SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1962

ANALIZA I DEMONSTRACIJA ALATA ZA VARANJE U UMREŽENIM VIDEOIGRAMA

Marin Prusac

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1962

ANALIZA I DEMONSTRACIJA ALATA ZA VARANJE U UMREŽENIM VIDEOIGRAMA

Marin Prusac

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 1962

Pristupnik:	**Marin Prusac (0036547678)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Mirko Sužnjević

Zadatak: **Analiza i demonstracija alata za varanje u umreženim videoigrama**

Opis zadatka:

U umreženim videoigrama, varanje može ozbiljno utjecati na igračko iskustvo i ravnotežu videoigre. Različiti alati za varanje omogućuju korisnicima manipulaciju podacima videoigre, što može rezultirati nepoštenim prednostima pojedinog igrača u umreženoj videoigri. U ovom radu potrebno je analizirati postojeće alate za varanje u umreženim videoigrama, uključujući njihove metode i tehnike, te opisati teoretsku podlogu na kojoj se temelje alati za varanje u igri. Zatim, odabrati jedan alat i razviti jednostavnu umreženu videoigru u kojoj će se demonstrirati rad odabranog alata. Igra treba biti dizajnirana na način da prikazuje kako varanje putem alata može utjecati na igranje, poput manipulacije rezultatima, premještanja objekata ili promjene stanja u igri. U radu je potrebno detaljno opisati proceduru korištenja alata za varanje, korake potrebne za intervenciju u videoigru te načine na koje se alat koristi za promjenu ponašanja igre.

Rok za predaju rada: 23. lipnja 2025.

Hvala mentoru, Mirku Sužnjeviću, na strpljenju i zanimljivoj temi.

Hvala FER-u na svom znanju pruženom u zadnje tri godine.

Sadržaj

1. Uvod

Umrežene videoigre predstavljaju jednu od najbrže rastućih grana zabavne industrije, s milijunima igrača koji se svakodnevno povezuju. Posebno su popularne umrežene igre natjecateljskog tipa, u kojima se igrači natječu za dobitak igre, a neki u tome nalaze životnu karijeru sudjelovanjem u sve popularnijim E-Sports natjecanjima.

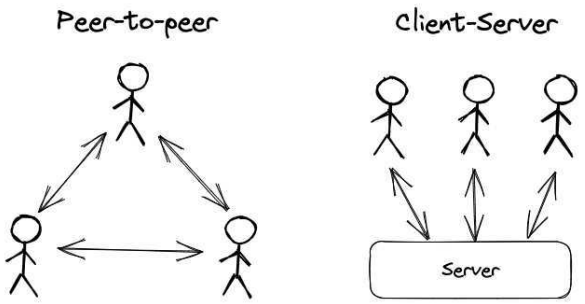
Kako bi se razvilo okruženje u kojemu je igračima zabavno igrati, važno je da su uvjeti igre pravedni prema svima koji sudjeluju. Nažalost, paralelno s rastom popularnosti mrežnih igara, raste i problem varanja koji može značajno narušiti igračko iskustvo i kompetitivnu ravnotežu. Varanje nije samo tehnički problem – ono ima dalekosežne posljedice na ekonomiju igara, reputaciju razvojnih tvrtki i zadovoljstvo igrača.

Za rješavanje problema varanja potrebno ga je razumjeti i promatrati iz perspektive osobe koja vara te s time na umu oblikovati i zaštiti videoigru. U slučaju neopreznog programiranja, igra je podložna jednostavnim mehanizmima varanja. Ovaj rad demonstrira propuste u naivnom razvijanju primjerne videoigre i kako se oni iskorištavaju u svrhu varanja, s očekivanim doprinosom podizanja svjesnosti o problemima varanja i njihovom rješavanju.

Struktura rada prati logičan slijed od teorijskih osnova do praktične implementacije. U poglavljima 2. i 3. razmatraju se trenutni industrijski standardi umrežavanja te aktualni problemi varanja i njihovog sprječavanja. U poglavljima 4. i 5. specificira se i prikazuje izvedba ranjive videoigre. Slijedi poglavlje 6. u kojem se demonstrira varanje na primjeru opisane videoigre i teorija iza alata korištenih za varanje. Koraci za sprječavanje varanja razmotreni su u poglavlju 7., dok 8. povlači zaključak.

2. Mrežne arhitekture videoigara

Umreženost u videoigrama složena je komponenta koja omogućava međusobno povezivanje igrača u realnom vremenu, stvarajući zajedničko iskustvo igranja. Moderne videoigre koriste raznovrsne arhitekture i protokole kako bi ostvarile efikasnu komunikaciju između klijenata, omogućili sinkronizirano stanje igre i minimizirali latenciju. Glavne arhitekture uključuju klijent-poslužitelj (engl. client-server) i peer-to-peer modele [1]. Usporedba komunikacije između klijent-poslužitelj i peer-to-peer modela prikazana je na slici 2.1.



Slika 2.1. Komunikacija u mrežnim arhitekturama [1]

2.1. Klijent-poslužitelj arhitektura

Klijent-poslužitelj arhitektura je standard u modernim umreženim igrama [2], gdje središnji poslužitelj upravlja stanjem igre, dok klijenti od njega primaju te mu šalju podatke. U ovom modelu, poslužitelj ima većinsku kontrolu nad stanjem igre što, osim otežavanja varanja, rasterećuje klijente od teže logike igre te je time pogodan za skaliranje [1].

