odnosno ne možemo replicirati njihovo stanje na serveru klijentima, dakle dolazi do gubitka sinkronizacije pozicija. Primijetimo razlike pozicije krhotina uništene zgrade na sljedećoj slici.



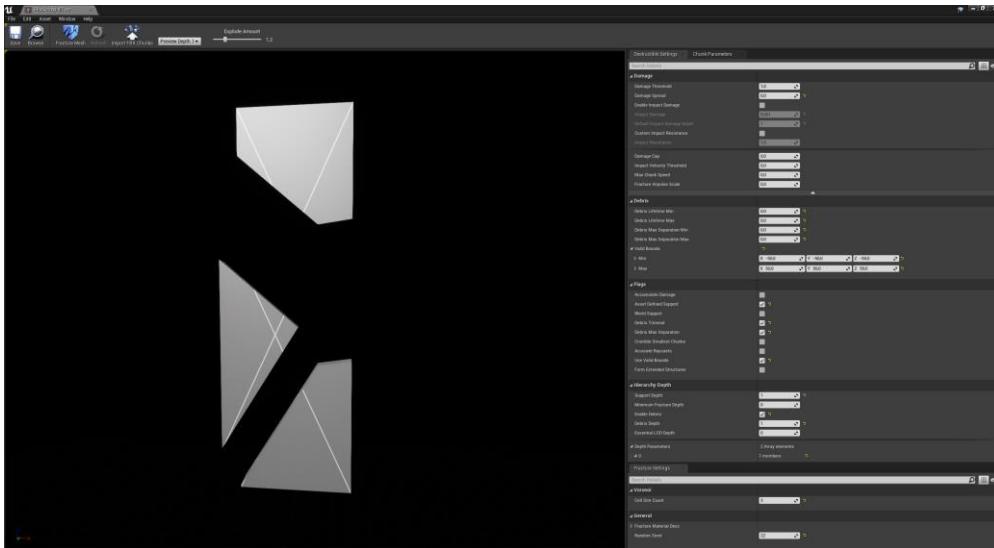
Slika 96 Različite pozicije krhotina na klijentima

Nadalje dolazimo do pojma prag velikih krhotina (eng. Large Chunk Threshold), on definira da krhotine ispod praga imaju koliziju, dok krhotine iznad njega nemaju koliziju. Zbog ovih ograničenja prisiljeni smo na vrlo mali broj mogućih konfiguracija uništivih objekata.

Dakle, konfiguracija koja se preporuča je sljedeća:

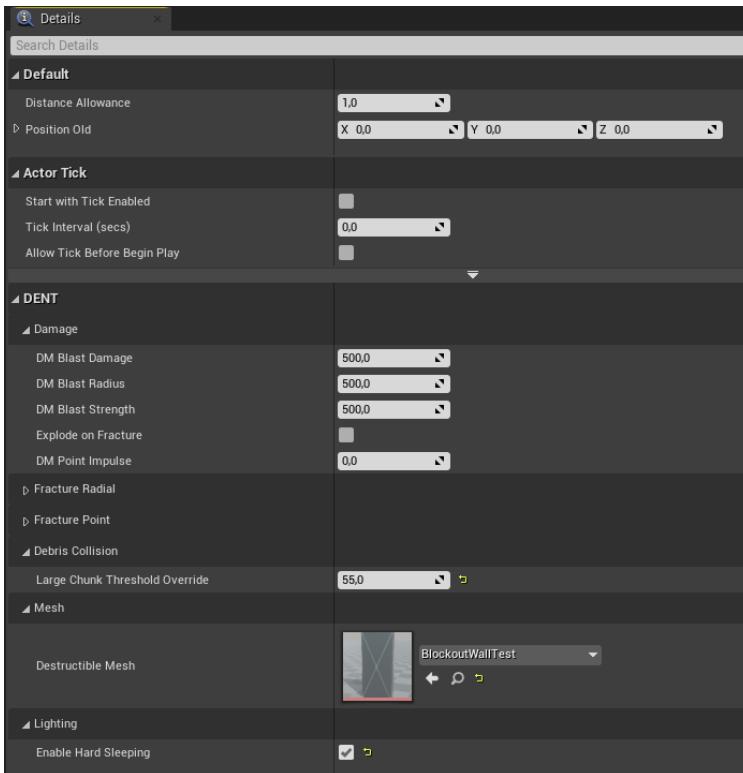
- Ne simulirati krhotine nakon što se one odvoje od glavnog objekta, jer im je tada pozicija nepoznata.
- Sve krhotine neka budu povezane direktno na temeljnu, potpornu krhotinu (eng. Support Chunk), tako da nikada nije moguće doći do slučaja gdje dvije krhotine ostanu povezane, ali odvojene od glavnog objekta, jer bi se dogodio prvi slučaj.

Jedan ovako konfiguriran uništivi objekt, dostupan je na putanji „Content/TakeoverPrototype/DestructibleEnvironment/StaticMeshPhysics/BlockoutWallTest“. Na sljedećoj slici je vidljiva konfiguracija.



Slika 97 Pravilno konfiguriran uništivi objekt

Nadalje u DENT je konfiguriran kao što je prikazano na sljedećoj slici.



Slika 98 Konfiguracija uništivog objekta

Kako bi ovu konfiguraciju mogli koristiti u sklopu većeg objekta, potrebno je konfigurirati koliziju, koristeći kolizijsko filtriranje [107], tako da se ne događaju kolizije

između fizičkih tijela, konfiguracija je dostupna na putanji „Content/ TakeoverPrototype/ DestructibleEnvironment/ StaticMeshPhysics/ MyStaticMeshPhysicsTestV4“ i „...DestructibleWallFinal“, te vidljiva na sljedećoj slici.



Slika 99 Konfiguracija kolizije

Treba napomenuti da je korijen na koji se ovi zidovi priključuju kolizijskog tipa „Destructible“. Ovaj korijen je potreban jer želimo da su uništivi objekti dio dinamičnog svijeta, odnosno da su kompatibilni sa deformacijom terena i slično. Rezultat je vidljiv u video zapisu sljedećeg navoda [108].



Slika 100 Sinkroniziran uništivi objekt [108]

Moguće je konfigurirati uništivi objekt sa više krhotina, ali tada on treba biti uništen u potpunosti kada zaprimi štetu, ovo je povoljno za manje objekte.

Treba napomenuti da postoji alat zvan PhysXLab [109] pomoću kojeg je moguće ručno oblikovati krhotine tako da se navedeni problemi nikada ne dogode, ali za potrebe ovog rada zadana konfiguracija je zadovoljavajuća.

### **3.5.5.2. Trenutna generacija – PhysX Blast**

PhysX Blast je trenutna generacija uništivih objekata, ona pruža bolje performanse od APEX te ima nešto bolju kontrolu nad umreženjem [105].

Ova značajka dostupna je kao dodatak za UE4 na „Nvidia GameWorks“ platformi [110]. Ali se ona ne obrađuje u ovom radu, jer temeljni problemi nisu riješeni, to je još uvijek zaključana platforma koju ne možemo modificirati za svoje potrebe, te se zato neće trošiti vrijeme na njenu implementaciju.

### **3.5.5.3. Buduća iteracija – Chaos destrukcija**

Gledajući u budućnost, u sklopu sa Chaos fizičkim sustavom, dobivamo i ukomponirani sustav za uništive objekte. Ovo rješenje je potpuno integrirano u UE [111], u smislu da je otvorenog je tipa, dakle bilo tko ga može modificirati za svoje potrebe, nadalje ono se u potpunosti se može kontrolirati u UE, bez vanjskih alata.

Za nas najvažnija činjenica je da je ovo rješenje primjerno za korištenje u umreženim okruženjima, dakle nismo ograničeni na samo neke konfiguracije.

Naravno to znači da se krhotine mogu pratiti i nakon odvajanja od objekta. Nadalje krhotine su integrirane u druge aspekte igre poput:

- Navigacija za AI [112]
- Sustav za efekte
- Sustav za zvuk
- Generiranje raznih događaja na koje okoliš može reagirati, na primjer pri udaru krhotine u pod može se generirati reakcija lika i slično.
- Potpora za trajnost (sustav za praćenje stanja)
- Spremljena simulacija (eng. Cached Simulation) gdje iznimno kompleksne simulacije možemo provesti parcijalno prije njihovog izvođenja.

## Chaos: key concepts

- Construction Best practices
- Geometry Collections
- Clustering
- Fracturing
- Fields
- Connection Graph
- Cached Simulations
- Niagara Integration



Slika 101 Temeljni koncepti "Chaos" [112]

U smislu načina rada, ne razlikuje se puno od PhysX, te se ponovo koristi graf povezanosti, njegova prava prednost leži u dubokoj integraciji.

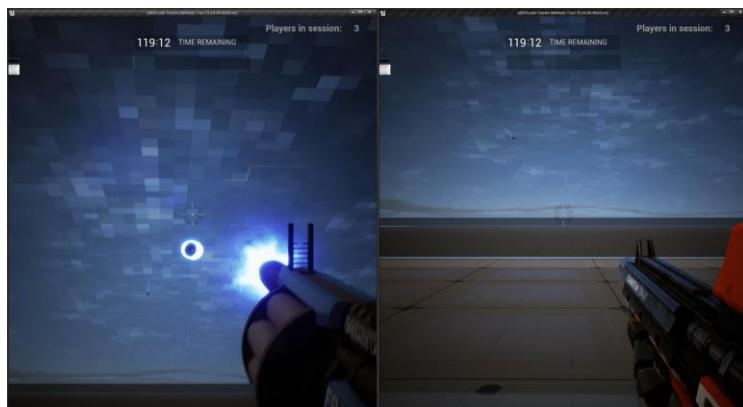
### 3.5.6. Simulacija projektila i balistike

Sustav za simulaciju projektila i balistike je kritičan, ne samo sa strane realizma igre, nego i sa strane sigurnosti igre, te detaljnosti simulacije, koristi se u simuliranju metaka i njihova probijanja objekata.

#### 3.5.6.1. Projektili

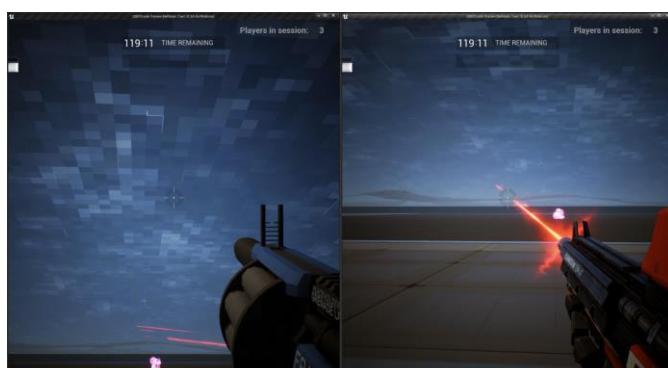
Projektile možemo implementirati na dva načina, kao fizički objekt i linijskim tragom (eng. Line Trace), koje ćemo sada usporediti.

Projektil kao fizički objekt je običan fizički objekt kojem se pri izlazu iz oružja zadaje brzina, te može biti u interakciji sa okruženjem.



Slika 102 Projektil kao fizički objekt u igri

Linijski trag je druga metoda kojim se projektili mogu implementirati, to je jednostavna linija koja se povlači od jedne do druge točke, te se ispituje sa kojim je sve objektima na tom putu u koliziji.



Slika 103 Linijski trag u igri

Očito je da svaki metak, na sebe prima razne utjecaje, kao što su trenje zraka, gravitacija te vjetar i slično, te mu je potrebno neko vrijeme da stigne do svoje mete.

Sve navedeno fizički projektil može podnijeti sa svojom funkcionalnošću, jer je dio fizičkog sustava.

Linijski trag je u ovom smislu precizan na kratke etape, te svako oružje ima gotovo beskonačan domet, bez ikakvog utjecaja gravitacije, te bez navođenja mete, jer svaki projektil odmah dolazi do svoje mete.

Sa strane sigurnosti igre, fizički projektili su vrlo sigurni, jer se mogu ispaliti sa serverske strane te zato biti gotovo nemogući za eksploataciju.

Linijski trag ovisi o korisničkoj strani, dakle, ako korisnik ne vidi metu, ne može ju pogoditi jer on određuje početak i kraj puta koji projektil prolazi, ovo otvara i razne sigurnosne probleme za igrače koji varaju.

Moguće je ispaliti linijski trag na serverskoj strani, ali onda se oslanjamamo na činjenicu da i server sve vidi, što nije uvijek slučaj. Osim toga, ako je on sa serverske strane onda može doći do većeg odaziva pri ispaljivanju te pogotku, što poništava sve prednosti ove metode.

Zbog svih ovih činjenica odabrana je fizička simulacija projektila. Već postojeća simulacija projektila u projektu je zadovoljavajuća za potrebe projekta, te se samo modificiralo ponašanje pogodaka, odnosno balistika.