Glavni nedostatak leži u većoj latenciji koja se ispoljava na klijentima. Izvršavanje neke akcije zahtjeva da se ona najprije proslijedi poslužitelju, a on ju zatim evaluira i primijeni na stanje igre te klijentima šalje osvježene podatke. Zbog toga igrači primjećuju

kašnjenje vizualne promjene za njihovim unosima. Drugi nedostatak je cijena održavanja poslužitelja koju manje tvrtke ne mogu priuštiti [1].

2.2. Peer-to-Peer arhitektura

Alternativa klijent-poslužitelj arhitekturi je peer-to-peer arhitektura. U ovoj arhitekturi klijenti su zaduženi za svu logiku i međusobno izravno komuniciraju. Izvedbe ove arhitekture se razlikuju. Konkretno u slučaju pogonskog sustava Unity [3], peer-to-peer arhitektura u kojoj je jedan od igrača domaćin, a drugi gosti, zapravo spada pod klijent-poslužitelj topologiju, dok je potpuna ravnopravnost klijenata ostvarena u topologiji distribuiranog autoriteta [4].

Davanje klijentima kontrole nad dijelovima logike i stanja s kojim najviše interaktiraju smanjuje latenciju. Mana ove arhitekture je opterećivanje klijenata s većim odgovornostima i rizik od varanja [1].

3. Varanje u videoigrama

Razlog zašto bi neki igrač varao nije samo taj da bi se više zabavio od drugih; mnogi igrači varaju u svrhu vlastite financijske dobiti. Oni koriste alate za varanje kako bi osvojili natjecanja ili napredovali svoj igrački profil kojeg će kasnije prodati [5], [6]. Obično igrači koji koriste alate za varanje nisu oni koji su razvili te alate, već ih kupuju od programera po visokoj cijeni [7]. Bez obzira na motivaciju varanja, ono gotovo uvijek rezultira lošijim iskustvom drugih igrača. Nedostatak mehanizama za sprječavanje i sankcioniranje varanja loše utječe na reputaciju razvojne tvrtke zbog čega mnoge od njih ulažu mnogo u takve mehanizme [8].

3.1. Igre ranjive na varanje

Bilo da je namjera varati ili spriječiti varanje, valja najprije dobro poznavati moguće ranjivosti razvijane videoigre. Velika većina ranjivosti proizlazi iz činjenice da je igračima dana izravna kontrola nad stanjem igre [9]. Peer-to-peer igre inherentno imaju ovakav problem jer ne postoji neutralni središnji autoritet. To znači da se sva logika nalazi na klijentskim aplikacijama, a kako su one instalirane na igračevom računalu, podložne su bilo kakvim zlonamjernim manipulacijama. Ipak, priroda je ovakvih igara da igrači nemaju razloga varati u njima s obzirom na to da su prvenstveno namijenjene da se igraju među poznanicima i to često u kooperativnom modu.

Međutim, čak i u natjecateljskim igrama koje koriste klijent-poslužitelj arhitekturu, programeri dodjeljuju klijentima dio kontrole koja nije provjerena na poslužitelju. To se čini u svrhu boljih performansi i jednostavnosti. Primjerice u popularnoj videoigri *Dead by Daylight*, svaki igrač upravlja kretanjem svojeg lika; na klijentu igrača ubojice provodi se provjera pogotka drugih igrača pri zamahu oružja [10]. To izaziva frustracije kada se ova kontrola zloupotrijebi, ali i kada igrač ubojica ima slabiju internetsku vezu, što mu

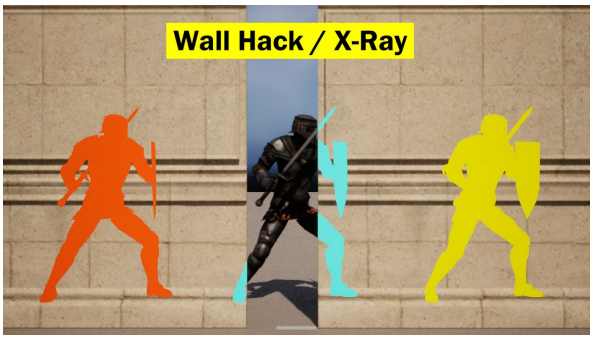
zapravo ide u korist [10].

Najgora je vrsta ranjivosti ona koja igraču omogućuje napad na samu infrastrukturu koja poslužuje igru, poput udaljenog izvršavanja koda. Potrebno je osnovno poznavanje sigurnosti računalnih sustava za sprječavanje ovakve nesigurnosti. Ona omogućuje napadaču da prouzroči znatno veću štetu od samog varanja u videoigri i zahtijeva poznavanje drugačije discipline iskorištavanja ranjivosti.

3.2. Varanje u igrama s autoritativnim poslužiteljem

Videoigre u kojima poslužitelj ima konačnu riječ o stanju i logici igre nazivaju se igre s autoritativnim poslužiteljem [11]. U takvim videoigrama klijenti (kojima izravno upravljaju igrači) šalju naredbe poslužitelju, a on utvrđuje pravilnost tih naredbi i njihov ishod te nazad šalje osvježeno stanje igre koje se na igračevim računalima vizualno prikazuje. Zlonamjerna uporaba klijentske aplikacije u igrama ovakvih arhitektura uglavnom ne može igraču donijeti nikakvu prednost jer se logika nalazi na poslužitelju, a on se ne može mijenjati [12].

U jednostavnim videoigrama, relativno je jednostavno osigurati potpuni autoritet poslužitelja, ali u slučajevima 3D igara, nije lako odrediti trebaju li stanja pojedinih objekata biti poslana određenom igraču. Primjerice, svaki bi igrač trebao vidjeti protivničkog igrača ako mu se nalazi u vidnom polju, ali teško je, ili računski zahtjevno, savršeno provjeriti tu činjenicu. Zbog toga se takva funkcionalnost obično tek djelomično ostvaruje, a u gorem slučaju sve su informacije poslane klijentima. Tada je moguće koristiti tzv. *wallhack* tip alata za varanje [13] koji je prikazan na slici 3.1. Oni analiziraju primljeno stanje igre te na temelju njega na zaslon crtaju obrise protivničkih igrača preko svih drugih vizualnih elemenata. To omogućuje igraču koji vara da vidi svoje protivnike kroz zidove, odakle i dolazi naziv ovog alata.



Slika 3.1. Wallhack [14]

Čak i u igrama u kojima je poslužitelj potpuno autoritativan i gdje nema slanja nedozvoljenih informacija klijentima, moguće je koristiti razne druge alate. Oni funkcioniraju tako da čitaju stanje igre dostupno klijentu, bilo iz radne memorije ili sa zaslona, a zatim na temelju izračuna unose naredbe umjesto igrača, ponekad i do mjere da ga mogu u potpunosti zamijeniti ako se koristi napredniji oblik umjetne inteligencije [15]. U ovakvim alatima iskorištavaju se prednosti računalnih sustava kao što su manje vrijeme reakcije, preciznije upravljanje i, u krajnjim slučajevima, pametnije odlučivanje.

3.3. Sprječavanje varanja

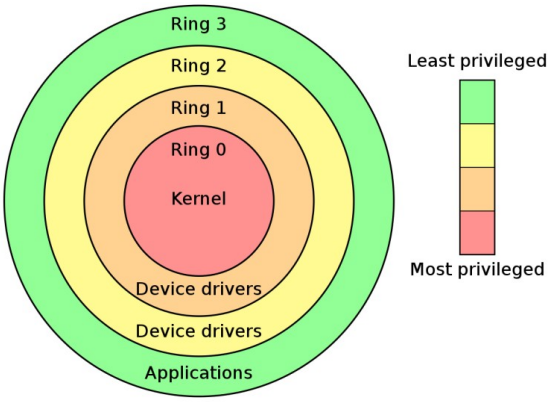
3.3.1. Programi protiv varanja

Budući da dobro dizajnirana arhitektura videoigre nije dovoljna za potpunu prevenciju varanja, danas se koriste specijalizirani programi protiv varanja (engl. anti-cheat) [16], [15]. Oni se instaliraju, uz videoigru koju štite, na igračevu računalu. Obično nije moguće zaobići instaliranje takvog programa, a da se igra i dalje može uspješno koristiti [17]. Ti programi analiziraju ostale procese na računalu i provjeravaju mijenja li neki od njih radnu memoriju ili kod videoigre na nedopušten način [16]. Iako nisu savršeni, s vremenom postaju sve učinkovitiji kako se otkrivaju novi načini za njihovo zaobilaženje. Time je varanje znatno otežano.

3.3.2. Varanje na razini jezgre

Glavni način zaobilaženja programa protiv varanja je varanje na razini jezgre. Jezgra je obično rezervirana samo za programe od velike važnosti za ispravno funkcioniranje računala, ali moguće je pokretati i vlastite programe na toj razini [18]. Na ovaj način

privilegirani alati za varanje uspješno zaobilaze većinu programa protiv varanja ako i oni sami nisu pokrenuti u privilegiranom modu. Na slici 3.2. vidljivi su prsteni zaštite koji štite različite razine pristupa resursima.



Slika 3.2. Razine privilegije programa [19]

Danas je ova tema pogotovo kontroverzna jer sve više kompanija inzistira na instaliranju programa protiv varanja na razini jezgre (engl. kernel-level anti-cheat) [20]. Budući da ti programi imaju najveće moguće privilegije, aktivno analiziraju svaki dio računala, djeluju i kada igra nije pokrenuta te kontinuirano komuniciraju s poslužiteljem tvrtke koja ga je razvila, mnogi igrači to vide kao kršenje privatnosti [20]. Ovakvi programi također inzistiraju na specifičnim postavkama računala i operacijskim sustavima; igračima koji ne mogu ili ne žele uskladiti svoje računalo sa zahtjevima videoigre onemogućeno je njeno igranje [21].

3.4. Povezanost s drugim disciplinama

Varanje u videoigrama čini se kao vrlo usko područje primjene tehničkih znanja, međutim ova disciplina povezana je i s općenitim hakiranjem programa, a i dobronamjernim modiranjem igara.

3.4.1. Modiranje igara

"Modovi" su dodaci videoigri koje razvija zajednice igrača kako bi joj dodali novu ili izmijenjenu funkcionalnost [22]. Svaka tvrtka ima svoju politiku prema dozvoli modiranja

svojih videoigara. U videoigrama koje nisu natjecateljskog tipa, modiranje je obično dozvoljeno, a ponekad i službeno podržano [23].

Varanje videoigre može se smatrati i njenim modiranjem. Povlačenje granice između varanja i modiranja nije uvijek tako jednostavno. Primjerice, u videoigri *Minecraft* modiranje je podržano i često prakticirano, zbog čega Minecraft broji velik broj modova [24], ali pojedini Minecraft serveri na kojima se mogu igrati natjecateljske mini igre ne dozvoljavaju uporabu modova, odnosno tada se oni smatraju varanjem [25].