### 3.5.6.2. Balistika

Cilj sustava balistike je simulacija probijanja i štete koju bi pogodak projektila uzrokovao nekom objektu. Njegova primjena je široka, od probijanja zidova do oklopa tenka i slično. Općenito podiže razinu detaljnosti i realizma igre te ako se primjenjuje dovoljno široko i implementira sa sustavom uništavanja može biti odličan. Ovaj sustav inspiriran je većinskim igrom „War Thunder“ [113].

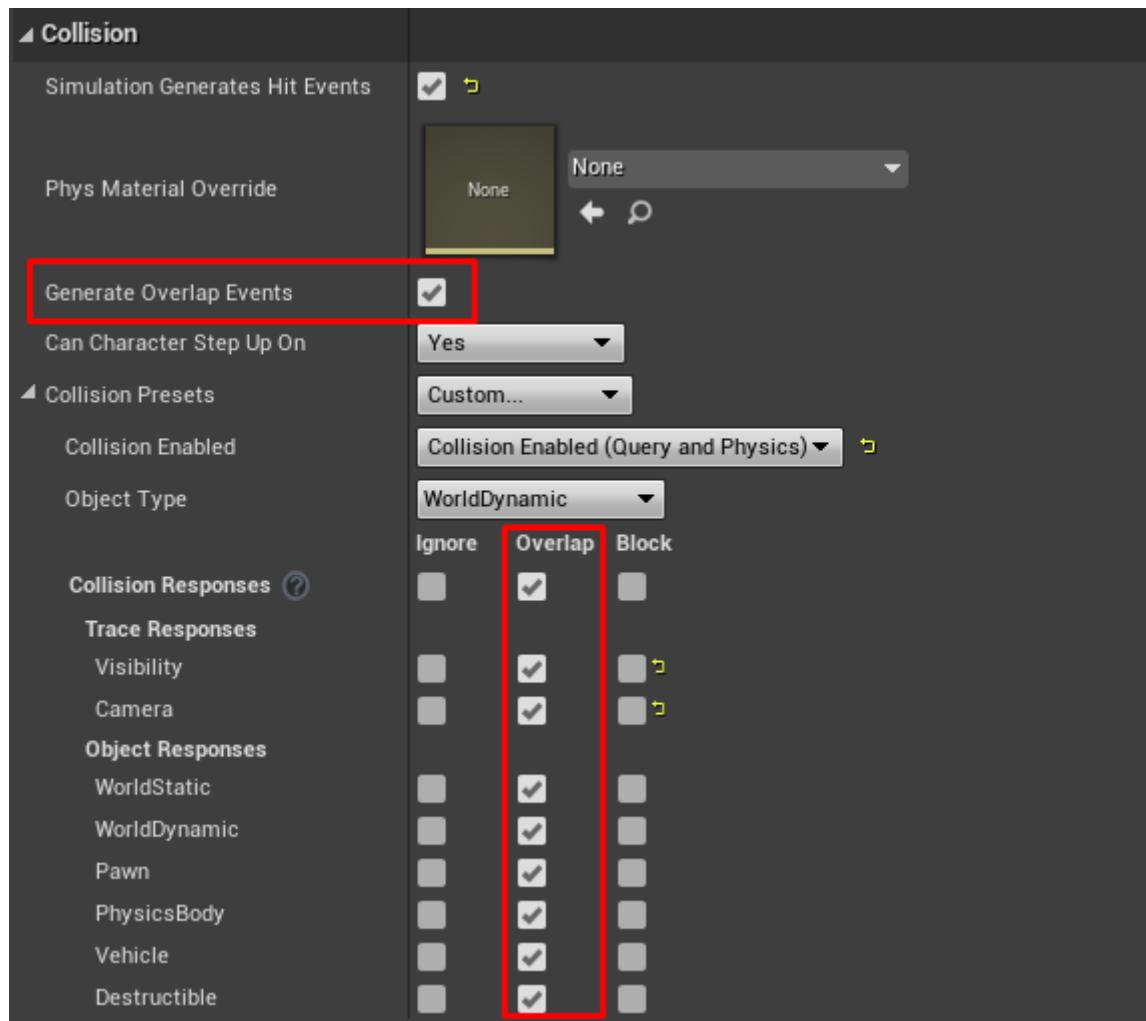
Velika većina dostupnih izvedbi oslanja se na linijski trag, ne na fizički simulirane objekte, što daleko olakšava izvedbu, ali očito nije povoljna za naš dizajn.

Postoji jedna javno dostupna implementacija koja koristi fizički simulirane objekte, to je „Mipmap“ izvedba [114]. Ukratko, u njoj se autor oslanja na kolizijski događaj odbijanja (kada se fizički objekt odbije od drugog), te onda odlučuje da li će projektil probiti metu ili ne.

Nedostatak ove metode je to što se nakon probijanja projektil mora obrisati, i ponovo stvoriti sa druge strane probijenog objekta, što je jako pogubno za performanse umreženja.

Prisjetimo se da stvaranje objekata mora biti sa serverske strane, te je za to potrebno poslati pozicijske podatke, uz brzinu i slične attribute. Ovo bi bilo pogubno za detaljna okruženja, gdje je gotovo svaki objekt moguće probiti.

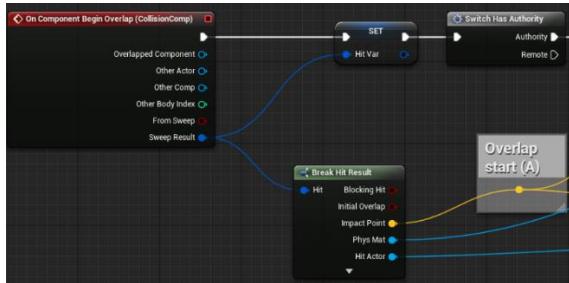
Zbog ovoga osmišljena je nova metoda, koja se oslanja na događaj preklapanja kolizije. Kada se kolizija dogodi mi ne zaustavljamo projektil, odnosno isključili smo bilo kakvo fizičko blokiranje projektila, ali mi to još uvijek možemo pratiti preko događaja preklapanja.



Slika 104 Kolizijska komponenta metka

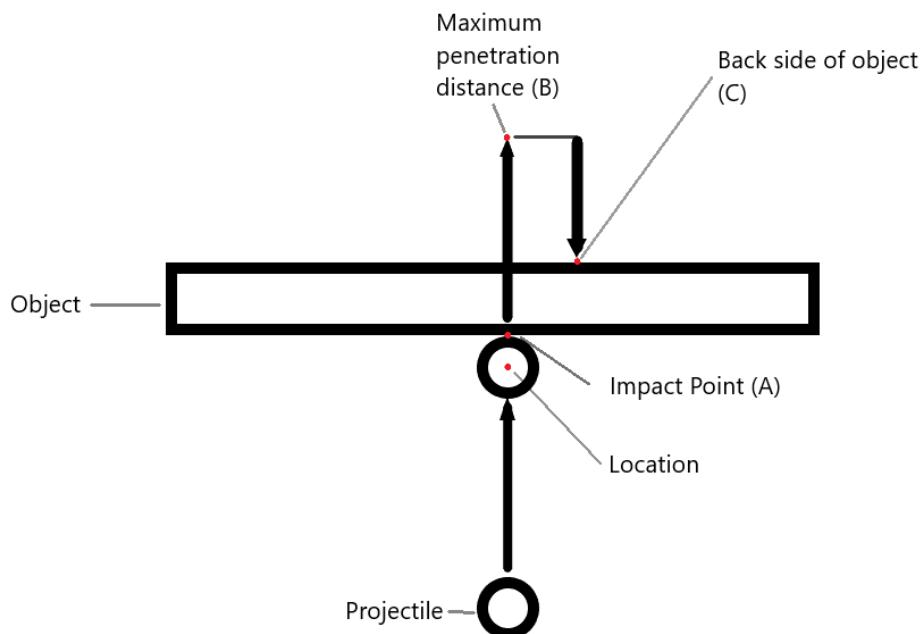
Na ovaj način ne šaljemo nikakve podatke o stvaranju novog projektila, njegovi pozicijski podaci se šalju ovisno o njegovim mrežnim postavkama.

Dakle, pri preklapanju projektila sa objektom, na projektilu se aktivira događaj preklapanja, tada spremamo podatke o preklapanju, treba primijetiti da se nadalje kod izvodi samo na serverskoj strani.



Slika 105 Događaj preklapanja

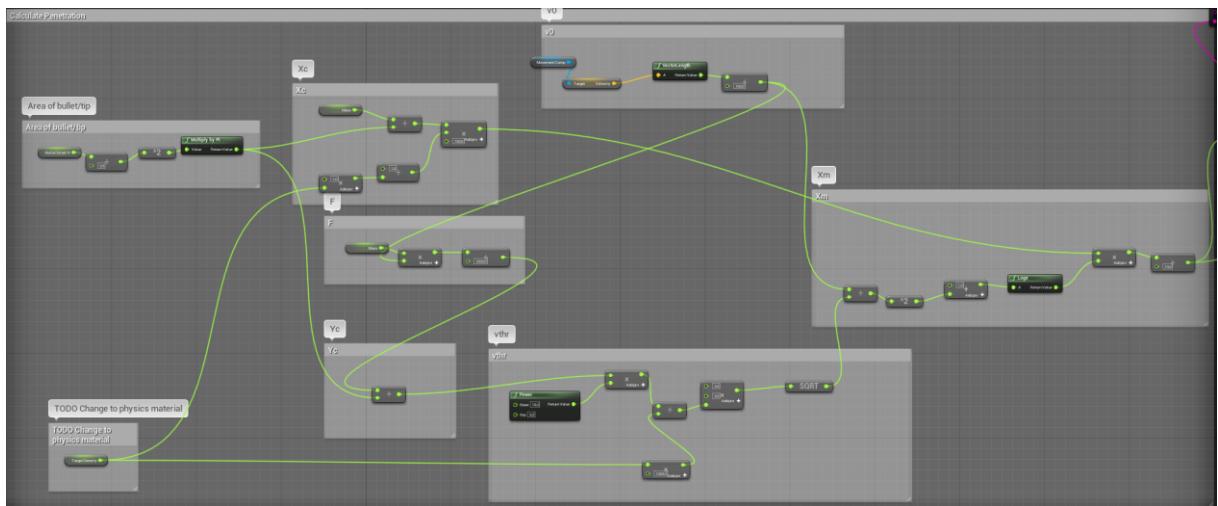
Ovdje je potrebno promotriti neke od točaka koje će nam biti potrebne za razne izračune.



Slika 106 Točke potrebne za izračune, vlastita izrada

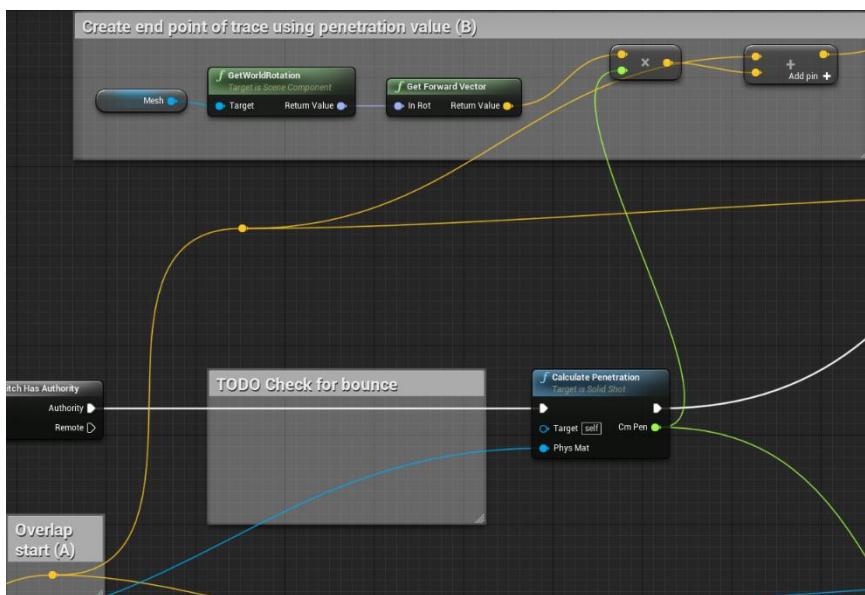
- A je točka kolizije (eng. Impact Point), ona nam označuje točku koju je projektil inicijalno udario.
- B je točka maksimalnog probijanja, odnosno ako može stići do nje, smatramo da je probio objekt.
- C je točka na poleđini objekta kojeg probijamo, služi za izračun brzine nakon probijanja.