Načini na koji se videoigre modiraju, ipak, imaju neke različitosti. Mijenjane ključne datoteke koje videoigra koristi mogu se objaviti i omogućiti drugima da zamijene svoje datoteke s modiranim [24]. To nije idealno rješenje jer bi instaliranje drugog moda pregazilo promjene prvog instaliranog moda. Zbog toga je ključno ostaviti originalne datoteke nepromijenjenima, a umjesto toga treba dodati nove datoteke koje sadrže izmijenjene funkcionalnosti. Te nove datoteke unose se u proces videoigre utovarivačem modova (engl. mod loader) [26]. Na taj način omogućeno je egzistiranje više modova odjednom. Važno je napomenuti da to ne garantira uspješno djelovanje svih modova jer oni mogu imati konflikte pri redefiniranju istih aspekata igre.

3.4.2. Hakiranje drugih tipova softvera

Hakiranje bilo kojeg programa zahtijeva određenu razinu reverznog inženjeringa. Da bi se promijenila funkcionalnost programa, potrebno je dobro poznavati njegov kod, a za njegovo proučavanje koriste se debuggeri, dekompilatori i čitači radne memorije.

Za varanje u Unity igrama najčešće se koriste .NET dekompilatori koji također mogu poslužiti za manipulaciju funkcionalnosti drugih softvera razvijenih na .NET platformi. Njihovom dekompilacijom može se dobiti oblik koda blizak izvornom [27]. Takav kod lakše se analizira, a ponekad ga je moguće izmijeniti i takvog kompilirati. U slučaju drugih programa, češće nije moguće dobiti izvorni kod, što ograničava mogućnost izmjena [28]. Tada je potrebna analiza na razini strojnog koda i izravno mijenjanje radne memorije.

4. Razvijena ranjiva videoigra

4.1. Inspiracija i opis

U sklopu rada razvijena je umrežena videoigra *Pa-Pa!*. Inspiracija za igru bila je istoimena igra iz djetinjstva. Suprotstavljeni igrači počinju skriveni na različitim dijelovima područja za igru. Kada igra službeno počne, šuljaju se po području i traže protivničkog igrača. U trenutku pronalaska, igrač vikne "Pa pa Marin!" (u slučaju da se protivnik zove Marin) i time je pronađeni igrač eliminiran iz igre. Ako su u igri samo dva igrača, tada igrač koji je našao protivničkog igrača i uzviknuo pobjeđuje rundu.

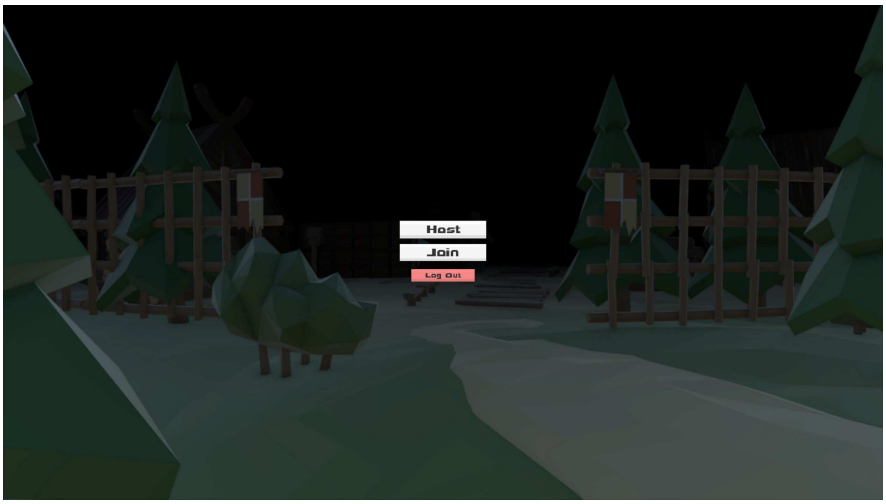
Videoigra se igra na sličan način uz neke izmjene. Igraju dva igrača, od kojih je jedan domaćin, a drugi gost. Gost se pridružuje domaćinu odabirom sobe na listi dostupnih soba. Domaćin pokreće igru u trenutku kada su oba igrača spremna. Na početku runde, igrači su postavljeni na dvije nasumično odabrane pozicije. Igrač koji je prvi našao protivničkog igrača pobjeđuje rundu i osvaja 5 bodova. Zatim su igrači stvoreni na drugim pozicijama te počinje sljedeća runda. Na igraćem se području također pojavljuju novčići, svaki od kojih nagrađuje 1 bod igraču koji ga je skupio. Igra traje sve dok jedan igrač nije osvojio 30 bodova.

4.2. Povezivanje igrača

Pri pokretanju igre, pojavljuje se početno sučelje s pozadinskom muzikom prikazano na 4.1. Igrač ima mogućnost anonimne prijave.

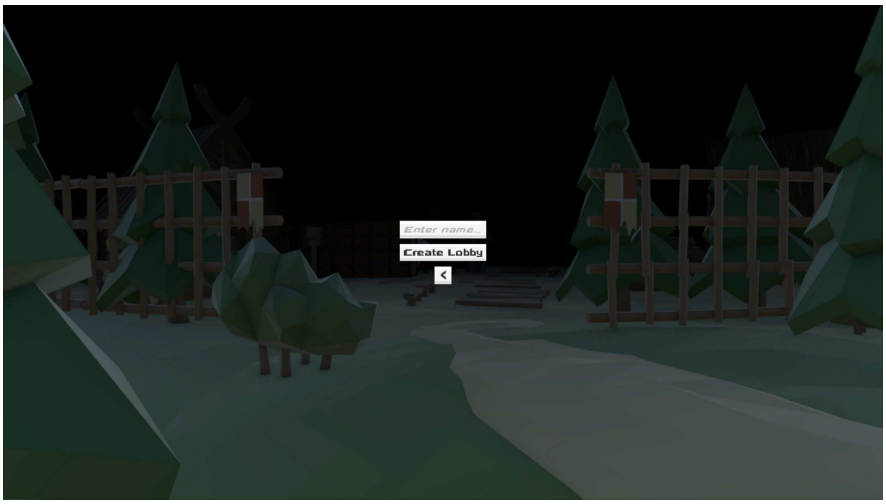
Slika 4.1. Početno sučelje igre

Nakon uspješne prijave, igrač u glavnom izborniku (4.2.) ima opciju napraviti novu sobu (engl. lobby) u kojem će slučaju taj igrač biti domaćin (engl. host), ili se priključiti već napravljenoj sobi i biti gost (engl. guest).