Pošto pri preklapanju odmah znamo točku A, sada je potrebno pronaći točku B izračunom maksimalnog probijanja. Ovo se radi u zasebnoj funkciji radi čistoće koda. Ova metoda zasnovana je na pojednostavljenoj Pizardovoj metodi za projektile srednje brzine [115]. Ovdje uzimamo u obzir površinu vrha metka, brzinu metka, njegovu težinu te materijal objekta kojeg probijamo (zasad su svi objekti od čelika). Moguće je ovo unaprijediti preciznijim formulama [116] ali za potrebe ovog projekta ovo je zadovoljavajuće.



Slika 107 Izračun maksimalnog probijanja

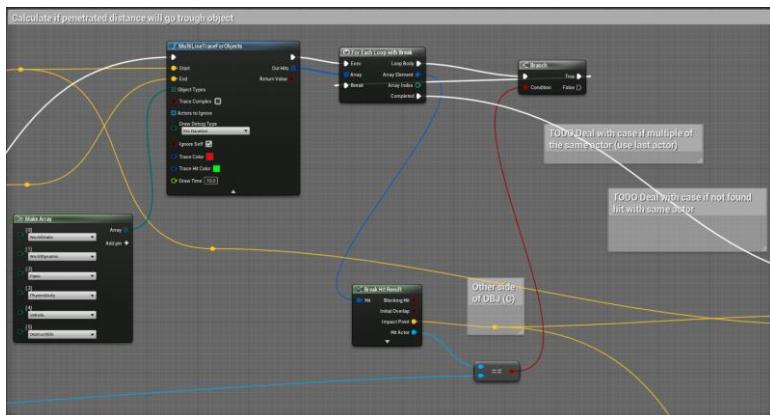
Sada možemo izračunati točku maksimalnog probijanja, korištenjem smjera kretanja metka, što pomnožimo sa dobivenom vrijednošću te dodamo točci A.



Slika 108 Izračun točke maksimalnog probijanja

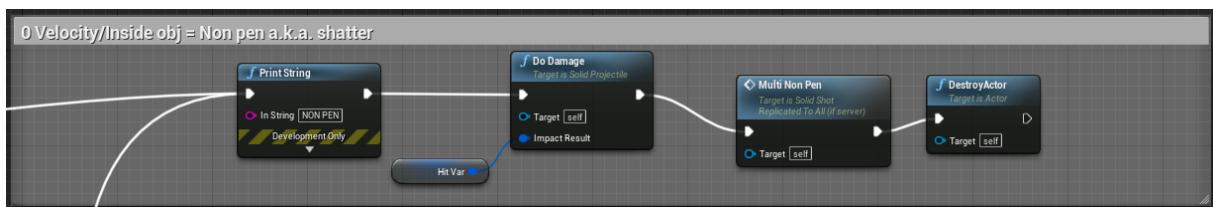
Da bi pronašli točku C, koristiti ćemo više-linijski trag, koji će nam vratiti polje pogodaka objekata s kojima je ušao u koliziju između B i A. Kada u tom polju pronađemo naš

objekt, ispitujemo da li je lokacija pogotka ista početnoj lokaciji, odnosno točci B. Ovo se radi jer se kolizija unutar objekta odmah aktivira, što znači da nismo probili objekt. Treba spomenuti da može doći do slučaja gdje postoji praznina između A i B, koju bi projektil mogao probiti, ali zbog dizajna sustava smatra se da ne može, ovo se može izbjegić uzimanjem samo zadnje kolizije sa oblikom.



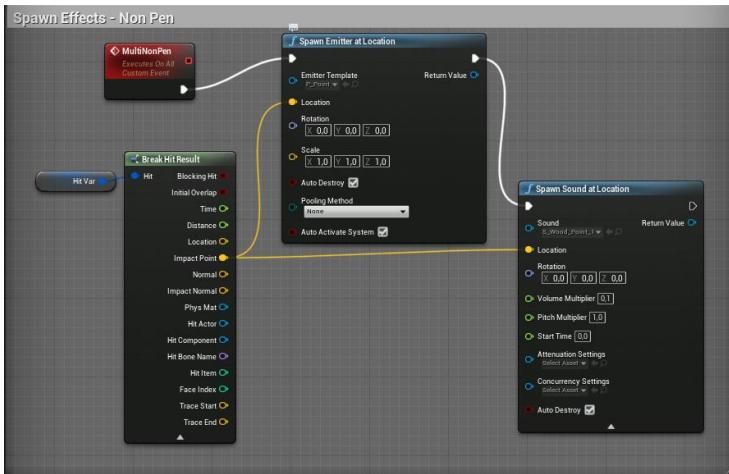
Slika 109 Pronalaženje točke na poleđini objekta

U slučaju da nismo probili objekt, njemu treba zadati štetu, obavijestiti klijenta te uništiti projektil.



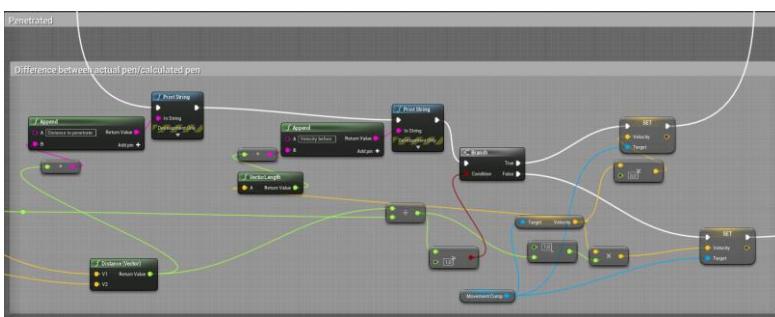
Slika 110 Slučaj ne probijanja objekta

Pri obavještavanju klijenta stvaramo vizualne efekte i zvuk. Ovo stvara puno manje informacija od prije navedene metode stvaranja projektila.



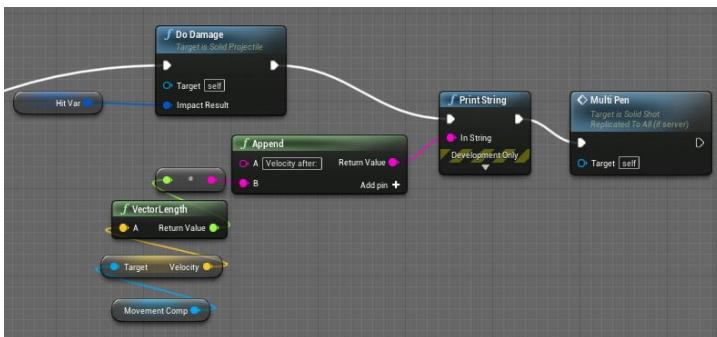
Slika 111 Stvaranje efekata na klijentskoj strani

U slučaju da smo probili objekt, moramo projektlu smanjiti brzinu, to radimo prema omjeru maksimalnog probijanja sa udaljenošću točaka A i C. Omjer množimo sa brzinom, te ako je ona jednaka nuli, opet pokrećemo uništavanje projektila.



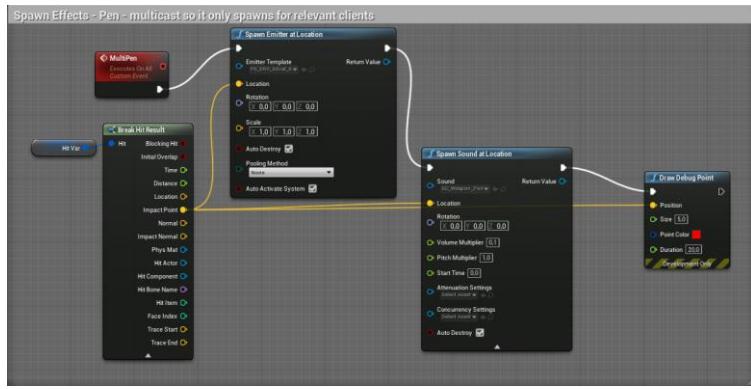
Slika 112 Slučaj probijanja objekta

Ako je projektil nastavio izvan objekta, tada moramo njemu zadati štetu i obavijestiti klijenta.



Slika 113 Nastavak projektila

Pri obaviještavanju klijenta ponovo stvaramo vizualni efekt i zvuk, za pomoć pri vizualizaciji stvara se i crvena točka, ali treba spomenuti da je ona na klijentskoj strani, te može biti neprecizna. Preciznost sustava moguće je očitati iz zapisnika na ekranu.



Slika 114 Efekti probijanja na korisničkoj strani

## **3.6. Funkcionalnost igre i meča**

Cilj ovog poglavlja je primarno dizajnirati umreženu igru sljedeće generacije, sa dizajnom primjerenim za procijenjeni broj igrača, te implementirati temeljne značajke kako bi ju mogli testirati. Kao što smo spomenuli na početku ovog rada, postoje dvije dominantne vrste umreženih igara, one koje su temeljene na mečevima, te one temeljene na trajnim svjetovima.

Osobno smatram da su neke od najvažnijih umreženih igara temeljene na trajnim svjetovima, prvo bitno „San Andreas : Multi Player Role Playing“ kroz svoju društvenu simulaciju, te „DayZ“ kroz svoju apokaliptičku simulaciju. Ultimativno smatram da se baš u igrama ovakvog tipa skriva idealan dizajn. Ali su ovakve igre iznimno teške za dizajnirati i usavršiti, te je vrijeme potrebno za njihov dostačni razvoj jednostavno preveliko za ovaj rad, na primjer igra Star Citizen, trenutno najnaprednija igra ove vrste, je u razvoju je od 2011.g. [117] [118].

Zbog ovoga, dizajnirati će se igra temeljena na mečevima. Glavna inspiracija za dizajn je modifikaciji za „Battlefield 2“, zvana „Project Reality“, koju osobno smatram vrhuncem dizajna umreženih igara temeljenih na mečevima sa visokim brojem igrača.

### **3.6.1.Dizajn funkcionalnosti igre i meča**

Kao što smo spomenuli na samom početku, interakcija između igrača je najveća prednost umreženih igara, najkvalitetnija vrsta interakcije je surađivanje (eng. Teamwork), ona je vrlo bliska temeljnoj ljudskoj potrebi za udruživanjem i preživljavanjem.

Kreativnost je druga ljudska potreba važna pri dizajnu igara. Ona se osigurava kroz davanje alata igračima, s kojima oni oblikuju tijek igre po svojoj želji. Ovo možemo zamisliti kao dizajniranje pješčanika (eng. Sandbox), dakle stvaramo okruženje u kojem igrač iskazuje kreativnost kroz korištenje alata uz ograničenja svijeta, sa ciljem pobjede. Ovaj pješčanik treba dizajnirati oko dostupnog broja igrača, kao što smo utemeljili u prijašnjim poglavljima, to je trenutno 200, a u budućnosti do otprilike 1800.

Držati ćemo se i nekih principa koji čine popularne igre [16], dakle želimo jasne uvjete pobjede, sa ograničenim vremenom trajanja, uz eskalaciju bitke i sustav progresije. Temeljno radi se o igri gdje dva tima međusobno pokušavaju dominirati istim svijetom.

### **3.6.1.1. Uvjet pobjede**

Uvjet pobjede mora biti jednostavan i jasan svim igračima. Temelji se na sustavu bodova, svaki tim kroz kontrolu točaka u svijetu dobiva bodove utjecaja, prvi tim koji skupi dovoljno takvih bodova pobjeđuje. Način na koji će igrači osigurati te kontrolne točke je na njima.

Točke u svijetu organizirane su u stablo, koje počinje u bazi jednog tima, širi se kroz svijet, i završava u bazi drugog tima. Ako se veza između baze i točke prekine zauzimanjem neke točke između njih, tada se ne skupljaju bodovi za taj tim.

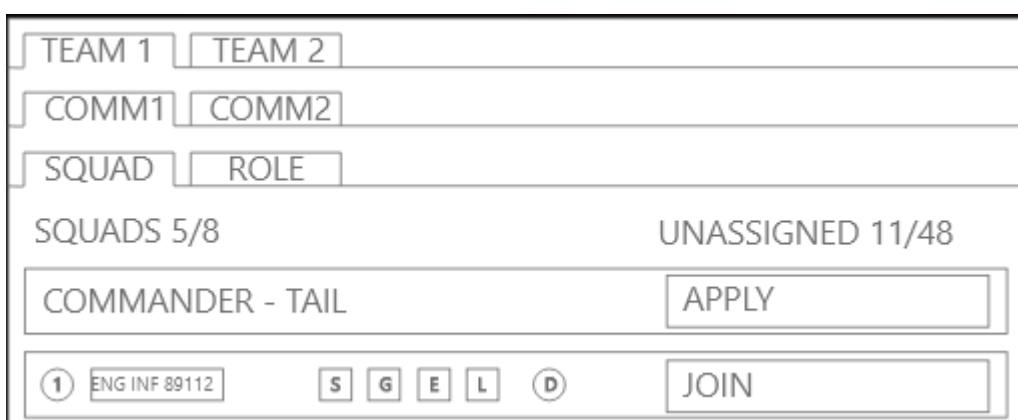
### **3.6.1.2. Udruživanje kroz lanac zapovijedanja**

Da bi poboljšali udruživanje, interakciju te organizaciju i tijek igre, uvesti ćemo koncept lanca zapovijedanja.