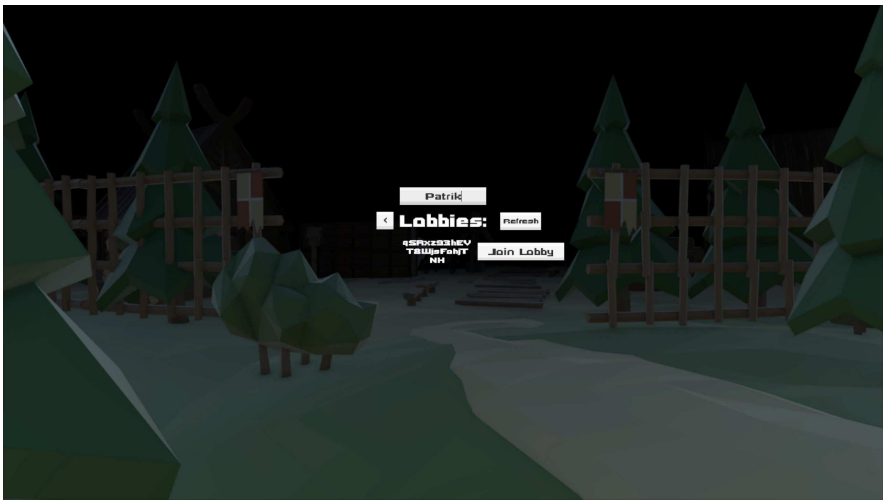
Slika 4.2. Glavni izbornik

U slučaju da je izabrao napraviti novu sobu, pristupa sučelju izrade sobe (4.3.). Najprije mora odabrati svoje ime. Zatim pritišće gumb za kreiranje, što ga vodi do sučelja same sobe, vidljivo na 4.5.



Slika 4.3. Sučelje pravljenja nove sobe

U slučaju da je odlučio pridružiti se domaćinu, pristupa sučelju za pridruživanje postojećoj sobi (4.4.). Također mora odabrati ime, a onda odabrati kojoj se sobi želi pridružiti. Odabirom sobe, prikazuje se njeno sučelje.



Slika 4.4. Sučelje pridruživanja sobi

Imena domaćina i gosta prikazani su u korisničkom sučelju sobe. Ako su oba igrača prisutna, domaćin ima mogućnost pokrenuti igru.

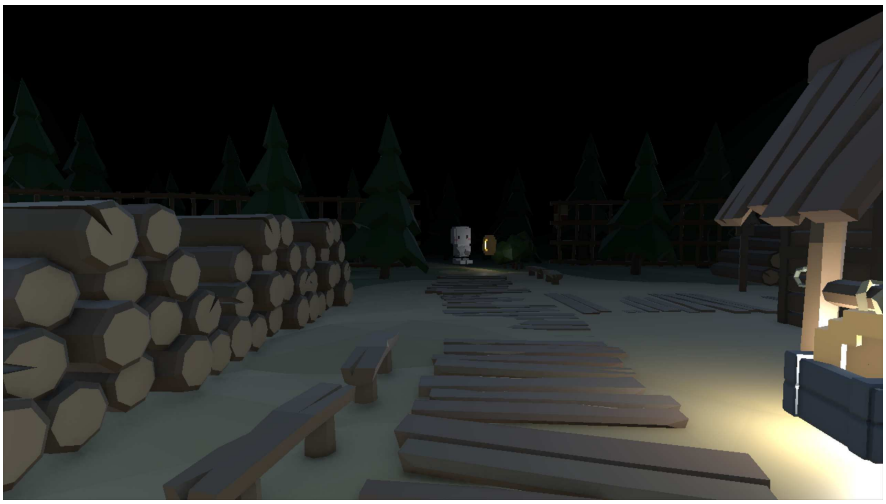
Slika 4.5. Sučelje sobe

4.3. Igranje igre

Oba igrača najprije se smještaju na dvije nasumične lokacije kao što je prikazano na 4.6. Nakon par sekundi, mogu se početi kretati s tipkama W, A, S i D na tipkovnici, gledati uokolo pomakom miša te skakati s razmaknicom.

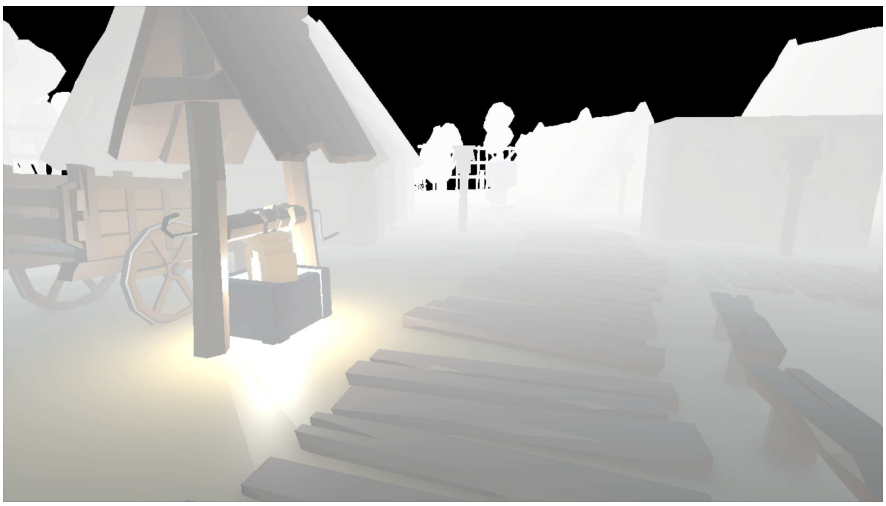
Slika 4.6. Nasumično pozicioniranje

Igrači se šeću područjem i po mogućnosti skupljaju novčiće, svaki vrijedan 1 bod. Ovo traje sve dok jedan od igrača ne nađe drugog. Na slici 4.7. prikazan je trenutak uočavanja lika protivničkog igrača.



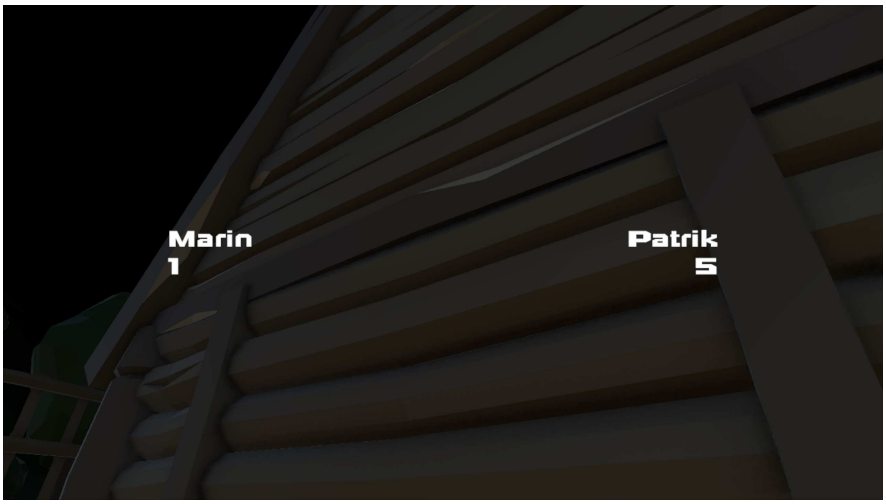
Slika 4.7. Pronalazak protivnika

U trenutku pronalaska protivničkog igrača, potrebno je pritisnuti lijevi gumb miša te je tada runda pobijeđena i osvojeno je 5 bodova. Igraču koji je izgubio rundu prikazuje se efekt gubitka, kao što je vidljivo na 4.8. Ako igrač pritisne lijevi gumb miša, a da nije vidio protivničkog igrača, neće moći iskoristiti ga narednih 5 sekundi, a runda se nastavlja.



Slika 4.8. Vizualni efekt gubitka runde

Na kraju svake runde prikazuju se dosad osvojeni bodovi (4.9.) i započinje nova runda. Igra se nastavlja sve dok jedan od igrača ne osvoji 30 bodova. Prikazi pobjede i gubitka prikazani su na slikama 4.10., odnosno 4.11.



Slika 4.9. Prikaz bodova na kraju svake runde

Slika 4.10. Sučelje u slučaju pobjede igre

Slika 4.11. Sučelje u slučaju gubitka igre

5. Programska izvedba videoigre

5.1. Razvojni alati

Igra opisana u poglavlju 4. razvijena je sljedećim razvojnim alatima.

5.1.1. C#

C# je objektno orijentirani programski jezik koji je razvio Microsoft i koji se izvršava na .NET platformi. Jezik je jednostavan za učenje, podržava više programskih paradigmi (uključujući objektno, funkcijsko i generičko programiranje) te ima snažnu tipizaciju i bogatu standardnu biblioteku. C# se široko koristi za razvoj desktop, web i mobilnih aplikacija, ali i za razvoj videoigara, posebno u kombinaciji s Unity Engineom [29].

5.1.2. Unity Engine

Unity je pogonski sustav koji omogućuje razvoj 2D i 3D igara te interaktivnih simulacija za različite platforme, uključujući računala, mobilne uređaje, konzole i uređaje proširene stvarnosti. Unity koristi C# kao glavni skriptni jezik (uz mogućnost korištenja JavaScripta), a razvoj se odvija kroz vizualno sučelje Unity Editora koje omogućuje brzo prototipiranje, uređivanje scena, upravljanje resursima i testiranje igre u stvarnom vremenu. Sadrži napredne alate za fiziku, animaciju, grafiku, zvuk i mrežno programiranje, te ima veliku zajednicu i bogat ekosustav dodataka i resursa [3].

5.1.3. Netcode for GameObjects

Netcode je visokorazinska biblioteka za implementaciju mrežne komunikacije u sklopu Unityja. Ostvaruje ponašanje poslužitelja i klijenta u istom kodu i sinkronizira objekte među svim instancama videoigre prateći definirane postavke [30]. Svi objekti, odnosno

razredi njima dodijeljeni, mogu imati različito ponašanje ovisno o tome na kojoj instanci se izvršavaju, omogućujući željenu podjelu vlasništva nad objektima. Specifično Netcode for Gameobjects (skr. NGO) barata s često korištenim GameObject tipovima objekata u Unityju [31], dok Netcode for Entities upravlja objektima u sklopu Entity Component System (skr. ECS) arhitekture [32], [33].