Na vrhu lanca stoji zapovjednik, njegova uloga je primarno orkestracija bitke, kroz zadavanje zapovijedi.

Ispod zapovjednika stoje odredi, kojima upravlja voditelj odreda, dok ispod njega stoje igrači. Uloga voditelja odreda je implementacija zapovjednikovih naredbi kroz zadavanje svojih naredbi. Dok je uloga igrača provedba naredbi voditelja odreda.

Jedan zapovjednik može efektivno upravljati sa 10 voditelja odreda, te tako i jedan voditelj odreda može upravljati sa 10 igrača. Treba napomenuti da ako bi tehnološki mogli ostvariti veći broj igrača, trebalo bi omogućiti više zapovjednika po timu u jednoj igri.



Slika 115 Koncept za izbornik odabira odreda,vlastita izrada (RedeployMenue)

Zapovjednik može odredima dati naredbe (a tako i voditelj odreda svom odredu), slične kao pri AI:

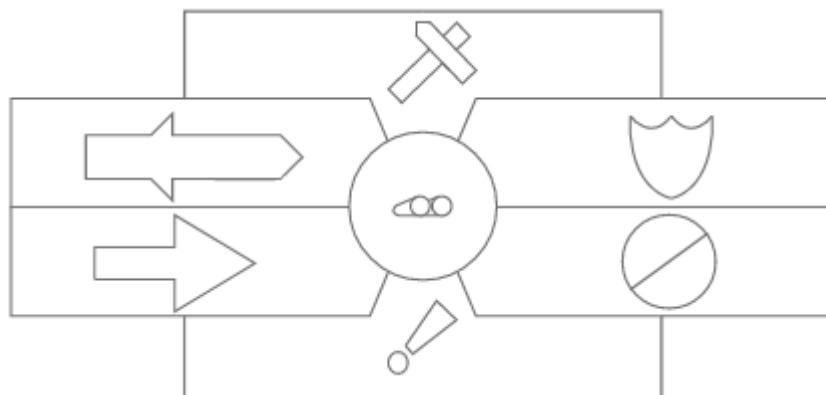
- Kreni do označene pozicije
- Zaustavi se
- Brani ovu poziciju
- Napadni ovu poziciju
- Sagradi
- Povuci se
- Regрутiraj (Ponovo stvori mrtve članove odreda)

Ovo bi se dalje moglo proširiti sa ulančavanjem naredbi, na primjer zadavanjem jedne naredbe, te pritiskom tipke „Shift“ te zadavanje sljedeće, slično kao i u „Company of Heroes 2“.

Zbog kompleksnosti određivanja taktike i naredbi potreban je sustav za glasovnu komunikaciju, na lokalnoj – prostornoj razini, na razini odreda, te na razini zapovjednika.

Nadalje, za lakše postizanje strateških ciljeva, mogli bi imati odrede umjetne inteligencije, kojima zapovjednik može upravljati. Slijedi da bi i odred igrača mogao imati neke članove koji su AI, za obavljanje raznih naredbi.

Nadovezano na AI, mogli bi iskoristiti značajku preuzimanja AI da se odmah vratimo u akciju ako smo umrli.



Slika 116 Koncept za izbornik naredbi, vlastita izrada (FullView – Commander)

Ovaj sustav, u sklopu sa kontrolnim točkama, pruža koncentriranu igru, sa širokim spektrom taktičkih mogućnosti za sve igrače u lancu. Dakle svaki igrač za sebe odlučuje kako će izvesti danu naredbu, ali svi ultimativno imaju isti cilj.

### **3.6.1.3. Motiviranje sustavom bodovanja**

Kako bi osigurali da je prosječni igrač motiviran za izvođenje naredbi, koristiti ćemo sustav bodovanja izvođenja naredbi, nazvan strateški, taktički i operacionalni bodovi:

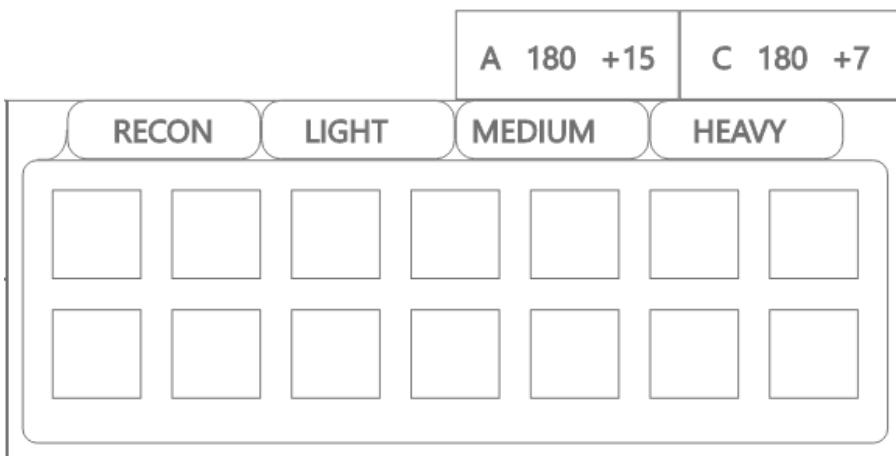
- Zauzimanjem točaka u svijetu zapovjednik dobiva strateške bodove.
- Pri završetku izvođenja pojedine naredbe zapovjednika, voditelj odreda dobiva taktičke bodove.
- Pri završetku izvođenja pojedine naredbe voditelja odreda, igrač dobiva operacionalne bodove.

Ovi bodovi mogu se iskoristiti za alate igranja, to mogu biti vozila, oružja, vrste odreda i slično, dakle:

- Zapovjednik iz bazena alata odabire alate koje želi da njegov tim koristi, te ih stavlja u strateški bazu, odabir alata košta strateške bodove.
- Voditelj odreda iz strateškog bazena odabire alate koje želi da njegov odred koristi, te ih stavlja u svoj taktički bazu, odabir alata košta taktičke bodove.
- Pojedini igrač u odredu iz taktičkog bazena odabire alate koje želi osobno koristiti, te ga to košta operacionalne bodove.

Alati u sustavu se dijele u faze eskalacije:

- Izvidnička faza - lagana transportna vozila, izvidnički zrakoplov, pješadijsko topništvo.
- Mehanizirana faza – lagana oklopna vozila, borilački zrakoplovi, protutenkovsko topništvo
- Faza srednje teškog oklopa – oklopna vozila, bliska zračna podrška, teško topništvo
- Faza teškog oklopa – teška oklopna vozila, bombarderi i slično.



Slika 117 Koncept za izbornik sustava eskalacije, vlastita izrada

Ovaj sustav osigurava eskalaciju bitke kroz tijek vremena. Inspiracija za ovaj sustav je igra „Company of Heroes 2“ sa svojim sustavom eskalacije, te igra „Crysis 1“ i „Counter Strike: Source“ sa svojim sustavom kupnje oružja tijekom igre.

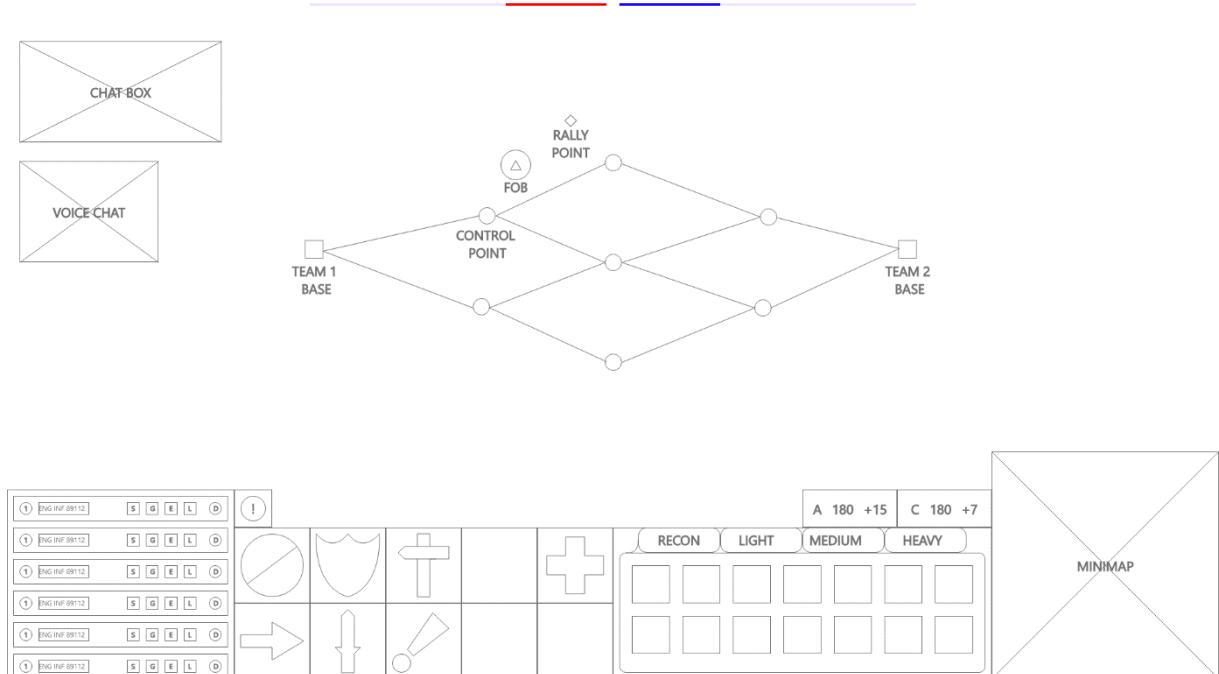
Nadalje ovaj sustav bi se mogao iskoristiti i u svrhu sustava progresije, naime pošto želimo zadržati naše igrače, periodički ih nagradujemo unaprijeđenima za njihove alate, na primjer bolja oprema i slično. Dakle ukupan broj bodova koji je igrač zaradio tijek jednog meča zbraja se u progresijske bodove, te se njima može trajno otključati mogućnost kupovanja nadogradnje. A tijekom meča se zarađeni bodovi mogu trošiti na otključane mogućnosti. Inspiracija za ovaj sustav je igra „Call Of Duty 4“, gdje se progresija prvi puta razvila u značajan sustav.

### 3.6.1.4. Logistički sustav

Kako bi se osigurao tijek igre, te frontalna linija bitke, potreban je sustav za ponovno stvaranje igrača koji su umrli. To će se ostvariti pomoću prednjih operativna baza (eng. forward operating base), koje igrač može izgraditi . Ovo se postiže već prije implementiranim objektima koji se mogu izgraditi.

Kako bi održali realizam, za njihovu izgradnju i korištenje bili bi potrebni logistički resursi, koji se stvaraju samo u bazi pojedinog tima. To su fizički objekti koji se konzumiraju izgradnjom pojedinih objekata. Radi lakšeg upravljanja i naređivanja, bilo bi povoljno logističke uloge ostvariti preko AI. Ovo bi dovodilo do zanimljivih taktičkih situacija gdje bi se mogle napadati, te tako i štititi logističke rute.

Na taktičkoj razini trebalo bi omogućiti voditeljima odreda da izgrađuju okupljalista (eng. Rally point), na kojima se igrači odreda ponovo mogu stvoriti, za lokalno održavanje frontalne linije bitke.



Slika 118 Prikaz zapovjednika, vlastita izrada - CommandingView-Commander

### 3.6.2.Implementacija funkcionalnosti igre i meča

Zbog kompleksnosti opisanih funkcionalnosti igre i meča, implementirati će se samo temeljne značajke važne za tijek igre, to su uvjeti pobjede i kontrolne točke.

Postoji nekolicina temeljnih klasa koje služe kao podloga za funkcionalnost igre. Na vrhu stoji svijet, on u sebi sadrži sve druge funkcionalnosti igre. Svijet ovog projekta dostupan je na putanji „Content/TakeoverPrototype/Takeover\_Medium“. Svijet se sastoji od:

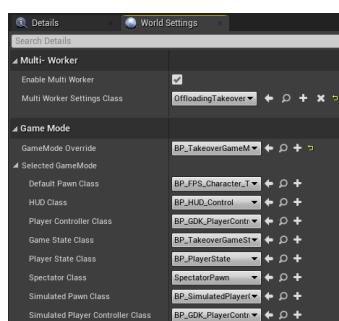
- Objekata i terena
- Pravila igre (eng. Game Mode)
- Više-serverske konfiguracije

Objekti su klase poput stvaratelja AI, upravljač vozila, uništivi objekti i sva ostale instance sličnih klasa. Teren je objekt koji predstavlja „zemlju“ ili „pod“ našeg svijeta, kombinira se sa deformacijom terena za bolje iskustvo.