5.1.4. Unity Gaming Services

Za ostvarenje povezivanja među igračima, koriste se Lobby service i Relay service dostupni u sklopu Unity Gaming Services (skr. UGS) [34]. Lobby service pruža lakšu implementaciju lobby sustava, odnosno povezivanja igrača u istu sesiju igre. Relay service prosljeđuje pakete među igračima, stvarajući iluziju peer-to-peer povezanosti tijekom trajanja runde igre. Svi servisi unutar UGS-a međusobno su kompatibilni i imaju odličnu integraciju s NGO-om.

5.2. Ključni dijelovi izvedbe

5.2.1. Ostvarenje umreženosti

Za umrežavanje igrača koriste se NGO (5.1.3.) i UGS (5.1.4.). Klijenti igrača najprije se moraju spojiti koristeći AuthenticationService (5.1.) u sklopu UGS-a pritiskom na "Sign In" gumb prikazan na slici 4.1.

AuthenticationManager.cs

```
32 if(UnityServices.State == ServicesInitializationState.Uninitialized)
33     await InitializeUnityServices();
34 AuthenticationService.Instance.SignedIn += OnSignedIn;
35 await AuthenticationService.Instance.SignInAnonymouslyAsync();
36 Instance = new AuthenticationManager();
```

Listing 5.1. Autentifikacija

Za upravljanje sobama koristi se LobbyService API. U slučaju odabira stvaranja nove sobe, korisnik poziva "CreateLobbyAsync" funkciju kao što je prikazano na 5.2.

LobbyManager.cs

```
131 var player = BuildPlayer(playerName);
132 var lobbyOptions = new CreateLobbyOptions
133 {
134     IsPrivate = !isPublic,
135     Player = player,
136 };
137  var lobby = await LobbyService.Instance.CreateLobbyAsync(Guid.NewGuid
       ().ToString(), 2, lobbyOptions);
138
139 Instance = new LobbyManager(lobby, true);
140 JoinedLobby?.Invoke();
```

Listing 5.2. Stvaranje nove sobe

Za pristupanje sobi, potrebno je znati "lobbyCode". Korisnik dolazi do dotičnog koda tako što najprije dohvati listu soba (5.3.). Na grafičkom sučelju prikazuju se sve dostupne sobe, a korisnik se može pridružiti bilo kojoj slobodnoj sobi (5.4.).

LobbyManager.cs

```
223 var options = new QueryLobbiesOptions
224 {
225     Count = 5,
226     Filters = new List<QueryFilter>
227     {
228         new(QueryFilter.FieldOptions.IsLocked, "false", QueryFilter.
       OpOptions.EQ)
229     }
230 };
231 var response = await LobbyService.Instance.QueryLobbiesAsync(options);
232 var lobbies = response.Results;
```

Listing 5.3. Dohvaćanje dostupnih soba

LobbyManager.cs

```
152 var player = BuildPlayer(playerName);
153 var joinOptions = new JoinLobbyByCodeOptions {
154     Player = player
155 };
```

```
156 var lobby = await LobbyService.Instance.JoinLobbyByCodeAsync(lobbyCode
    , joinOptions);
157 Instance = new LobbyManager(lobby, false);
158 JoinedLobby?.Invoke();
```

Listing 5.4. Pridruživanje sobi

Zadnji korak u povezivanju je pritisak na "Start Game" gumb. On poziva API RelayServicea. Tako uspostavlja klijent-poslužitelj vezu između klijenata, spremnu za razmjenu podataka o stanju igre. Relay se koristi na sličan način kao i Lobby. Dio implementacije pridruživanja ili pokretanja releja vidljiv je na 5.5. i 5.6. isječcima koda.

`RelayManager.cs`

```
56 var allocation = await RelayService.Instance.CreateAllocationAsync(19)
    ;
57 NetworkManager.Singleton.GetComponent<UnityTransport>().
    SetRelayServerData(allocation.ToRelayServerData(connectionType: "
    udp"));
58
59 if (!NetworkManager.Singleton.StartHost())
60 {
61     throw new Exception("Could not start host");
62 }
63
64 var relayCode = await RelayService.Instance.GetJoinCodeAsync(
    allocation.AllocationId);
65 await LobbyService.Instance.UpdatePlayerAsync(LobbyManager.Instance.
    LobbyId, AuthenticationService.Instance.PlayerId, new
    UpdatePlayerOptions
66 {
67     AllocationId = allocation.AllocationId.ToString()
68 });
69 await LobbyManager.Instance.ChangeLobbyData("relayCode", relayCode);
```

Listing 5.5. Pokretanje releja

RelayManager.cs

```
var playerId = AuthenticationService.Instance.PlayerId;
var allocation = await RelayService.Instance.JoinAllocationAsync(
    relayCode);
NetworkManager.Singleton.GetComponent<UnityTransport>().
    SetRelayServerData(allocation.ToRelayServerData(connectionType: "
    udp"));

if (!NetworkManager.Singleton.StartClient())
    throw new Exception("Could not start a client.");

var updatedPlayerOptions = new UpdatePlayerOptions
{
    AllocationId = allocation.AllocationId.ToString()
};

await LobbyService.Instance.UpdatePlayerAsync(LobbyManager.Instance.
    LobbyId, playerId, updatedPlayerOptions);
```

Listing 5.6. Pridruživanje releju

5.2.2. Upravljanje likom

Kretanje likova ostvareno je s pomoću "CharacterController" razreda pružanog od Unityja. Kretanje ovisi o parametrima poput brzine i snage skoka. Logika kretanja izvršava se na klijentu (5.7.).