Pravila igre se sastoje od:

- Lika kojim klijent upravlja
- UI klase
- Upravljača lika
- Klase stanja igre
- Klase stanja igrača
- Spektorske klase
- Klase simuliranog lika
- Klase upravljača simuliranog lika

Sama klasa pravila igre nalazi se na putanji „.../BP\_TakeoverGameMode“, dok se klasa stanja igre nalazi na „.../BP\_TakeoverGameState“.



Slika 119 Postavke svijeta

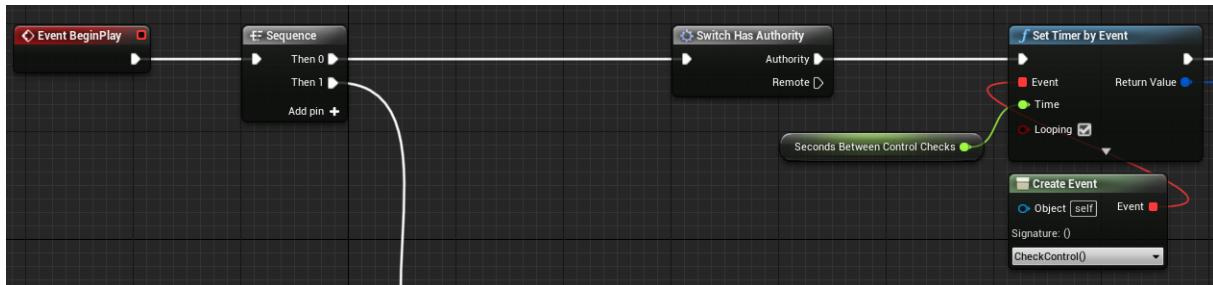
### 3.6.2.1. Kontrolne točke

Kontrolne točke su temeljni dio funkcionalnosti igre. Njihovo zauzimanje je zapravo uvjet pobjede. Trenutna implementacija je bazirana na kontrolnim točkama s kojima dolazi Spatial OS projekt, te samo modificirana za rad sa implementiranim likovima, jer su za potrebe ovog projekta te kontrolne točke zadovoljavajuće.

One se baziraju na stanjima:

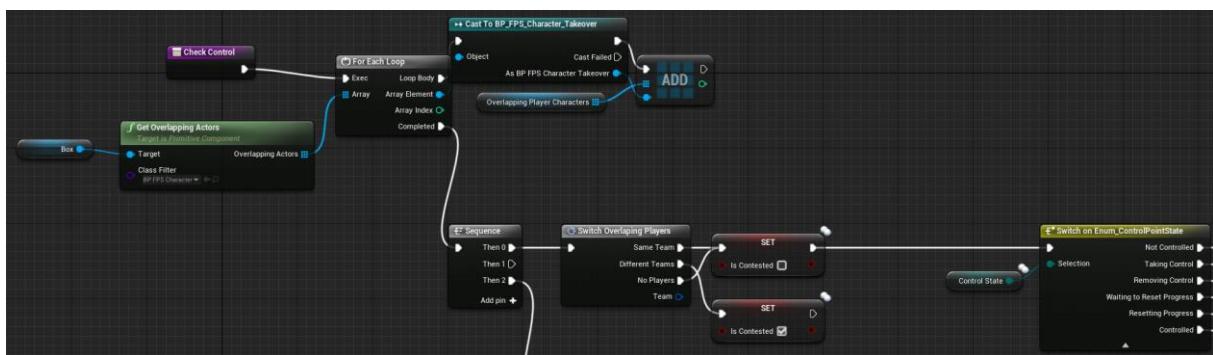
- Neutralno i neutralizira se
- Zauzeto i zauzima se
- Resetiranje i čekanje na resetiranje

Na početku njihove obrade zadaje se ponavljajuća funkcija koja provjerava zauzetost točaka.



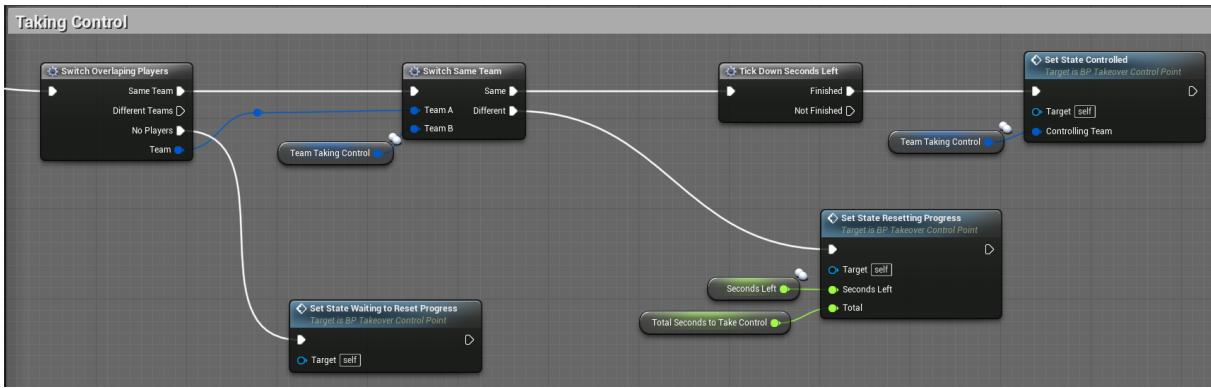
Slika 120 Provjeravanje kontrole

Pri provjeri zauzetosti zapravo se postavlja stanje pojedine točke.



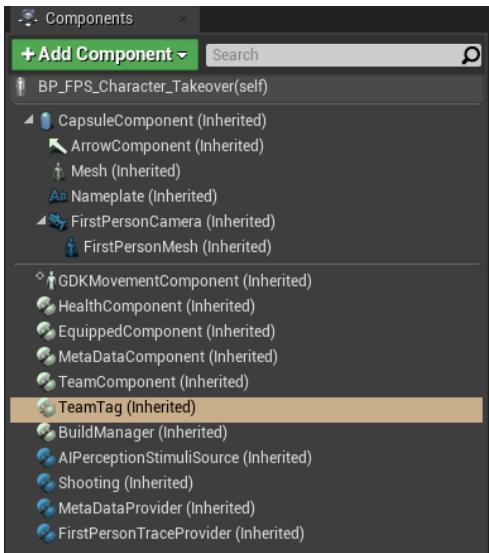
Slika 121 Postavljanje stanja točaka

Ova logika ovisi o broju igrača unutar kontrolne točke i kojem timu pripadaju.



Slika 122 Logika zauzimanja kontrolne točke

Ovo je moguće zbog komponente tima koju svaki igrač posjeduje.



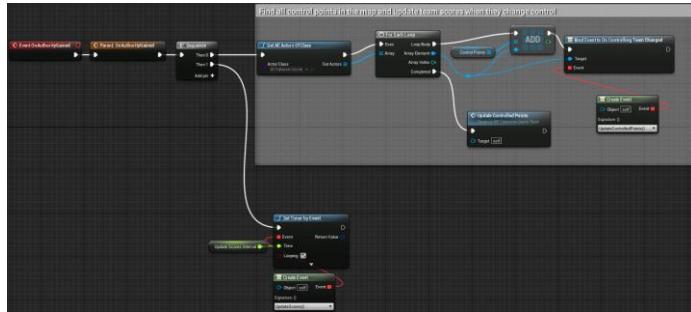
Slika 123 Komponenta tima na liku

### 3.6.2.2. Uvjeti pobjede

Uvjeti pobjede se implementiraju kroz klasu stanja igre. Ona zapravo služi za promjenu istih, stanja igre su dakle:

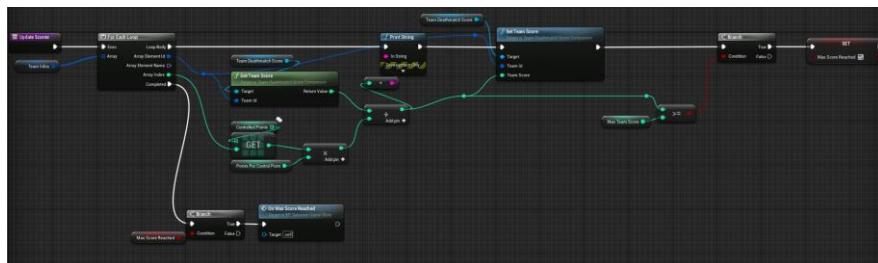
- Stanje prije igre (eng. Pre Game)
- Stanje tijekom igre (eng. In Game)
- Stanje nakon igre (eng. Post Game)

Dakle, na početku igre se inicijalizira meč, pronalaze se sve kontrolne točke, te se zabilježavaju u varijablu, te se poziva ponavljajuća funkcija za njihovu obradu (registraciju stanja), napisljeku se poziva ponavljajuća funkcija za ažuriranje bodovanja.



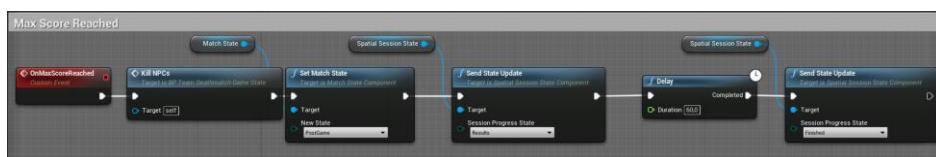
Slika 124 Inicijalizacija meča

Pri ažuriranju bodova provjerava se stanje svih kontrolnih točki, te se to zbraja sa brojem eliminacija protivnika.



Slika 125 Ažuriranje bodova

Kada se postigne maksimalni broj bodova, igra se prebacuje u stanje nakon igre i završava.

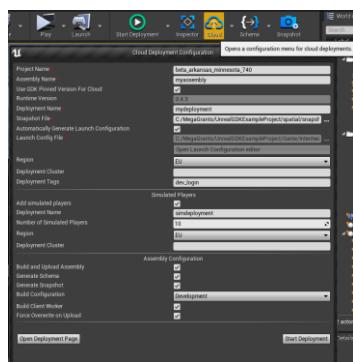


Slika 126 Završetak igre

### 3.6.2.3. Testiranje servera

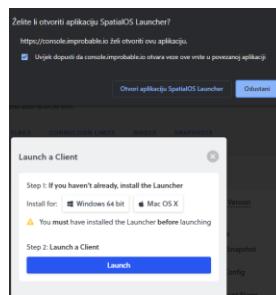
Da bi smo mogli testirati našu igru u pravome okruženju, potrebno ju je staviti u oblak. Koristiti ćemo besplatni rang servisa, koji nam nudi do 200 igrača na jednom serveru [62].

Proces započinje konfiguracijom servera, nabavlja se ime projekta sa Improbable konzole [119], te zadaje ime učitanom sklopu (eng. Assembly) i učitanom rasporedu (eng. Deployment). Nadalje je moguće uključiti simulirane igrače, ali smo limitirani na samo njih 10.



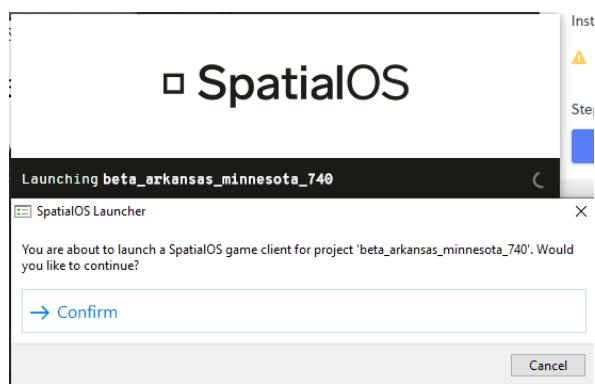
Slika 127 Konfiguracija servera

Nakon što smo igru učitali na server, možemo ju pokretati na lokalnom računalu kroz prije navedenu konzolu.



Slika 128 Pokretanje u konzoli

Ovo omogućuje bilo kome da skinie igru te ju instalira i igra.



Slika 129 Lokalno pokretanje igre

## 4. Zaključak

Umrežene igre, kao temeljno jedinstven medij, dopuštaju interakciju ljudi u simuliranom svijetu. Trenutna tehnologija ograničava broj interakcija te složenost i detaljnost tog svijeta. U ovom smo radu dokazali da je moguće probiti ograničenja trenutne generacije, da su mogući svjetovi sa stotinama igrača, uz veoma detaljna i dinamična okruženja.

Započeli smo postavljanjem teorijskih temelja za obije tehnike umreženja, sinkronizacijom stanja te deterministički diskretnim korakom. Umreženje igrača postigli smo korištenjem više-serverske arhitekture u Spatial OS i Unreal Engine 4, koja obrađuje jednu simulaciju sa više računala. Efikasnost umreženja doveli smo do najveće moguće razine koristeći dinamičko upravljanje interesom.

Detaljnost i dinamičnost okruženja postigli smo raznim umreženim značajkama. Kroz vozila prikazali smo temeljni problem umreženja, te ga riješili na više načina. Sa algoritmom za stabiliziranje fizičkih objekata postigli smo interaktivne fizičke objekte u umreženom okruženju. Njih smo kombinirali sa uništivim objektima, objektima koji se mogu izgraditi i deformacijom terena za istinito dinamično okruženje. Naposljetku, sve objekte obradili smo uz sustav balistike, što je dalo još jedan sloj realizma našoj igri.

Primjereno novoj specifikaciji igara sljedeće generacije, dizajnirali smo funkcionalnost meča koja rješava probleme udruživanja i motivacije korištenjem lanca zapovijedanja i sustava bodovanja. Naposljetku smo sve testirali na pravom serveru kako bi potvrdili funkcionalnost.

Osobno smatram da je ovaj rad dostatna podloga za produkciju umrežene igre bilo koje vrste, jer su mogućnosti koje se otvaraju dodatnim performansama iznimno široke.

# Popis literature

- [1] Microsoft, »Visual Studio 2019 version 16.10 Release Notes,« 15 06 2021. [Mrežno]. Available: <https://docs.microsoft.com/en-us/visualstudio/releases/2019/release-notes>. [Pokušaj pristupa 24 06 2021].
- [2] S. Horvath, »The Imagination Engine: Why Next-Gen Videogames Will Rock Your World,« 17 05 2012. [Mrežno]. Available: <https://www.wired.com/2012/05/ff-unreal4/>. [Pokušaj pristupa 24 06 2021].
- [3] K. ORLAND, »Unreal Engine 4 now available as \$19/month subscription with 5% royalty,« 19 03 2014. [Mrežno]. Available: <https://arstechnica.com/gaming/2014/03/unreal-engine-4-now-available-as-19month-subscription-with-5-royalty/>. [Pokušaj pristupa 24 06 2021].
- [4] Epic Games, »Blueprint Visual Scripting,« 2021. [Mrežno]. Available: <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/>. [Pokušaj pristupa 24 06 2021].
- [5] Improbable, »Multiplayer Networking,« 2021. [Mrežno]. Available: <https://www.improbable.io/multiplayer-networking>. [Pokušaj pristupa 24 06 2021].
- [6] Improbable, »SpatialOS Game Development Kit (GDK) for Unreal,« 2021. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/lang-en/docs>. [Pokušaj pristupa 24 06 2021].
- [7] Epic Games, »Unreal Engine on github,« 2021. [Mrežno]. Available: <https://www.unrealengine.com/en-US/ue4-on-github?sessionInvalidated=true>. [Pokušaj pristupa 24 06 2021].
- [8] Improbable, »The SpatialOS GDK for Unreal Plugin,« 03 11 2020. [Mrežno]. Available: <https://github.com/spatialos/UnrealGDK>. [Pokušaj pristupa 24 06 2021].
- [9] Improbable, »Set up the fork and plugin,« 2021. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/get-started-2-set-up-the-fork-and-plugin>. [Pokušaj pristupa 24 06 2021].
- [10] Atlassian, »Sourcetree,« 2021. [Mrežno]. Available: <https://www.sourcetreeapp.com/>. [Pokušaj pristupa 24 06 2021].
- [11] J. Petanjek, »Next Generation FPS,« 20 06 2021. [Mrežno]. Available: <https://github.com/jpetanjek/NG-FPS>. [Pokušaj pristupa 24 06 2021].

- [12] GitHub, »Install Git,« 2021. [Mrežno]. Available: <https://github.com/git-guides/install-git>. [Pokušaj pristupa 24 06 2021].
- [13] Adobe, »Adobe XD,« 2021. [Mrežno]. Available: <https://www.adobe.com/products/xd.html>. [Pokušaj pristupa 24 06 2021].
- [14] C. Crawford, »Chris Crawford Dragon Speech "I Had a Dream",« 14 06 2017. [Mrežno]. Available: <https://www.youtube.com/watch?v=CBj4S24074>. [Pokušaj pristupa 24 06 2021].
- [15] A. Millard, »Putting The Players Back In Multiplayer,« 11 06 2019. [Mrežno]. Available: <https://youtu.be/39DqD38J-I0>. [Pokušaj pristupa 24 06 2021].
- [16] T. Wilde, »Can we predict what games will be hits on Twitch?,« 23 01 2021. [Mrežno]. Available: <https://www.pcgamer.com/can-we-predict-what-games-will-be-hits-on-twitch/>. [Pokušaj pristupa 24 06 2021].
- [17] Chris Battle(non)sense, »Valorant Netcode Analysis,« 25 07 2020. [Mrežno]. Available: <https://www.youtube.com/watch?v=ftC1RpI8mtg>. [Pokušaj pristupa 24 06 2021].
- [18] Epic Games, »Networking in 4.20: The Replication Graph | Feature Highlight | Unreal Engine Livestream,« 19 07 2018. [Mrežno]. Available: <https://youtu.be/CDnNAAzgltw?t=575>. [Pokušaj pristupa 24 06 2021].
- [19] Chris Battle(non)sense, »Fortnite beats PUBG 's Terrible 17Hz Netcode,« 25 10 2017. [Mrežno]. Available: [https://www.youtube.com/watch?v=SwZ\\_NUruGTM&t=145s](https://www.youtube.com/watch?v=SwZ_NUruGTM&t=145s). [Pokušaj pristupa 24 06 2021].
- [20] Chris Battle(non)sense, »Fortnite beats PUBG 's Terrible 17Hz Netcode,« 17 10 2017. [Mrežno]. Available: [https://www.youtube.com/watch?v=SwZ\\_NUruGTM&t=373s](https://www.youtube.com/watch?v=SwZ_NUruGTM&t=373s). [Pokušaj pristupa 24 06 2021].
- [21] DICE, »New Update Addressing “Netcode” Rolling Out,« 2016. [Mrežno]. Available: <https://battlelog.battlefield.com/bf4/news/view/new-netcode-update-rolling-out/>. [Pokušaj pristupa 24 06 2021].
- [22] Chris Battle(non)sense, »BF 120Hz Tickrate: (How) Does it Work?,« 24 05 2015. [Mrežno]. Available: <https://www.youtube.com/watch?v=7nO9bZm8ceY&t=175s>. [Pokušaj pristupa 24 06 2021].
- [23] Bohemia Interactive, »Real Virtuality,« 26 04 2021. [Mrežno]. Available: [https://community.bistudio.com/wiki/Real\\_Virtuality](https://community.bistudio.com/wiki/Real_Virtuality). [Pokušaj pristupa 24 06 2021].
- [24] Bohemia Interactive, »Enfusion,« 06 02 2021. [Mrežno]. Available: <https://community.bistudio.com/wiki/Enfusion>. [Pokušaj pristupa 24 06 2021].

- [25] L. Johnson, »Why is it so hard to stop cheating in videogames?,« 08 12 2016. [Mrežno]. Available: <https://www.pcgamer.com/why-is-it-so-hard-to-stop-cheating-in-videogames/>. [Pokušaj pristupa 24 06 2021].
- [26] Day Break Games, »PLANETSIDE 2 FOR THE PC - FAQ,« 2021. [Mrežno]. Available: <https://www.planetside2.com/faq>. [Pokušaj pristupa 24 06 2021].
- [27] Ookla, »Speedtest Global Index,« 2021. [Mrežno]. Available: <https://www.speedtest.net/global-index#mobile>. [Pokušaj pristupa 24 06 2021].
- [28] Clouvider, »10Gbps Dedicated Servers,« 2021. [Mrežno]. Available: <https://www.clouvider.co.uk/10gbps-dedicated-servers/>. [Pokušaj pristupa 24 06 2021].
- [29] Epic Games, »Unreal Engine 5 Revealed! | Next-Gen Real-Time Demo Running on PlayStation 5,« 13 05 2020. [Mrežno]. Available: <https://www.youtube.com/watch?v=qC5KtatMcUw>. [Pokušaj pristupa 24 06 2021].
- [30] G. Fiedler, »Networking for Physics Programmers,« 04 10 2018. [Mrežno]. Available: <https://www.youtube.com/watch?v=Z9X4lysFr64&t=3297s>. [Pokušaj pristupa 24 06 2021].
- [31] G. Fiedler, »Deterministic Lockstep,« 29 09 2014. [Mrežno]. Available: [https://gafferongames.com/post/deterministic\\_lockstep/](https://gafferongames.com/post/deterministic_lockstep/). [Pokušaj pristupa 24 06 2021].
- [32] T. Sweeney, »On Netcode in Unreal Engine,« 21 07 1999. [Mrežno]. Available: <https://docs.google.com/document/d/1KGLbEfHsWANTTgUqfK6rkpFYDGvnZYjBN18sxq6LPY/edit>. [Pokušaj pristupa 24 06 2021].
- [33] R. Whitehead, »Is the architecture of SpatialOS itself based on ECS?,« 18 05 2018. [Mrežno]. Available: <https://forums.improbable.io/t/is-the-architecture-of-spatialos-itself-based-on-ecs/4126/3>. [Pokušaj pristupa 24 06 2021].
- [34] A. Noon, »Order from Chaos - Destruction in UE4 | GDC 2019 | Unreal Engine,« 26 03 2019. [Mrežno]. Available: <https://www.youtube.com/watch?v=iFKzXnliH50&t=273s>. [Pokušaj pristupa 24 06 2021].
- [35] P. B. Mark Terrano, »1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond,« 22 03 2001. [Mrežno]. Available: [https://www.gamasutra.com/view/feature/131503/1500\\_archers\\_on\\_a\\_288\\_network\\_.php](https://www.gamasutra.com/view/feature/131503/1500_archers_on_a_288_network_.php). [Pokušaj pristupa 24 06 2021].
- [36] D. Ratti, »Network Prediction Physics 1,« 13 05 2020. [Mrežno]. Available: <https://youtu.be/3HLvNu-j6eg?t=173>. [Pokušaj pristupa 24 06 2021].

- [37] G. Fiedler, »Snapshot Interpolation,« 30 09 2014. [Mrežno]. Available: [https://gafferongames.com/post/snapshot\\_interpolation/](https://gafferongames.com/post/snapshot_interpolation/). [Pokušaj pristupa 24 06 2021].
- [38] G. Fiedler, »Snapshot Compression,« 04 01 2015. [Mrežno]. Available: [https://gafferongames.com/post/snapshot\\_compression/](https://gafferongames.com/post/snapshot_compression/). [Pokušaj pristupa 24 06 2021].
- [39] G. Fiedler, »State Synchronization,« 05 01 2015. [Mrežno]. Available: [https://gafferongames.com/post/state\\_synchronization/](https://gafferongames.com/post/state_synchronization/). [Pokušaj pristupa 24 06 2021].
- [40] R. Whitehead, »Intimacy at scale: Building an architecture for density,« 02 06 2021. [Mrežno]. Available: <https://www.improbable.io/blog/intimacy-at-scale-building-an-architecture-for-density>. [Pokušaj pristupa 26 06 2021].
- [41] Epic Games, »Unreal Networking Architecture,« 2012. [Mrežno]. Available: <https://docs.unrealengine.com/udk/Three/NetworkingOverview.html#Physics>. [Pokušaj pristupa 24 06 2021].
- [42] SkippyFX, »TESTING MY PATIENCE — Hell Let Loose (Funny Moments),« 30 04 2021. [Mrežno]. Available: <https://www.youtube.com/watch?v=d3kBKFBoe-I&t=73s>. [Pokušaj pristupa 24 06 2021].
- [43] S. Duc, »WHAT MAKES APEX TICK: A DEVELOPER DEEP DIVE INTO SERVERS AND NETCODE,« 2021. [Mrežno]. Available: <https://www.ea.com/en-gb/games/apex-legends/news/servers-netcode-developer-deep-dive>. [Pokušaj pristupa 24 06 2021].
- [44] Epic Games, »Networking in 4.20: The Replication Graph | Feature Highlight | Unreal Engine Livestream,« 19 05 2018. [Mrežno]. Available: <https://www.youtube.com/watch?v=CDnNAAzgltw&t=4144s>. [Pokušaj pristupa 24 06 2021].
- [45] Unobtanium, »Unofficial Road to Dynamic Server Meshing - Tech Overview with Explanations,« 24 06 2020. [Mrežno]. Available: <https://robertsspaceindustries.com/spectrum/community/SC/forum/3/thread/road-to-dynamic-server-meshing-tech-overview-with->. [Pokušaj pristupa 24 06 2021].
- [46] Ambuaz, »Elder Scrolls Online info Mega Server,« 2014. [Mrežno]. Available: <https://elderscrollsonline.info/mega-server>. [Pokušaj pristupa 24 06 2021].
- [47] Star Vault, »Mortal Online 2 New Player FAQ,« 2020. [Mrežno]. Available: <https://www.mortalonline2.com/faq/>. [Pokušaj pristupa 24 06 2021].
- [48] Hadean Supercomputing, »Aether Engine Distributed Spatial Simulation,« 2020. [Mrežno]. Available: <https://hadean.com/spatial-simulation/>. [Pokušaj pristupa 24 06 2021].

- [49] Improbable, »Worker,« 04 09 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/glossary#section-worker>. [Pokušaj pristupa 24 06 2021].
- [50] Improbable, »World, entities, components,« 10 12 2019. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/world-entities-components>. [Pokušaj pristupa 24 06 2021].
- [51] Improbable, »Entity,« 10 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/glossary#section-entity>. [Pokušaj pristupa 24 06 2021].
- [52] Improbable, »Component,« 10 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/glossary#section-component>. [Pokušaj pristupa 24 06 2021].
- [53] Improbable, »Layers,« 06 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/layers>. [Pokušaj pristupa 24 06 2021].
- [54] Improbable, »Workers and load balancing,« 05 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/workers-and-load-balancing>. [Pokušaj pristupa 24 06 2021].
- [55] Improbable, »Authority,« 05 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/authority>. [Pokušaj pristupa 24 06 2021].
- [56] Improbable, »Actor handover,« 09 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/actor-handover>. [Pokušaj pristupa 24 06 2021].
- [57] Improbable, »Game client interest management,« 29 07 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/game-client-interest-management>. [Pokušaj pristupa 24 06 2021].
- [58] Improbable, »Operations,« 06 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/operations>. [Pokušaj pristupa 24 06 2021].
- [59] Improbable, »Persistence,« 04 09 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/glossary#section-persistence>. [Pokušaj pristupa 24 06 2021].

- [60] Improbable, »Schema and snapshots,« 27 06 2019. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/schema-and-snapshots>. [Pokušaj pristupa 24 06 2021].
- [61] Improbable, »GDC 2019 Sponsored Talk | Building Unreal worlds with SpatialOS | SpatialOS GDK for Unreal Engine,« 11 04 2019. [Mrežno]. Available: <https://www.youtube.com/watch?v=U3FHEBoN5Mo&t=577s>. [Pokušaj pristupa 24 06 2021].
- [62] Improbable, »Networking service pricing,« 12 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/managed-networking-services#section-networking-service-pricing>. [Pokušaj pristupa 24 06 2021].
- [63] Improbable, »Distributed physics without server boundaries,« 25 04 2017. [Mrežno]. Available: <https://www.improbable.io/blog/distributed-physics-without-server-boundaries>. [Pokušaj pristupa 24 06 2021].
- [64] Ayusharma, »IEEE Standard 754 Floating Point Numbers,« 15 03 2020. [Mrežno]. Available: <https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/>. [Pokušaj pristupa 24 06 2021].
- [65] Epic Games, »World Composition User Guide,« 2021. [Mrežno]. Available: <https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/LevelStreaming/WorldBrowser/>. [Pokušaj pristupa 24 06 2021].
- [66] Epic Games, »Unreal Engine 5 Noteworthy Changes,« 2021. [Mrežno]. Available: <https://docs.unrealengine.com/5.0/en-US/MigrationGuide/NoteworthyChanges/>. [Pokušaj pristupa 24 06 2021].
- [67] Epic Games, »Large World Coordinates,« 05 05 2021. [Mrežno]. Available: <https://github.com/EpicGames/UnrealEngine/blob/ue5-main/Engine/Source/Runtime/Core/Public/Misc/LargeWorldCoordinates.h>. [Pokušaj pristupa 24 06 2021].
- [68] Improbable, »Inspector,« 04 09 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/glossary#section-inspector>. [Pokušaj pristupa 24 06 2021].
- [69] Improbable, »Dynamic typebinding,« 01 10 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/dynamic-typebindings>. [Pokušaj pristupa 24 06 2021].
- [70] Epic Games, »RPCs,« 2021. [Mrežno]. Available: <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Networking/Actors/RPCs/>. [Pokušaj pristupa 24 06 2021].

- [71] Improbable, »Local deployment summary,« 11 09 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/local-deployment-workflow>. [Pokušaj pristupa 24 06 2021].
- [72] Improbable, »NetCullDistanceSquared,« 29 07 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/game-client-interest-management#section-net-cull-distance-squared>. [Pokušaj pristupa 24 06 2021].
- [73] Improbable, »NetCullDistanceFrequency,« 29 07 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/game-client-interest-management#section-net-cull-distance-frequency>. [Pokušaj pristupa 24 06 2021].
- [74] Improbable, »AlwaysInterested,« 29 07 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/game-client-interest-management#section-always-interested>. [Pokušaj pristupa 24 06 2021].
- [75] Improbable, »ActorInterestComponent,« 29 07 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/game-client-interest-management#section-actor-interest-component>. [Pokušaj pristupa 24 06 2021].
- [76] M. Zelina, »First try at expressing interest in actor components,« 15 04 2021. [Mrežno]. Available: <https://github.com/spatialos/UnrealGDK/commit/c1666215b469eeee24d720aeeee177490caf56dc>. [Pokušaj pristupa 24 06 2021].
- [77] J. Petanjek, »UE4 - FOV Based Replication - Dynamic Interest Management with Spatial OS,« 01 06 2021. [Mrežno]. Available: <https://www.youtube.com/watch?v=dEzemYdc2xY>. [Pokušaj pristupa 24 06 2021].
- [78] Improbable, »Offloading,« 12 08 2020. [Mrežno]. Available: <https://documentation.improbable.io/gdk-for-unreal/docs/5-offloading-in-game-tutorial>. [Pokušaj pristupa 24 06 2021].
- [79] Improbable, »Network configuration,« 06 2020. [Mrežno]. Available: <https://documentation.improbable.io/spatialos-overview/docs/network-configuration>. [Pokušaj pristupa 24 06 2021].
- [80] M. Maisak, »Hierarchical Task Network Planning AI,« 24 07 2020. [Mrežno]. Available: <https://maksmaisak.github.io/htn/#/>. [Pokušaj pristupa 24 06 2021].
- [81] Fandom, »Tactics (Origins),« 2021. [Mrežno]. Available: [https://dragonage.fandom.com/wiki/Tactics\\_\(Origins\)](https://dragonage.fandom.com/wiki/Tactics_(Origins)). [Pokušaj pristupa 24 06 2021].
- [82] Offworld industries, »OWI Enhanced Vehicle Movement,« 11 10 2019. [Mrežno]. Available:

[https://www.offworldindustries.com/docs/OWI\\_Enhanced\\_Vehicle\\_Movement\\_Documentation.pdf](https://www.offworldindustries.com/docs/OWI_Enhanced_Vehicle_Movement_Documentation.pdf). [Pokušaj pristupa 24 06 2021].

- [83] S. Pattuzzi, »Unreal Multiplayer Master: Video Game Dev In C++ Course,« 2021. [Mrežno]. Available: <https://www.gamedev.tv/p/unrealmultiplayer>. [Pokušaj pristupa 24 06 2021].
- [84] Epic Games, »Character Movement Component,« 2021. [Mrežno]. Available: <https://docs.unrealengine.com/4.26/en-US/InteractiveExperiences/Networking/CharacterMovementComponent/>. [Pokušaj pristupa 24 06 2021].
- [85] MazyModz, »MazyModz,« 2018. [Mrežno]. Available: <https://www.youtube.com/user/MazyModz>. [Pokušaj pristupa 01 05 2021].
- [86] J. Petanjek, »Vehicle Movement Component + Lag Compensation with INetworkPredictionInterface - Spatial OS - UE4.25,« 11 05 2021. [Mrežno]. Available: <https://www.youtube.com/watch?v=kljYVNgsRY>. [Pokušaj pristupa 24 06 2021].
- [87] Epic Games, »Minimizing Perceived Latency,« 2012. [Mrežno]. Available: <https://docs.unrealengine.com/udk/Three/NetworkingOverview.html#Minimizing%20Perceived%20Latency>. [Pokušaj pristupa 24 06 2021].
- [88] Epic Games, »Simulation,« 2012. [Mrežno]. Available: <https://docs.unrealengine.com/udk/Three/NetworkingOverview.html#Simulation>. [Pokušaj pristupa 24 06 2021].
- [89] NVidia, »Immediate mode,« 2017. [Mrežno]. Available: <https://forums.developer.nvidia.com/t/doc-of-api-immediate-mode/48873/7>. [Pokušaj pristupa 24 06 2021].
- [90] S. Pattuzzi, »Unreal Multiplayer Course - Section 4 - Krazy Karts,« 2017. [Mrežno]. Available: [https://github.com/UnrealMultiplayer/4\\_Krazy\\_Karts](https://github.com/UnrealMultiplayer/4_Krazy_Karts). [Pokušaj pristupa 24 06 2021].
- [91] Blue man, »Blue Man Vehicle Physics,« 20 07 2017. [Mrežno]. Available: <https://www.unrealengine.com/marketplace/en-US/product/blue-man-vehicle-physics?sessionInvalidated=true>. [Pokušaj pristupa 24 06 2021].
- [92] D. Ratti, »Network Prediction Physics 1,« 13 07 2020. [Mrežno]. Available: <https://www.youtube.com/watch?v=3HLvNu-j6eg>. [Pokušaj pristupa 24 06 2021].
- [93] Epic Games, »Unreal Engine 5 is now available in Early Access!,« 26 05 2021. [Mrežno]. Available: <https://www.unrealengine.com/en-US/blog/unreal-engine-5-is-now-available-in-early-access?sessionInvalidated=true>. [Pokušaj pristupa 24 06 2021].

- [94] J. Petanjek, »UE5 Network Prediction - instant control + server authoritative physics (Chaos),« 27 05 2021. [Mrežno]. Available: <https://www.youtube.com/watch?v=sPtAqvwQIkE>. [Pokušaj pristupa 24 06 2021].
- [95] D. Ratti, »Network Prediction Insights Part 1,« 18 03 2020. [Mrežno]. Available: [https://www.youtube.com/watch?v=\\_rdt-v1nFIY](https://www.youtube.com/watch?v=_rdt-v1nFIY). [Pokušaj pristupa 24 06 2021].
- [96] J. Petanjek, »Custom Server Authoritative Physics - accurate simulation without jitter - Spatial OS - UE4.25,« 25 04 2021. [Mrežno]. Available: <https://www.youtube.com/watch?v=oNLb4zOZbCM>. [Pokušaj pristupa 24 06 2021].
- [97] Epic Games, »Procedural Mesh,« 2021. [Mrežno]. Available: <https://docs.unrealengine.com/4.26/en-US/BlueprintAPI/Components/ProceduralMesh/>. [Pokušaj pristupa 24 06 2021].
- [98] CodingWithRus, »Mesh Deformation : Beginners Procedural Generation EP 7 - Unity3D,« 24 11 2020. [Mrežno]. Available: <https://youtu.be/5aXMMXPACpo>. [Pokušaj pristupa 24 06 2021].
- [99] G. Portelli, »Custom Struct Serialization for Networking in Unreal Engine,« 19 03 2016. [Mrežno]. Available: <http://www.aclockworkberry.com/custom-struct-serialization-for-networking-in-unreal-engine/>. [Pokušaj pristupa 24 06 2021].
- [100] J. Petanjek, »UE4 Terrain Deformation Demo + Spatial OS,« 23 04 2021. [Mrežno]. Available: [https://www.youtube.com/watch?v=0ns7Pm\\_6pSA](https://www.youtube.com/watch?v=0ns7Pm_6pSA). [Pokušaj pristupa 24 06 2021].
- [101] R. Schmidt, »Mesh Generation and Editing at Runtime in UE4.26,« 23 10 2020. [Mrežno]. Available: <http://www.gradientspace.com/tutorials/2020/10/23/runtime-mesh-generation-in-ue426>. [Pokušaj pristupa 24 06 2021].
- [102] J. L'Heureux, »The Art of Destruction in Rainbow Six: Siege,« 08 12 2017. [Mrežno]. Available: <https://www.youtube.com/watch?v=SjkQxowsL0I>. [Pokušaj pristupa 24 06 2021].
- [103] Epic Games, »Set Actor Enable Collision is not replicated,« 05 06 2017. [Mrežno]. Available: <https://issues.unrealengine.com/issue/UE-18622>. [Pokušaj pristupa 24 06 2021].
- [104] NVidia, »NVIDIA APEX Destruction,« 2021. [Mrežno]. Available: <https://www.nvidia.com/en-us/drivers/apex-destruction/>. [Pokušaj pristupa 24 06 2021].
- [105] NVidia, »Blast,« 2021. [Mrežno]. Available: <https://docs.nvidia.com/gameworks/content/gameworkslibrary/blast/blast.htm>. [Pokušaj pristupa 24 06 2021].

- [106] ByteSumPi LTD , »DENT - Networked Destruction - APEX,« 1 03 2017. [Mrežno]. Available: <https://www.unrealengine.com/marketplace/en-US/product/dent-destructible-environment?sessionInvalidated=true>. [Pokušaj pristupa 24 06 2021].
- [107] Epic Games, »Collision Filtering,« 17 04 2014. [Mrežno]. Available: <https://www.unrealengine.com/en-US/blog/collision-filtering>. [Pokušaj pristupa 24 06 2021].
- [108] J. Petanjek, »UE4 Networked Destruction (PhysX) demonstration with DENT,« 20 06 2021. [Mrežno]. Available: <https://www.youtube.com/watch?v=sEGvZf7ZNE0>. [Pokušaj pristupa 24 06 2021].
- [109] NVidia, »PhysX SDK,« 2021. [Mrežno]. Available: <https://developer.nvidia.com/physx-sdk>. [Pokušaj pristupa 24 06 2021].
- [110] NVidia, »Blast UE4 Plugin Quick Start,« 2017. [Mrežno]. Available: [https://gameworksdocs.nvidia.com/Blast/1.1/authoring\\_docs/BlastUe4\\_QuickStart.html](https://gameworksdocs.nvidia.com/Blast/1.1/authoring_docs/BlastUe4_QuickStart.html). [Pokušaj pristupa 24 06 2021].
- [111] A. Noon, »Order from Chaos - Destruction in UE4 | GDC 2019 | Unreal Engine,« 26 03 2019. [Mrežno]. Available: <https://www.youtube.com/watch?v=iFKzXnliH50&t=147s>. [Pokušaj pristupa 24 06 2021].
- [112] J. V. A. M. L. Matthias Worch, »Causing Chaos: The Future of Physics and Destruction in Unreal Engine | GDC 2019 | Unreal Engine,« 17 04 2019. [Mrežno]. Available: <https://www.youtube.com/watch?v=6T8Lzalq3Qs&t=779s>. [Pokušaj pristupa 24 06 2021].
- [113] Gaijin Entertainment, »THE SHOOTING RANGE #137: Ballistic Update / War Thunder,« 03 03 2019. [Mrežno]. Available: <https://youtu.be/gfGoEp0HbZc?t=577>. [Pokušaj pristupa 24 06 2021].
- [114] Mipmap Games, »Unreal Engine Tutorial - Bullet Physics / Projectile Physics and Penetration Part 1/5,« 26 11 2015. [Mrežno]. Available: <https://www.youtube.com/watch?v=jNVEimfgvm4>. [Pokušaj pristupa 30 06 2021].
- [115] Pizard, »The physics of projectile ballistics,« [Mrežno]. Available: [http://panoptesv.com/RPGs/Equipment/Weapons/Projectile\\_physics.php](http://panoptesv.com/RPGs/Equipment/Weapons/Projectile_physics.php). [Pokušaj pristupa 30 06 2021].
- [116] W. Odermatt, »Penetration and Perforation Calculator,« 2020. [Mrežno]. Available: <http://www.longrods.ch/perfcalc.php>. [Pokušaj pristupa 30 06 2021].
- [117] N. V. Matt Perez, »Exclusive: The Saga Of 'Star Citizen,' A Video Game That Raised \$300 Million—But May Never Be Ready To Play,« 01 05 2019. [Mrežno]. Available: <https://www.forbes.com/sites/mattpererez/2019/05/01/exclusive-the-saga-of-star-citizen->

a-video-game-that-raised-300-millionbut-may-never-be-ready-to-play/?sh=1321feaa5ac9. [Pokušaj pristupa 24 06 2021].

- [118] D. Swofford, »Legendary Designer Chris Roberts Making Re-entry into PC Gaming Stratosphere with Star Citizen from Cloud Imperium,« 11 09 2012. [Mrežno]. Available: <https://www.cnbc.com/id/100155462>. [Pokušaj pristupa 24 06 2021].
- [119] Improbable, »Console,« [Mrežno]. Available: <https://console.improbable.io/projects>. [Pokušaj pristupa 24 06 2021].

# Popis slika

Slika 1 InstallGDK.bat.....	3
Slika 2 Configuration Manager Visual Studio 2019 .....	3
Slika 3 Set as Startup Project.....	4
Slika 4 Build engine 1 sat .....	4
Slika 5 Build projekta.....	5
Slika 6 Build projekta.....	5
Slika 7 Pokretanje projekta .....	5
Slika 8 Povezivanje UE4 sa sustavom kontrole izvora.....	6
Slika 9 Konfiguracija sustava kontrole izvora.....	6
Slika 10 Pregled umreženih specifikacija trenutno popularnih igara [17].....	10
Slika 11 Mrežna opterećenost trenutno popularnih igara [20]. .....	11
Slika 12 Napredne tehnike umreženja u Battlefield 4 [22].....	12
Slika 13 Kompresija vektora i rotacije u UE4 .....	21
Slika 14 Hell Let Loose – objekt koji se može izgraditi nije repliciran [42] .....	23
Slika 15 Scavengers desinkronizacija pozicije igrača, vlastita izrada.....	24
Slika 16 Slojevitost [53] .....	28
Slika 17 Operacije [61] .....	29
Slika 18 Inspektor svijeta.....	30
Slika 19 Izbornik Spatial OS u UE4 .....	31
Slika 20 Konfiguracija umreženog izbacivanja po kvadratnoj udaljenosti .....	33
Slika 21 Uključivanje umreženog izbacivanja po udaljenosti sa modifikacijom frekvencije u postavkama projekta .....	34
Slika 22 Komponenta interesa za lika u UE4 .....	37
Slika 23 Konfiguirana komponente interesa za lika .....	38
Slika 24 Funkcija postavljanja rotacije lika u UE4 .....	41
Slika 25 Funkcija za ažuriranje interesa .....	41
Slika 26 Postavljanje ograničenja za vidljivo polje .....	42
Slika 27 Inspektor koji prikazuje interes pojedinog klijenta.....	42
Slika 28 Stvaranje lika pri replikaciji.....	42
Slika 29 Potpuno repliciran lik.....	43
Slika 30 Konfiguracija slojeva .....	46
Slika 31 Dodavanje konfiguracije svijetu.....	47
Slika 32 Zahtjev klijenta za stvaranje AI .....	48
Slika 33 Funkcija za stvaranje AI u stvaratelju .....	48

Slika 34 Stvaranje upravljača za AI .....	48
Slika 35 Jednostavno drvo odluka za AI .....	49
Slika 36 AI u igri .....	49
Slika 37 Squad dugo vrijeme za ponovno stvaranje, vlastita izrada .....	50
Slika 38 Klijent zahtijeva preuzimanje nad AI .....	51
Slika 39 Pronalaženje klijenta u stvaratelju.....	51
Slika 40 Pronalaženje AI u stvaratelju .....	51
Slika 41 Zamjena upravljača.....	52
Slika 42 Preuzeta kontrola nad AI.....	52
Slika 43 Replikacija kretnje u UE4 .....	54
Slika 44 Konfiguracija za pokretanje servera .....	55
Slika 45 OWI Enhanced Vehicle Movement u igri.....	55
Slika 46 Funkcija za slanje unosa.....	56
Slika 47 Rezultat unosa na klijentskoj strani, vlastita izrada .....	56
Slika 48 Rezultat unosa na serverskoj strani, vlastita izrada.....	57
Slika 49 Kašnjenje serverske pozicije, vlastita izrada .....	58
Slika 50 INetworkPredictionInterface u igri .....	60
Slika 51 Ručno izrađena fizika u igri.....	62
Slika 52 Aktiviranje proširenja za predikciju .....	63
Slika 53 Konfiguracija sustava za simulaciju fizike.....	63
Slika 54 Chaos fizika u igri [94].....	64
Slika 55 Deterministička fizika u igri, vlastita izrada .....	65
Slika 56 Spremanje objekata koji se nalaze na terenu.....	67
Slika 57 Notifikacija svih objekata na terenu o promjeni .....	67
Slika 58 Objekt postavljen kao inicijalno mrežno uspavan .....	68
Slika 59 Konstruiranje objekta .....	69
Slika 60 Kolizija koja pokreće funkciju .....	69
Slika 61 Buđenje mrežnog kanala .....	69
Slika 62 On Rep Mod Verts metode koja se izvodi na klijentskoj strani .....	70
Slika 63 Actor.cpp modificiran za potrebe deformacije terena.....	71
Slika 64 Struktura za spremanje događaja .....	72
Slika 65 Obrada strukture .....	72
Slika 66 Deformirani teren u igri .....	73
Slika 67 Događaji štete.....	74
Slika 68 Inverzna transformacija lokacije .....	74
Slika 69 Prolaženje kroz sve vrhove .....	75
Slika 70 Deformacija terena.....	75

Slika 71 Ažuriranje vrhova.....	75
Slika 72 Konstruktor klase Buildable.cpp .....	77
Slika 73 Postavljanje repliciranih varijabli kolizije.....	78
Slika 74 Funkcija koja se aktivira replikacijom kolizije.....	78
Slika 75 Funkcija za postavljanje objekata.....	79
Slika 76 Komponente zdravlja .....	79
Slika 77 Ažuriranje kolizijskih profila.....	79
Slika 78 Funkcije za primanje štete .....	80
Slika 79 Postavke projekta .....	80
Slika 80 Funkcija za pokretanje pregleda za izgradnju .....	80
Slika 81 Funkcija za zahtjev izgradnje .....	80
Slika 82 Pronalaženje korisničkog pogleda.....	81
Slika 83 Pregled izgradivog objekta.....	81
Slika 84 Funkcija za postavljanje više objekata u redu .....	81
Slika 85 Postavljanje više objekata u redu.....	82
Slika 86 Obrada slučaja probijanja .....	82
Slika 87 Funkcija za obradu probijanja pomoću linijskih tragova.....	83
Slika 88 Funkcija za postavljanje objekta na serverskoj strani .....	83
Slika 89 Implementacija funkcije za postavljanje objekta na serverskoj strani .....	83
Slika 90 Funkcija za stvaranje objekata u stvaratelju .....	84
Slika 91 Postavljeni objekti u igri .....	84
Slika 92 Modificirano oružje za izgradnju objekata .....	84
Slika 93 Izgrađeni objekt .....	84
Slika 94 Objekt pretvoren u skup krhotina .....	86
Slika 95 Hjerarhija i graf povezanosti [105] .....	87
Slika 96 Različite pozicije krhotina na klijentima .....	88
Slika 97 Pravilno konfiguiriran uništivi objekt .....	89
Slika 98 Konfiguracija uništivog objekta.....	89
Slika 99 Konfiguracija kolizije .....	90
Slika 100 Sinkroniziran uništivi objekt [108] .....	91
Slika 101 Temeljni koncepti "Chaos" [112] .....	93
Slika 102 Projektil kao fizički objekt u igri.....	94
Slika 103 Linijski trag u igri .....	94
Slika 104 Kolizijska komponenta metka.....	96
Slika 105 Događaj preklapanja .....	97
Slika 106 Točke potrebne za izračune, vlastita izrada .....	97
Slika 107 Izračun maksimalnog probijanja.....	98

Slika 108 Izračun točke maksimalnog probijanja .....	98
Slika 109 Pronalaženje točke na poledini objekta .....	99
Slika 110 Slučaj ne probijanja objekta .....	99
Slika 111 Stvaranje efekata na klijentskoj strani .....	100
Slika 112 Slučaj probijanja objekta .....	100
Slika 113 Nastavak projektila.....	100
Slika 114 Efekti probijanja na korisničkoj strani .....	101
Slika 115 Koncept za izbornik odabira odreda,vlastita izrada (RedeployMenue) .....	103
Slika 116 Koncept za izbornik naredbi, vlastita izrada (FullView – Commander) .....	104
Slika 117 Koncept za izbornik sustava eskalacije, vlastita izrada .....	106
Slika 118 Prikaz zapovjednika, vlastita izrada - CommandingView-Commander .....	107
Slika 119 Postavke svijeta.....	108
Slika 120 Provjeravanje kontrole .....	109
Slika 121 Postavljanje stanja točaka.....	109
Slika 122 Logika zauzimanja kontrolne točke .....	110
Slika 123 Komponenta tima na liku.....	110
Slika 124 Inicijalizacija meča .....	111
Slika 125 Ažuriranje bodova .....	111
Slika 126 Završetak igre .....	111
Slika 127 Konfiguracija servera .....	112
Slika 128 Pokretanje u konzoli.....	112
Slika 129 Lokalno pokretanje igre.....	113

## **Popis tablica**

Tablica 1 Istoimeni pojmovi UE i Spatial OS [69].....	32
Tablica 2 Opis ograničenja [57] .....	36
Tablica 3 Ažuriranja po polju .....	43
Tablica 4 Prosječni broj klijenata po polju .....	44
Tablica 5 Broj interakcija po polju .....	44
Tablica 6 Opis umreženih uloga [84] .....	59
Tablica 7 Opis koraka na zasebnim likovima [84] .....	59