PlayerControls.cs

```
private void Movement()
{
    _verticalVelocity -= gravity * Time.deltaTime;
    if (Grounded)
    {
        if (_jumping)
        {
            _verticalVelocity = jumpPower;
            _jumped = true;
        }
        else
```

```
111         {
112             if(_jumped)
113                 PlayAnimationRpc("Landed");
114             _jumped = false;
115             _verticalVelocity = 0;
116         }
117     }
118     _characterController.Move( _verticalVelocity * Time.deltaTime *
    Vector3.up);
119     var movementAxes = _inputSystem.Player.Movement.ReadValue<Vector2
    >();
120     var movementAxes3D = new Vector3(movementAxes.x, 0, movementAxes.y
    );
121     var yawRotation = Quaternion.Euler(0, _lookingDirection.y, 0);
122     var movementDir = yawRotation * movementAxes3D;
123     if (movementDir.magnitude > 0.01f)
124     {
125         _movementDirection = movementDir;
126     }
127     PlayAnimationRpc("Speed", movementAxes.magnitude);
128     PlayAnimationRpc("Vertical", _verticalVelocity);
129     var movement = Time.deltaTime * speed * movementDir;
130     _characterController.Move(movement);
131 }
```

Listing 5.7. Kretanje lika

U trenutku kada igrač vidi protivnika, može pritisnuti lijevu tipku miša i time pozvati funkciju za provjeru vidljivosti čiji je dio prikazano na 5.8. isječku koda. U slučaju pozitivnog rezultata klijent šalje poslužitelju poziv funkcije za pobjedu runde.

ArenaPlayer.cs

```
56 var result = _visionInstance.Check();
57 if (result)
58 {
59     GameManager.Instance.PahPahRpc(NetworkManager.LocalClientId);
60     UIManager.Instance.FlashAnimation(new Color(1,1,1,0.5f), 0.75f);
61 }
```

Listing 5.8. Detekcija protivnika

5.2.3. Bodovanje

Bodovi se dodjeljuju s pomoću "AssignPointsRpc" funkcije koja se poziva na poslužitelju. Funkcija se poziva u slučaju skupljanja novčića (5.9.) ili u slučaju detektiranja protivnika (5.10.).

CollectibleCoin.cs

```
28 private void OnTriggerEnter(Collider other)
29 {
30     if (!IsServer) return;
31     if (!_pickupable) return;
32     if (other.TryGetComponent<ArenaPlayer>(out var player))
33     {
34         CoinCollectedRpc(player.OwnerClientId);
35     }
36 }
37
38 [Rpc(SendTo.Server)]
39 private void CoinCollectedRpc(ulong clientId, RpcParams rpcParams =
       default)
40 {
41     _pickupable = false;
42     StartCoroutine(CollectTween());
43     GameManager.Instance.AssignPointsRpc(1, clientId);
44 }
```

Listing 5.9. Logika novčića

GameManager.cs

```
166 [Rpc(SendTo.Server)]
167 public void PahPahRpc(ulong clientId)
168 {
169     _hostPlayer.StopMovingRpc();
170     _guestPlayer.StopMovingRpc();
171     AssignPointsRpc(5, clientId);
172     RoundEndRpc(clientId, _hostPoints, _guestPoints);
173 }
```

Listing 5.10. Funkcija osvajanja runde

5.2.4. Vizualni i auditorni elementi

Vizualne i auditorne komponente ostvarene su s pomoću slobodnih resursa s interneta.

3D modeli preuzeti su s Unity Asset Store platforme [35]. U privitku A2. priložene su poveznice do korištenih resursa.

Za zvučne isječke korištena je incompetech mrežna stranica [36]. U privitku A1. priložena je lista akreditacija svakog isječka.

6. Metodologija i primjena varanja

6.1. Princip rada alata za varanje

Zbog velike raznolikosti alata za varanje, teško je objasniti jedan princip rada koji pokriva sve alate. Zbog toga, ovdje će biti opisani najčešće korišteni principi.

Najprije se prepoznaju dijelovi koda od interesa i potencijalne nesigurnosti koje može imati. Neke nesigurnosti mogu se prepoznati već u samom igranju igre, kao što je klijentski vođena logika, ili pretraživanjem na internetu ako je igra popularna. Zaštitni mehanizmi također su ponekad poznati od početka. S obzirom na apriornu analizu, odabire se tip alata koji bi mogao biti najkorisniji.

Ako je u interesu mijenjanje vrijednosti kao što je količina života ili bodova, a takve se vrijednosti prate na klijentu, onda je najbolje koristiti tip alata koji mijenja vrijednosti radne memorije u realnom vremenu [37]. Takvi programi koriste se tako da se najprije odabere proces od interesa na koji će se prikvačiti (proces videoigre u ovom slučaju), a onda se radi analiza svih vrijednosti u adresnom prostoru tog procesa. Traže se adrese ključnih varijabli tako da se ciljano mijenjaju njihove vrijednosti legitimnim igranjem (skupljanjem novčića, primanjem štete) i onda se radi usporedna analiza kako bi se filtrirale adrese koje ne odgovaraju promjenama ovih ključnih vrijednosti. Kada se saznala adresa varijable koja se traži, moguće je promijeniti tu vrijednost [37]. Na temelju tih adresa, programiraju se specijalizirani programi s jednostavnim grafičkim sučeljem i prodaju igračima koji žele varati. Ovakvi alati su jako često korišteni pa mnoge igre imaju mehanizme da zaštite mijenjanje memorije izvan opsega same igre. Za zaobilaženje ovakve zaštite koriste se Direct Memory Access (skr. DMA) uređaji koji mijenjanju vrijednost memorije na razini hardvera iako i za njih postoje mjere zaštite [38].

Ako je poslužitelj u potpunosti autoritativan, tada je pogodno korištenje alata koji

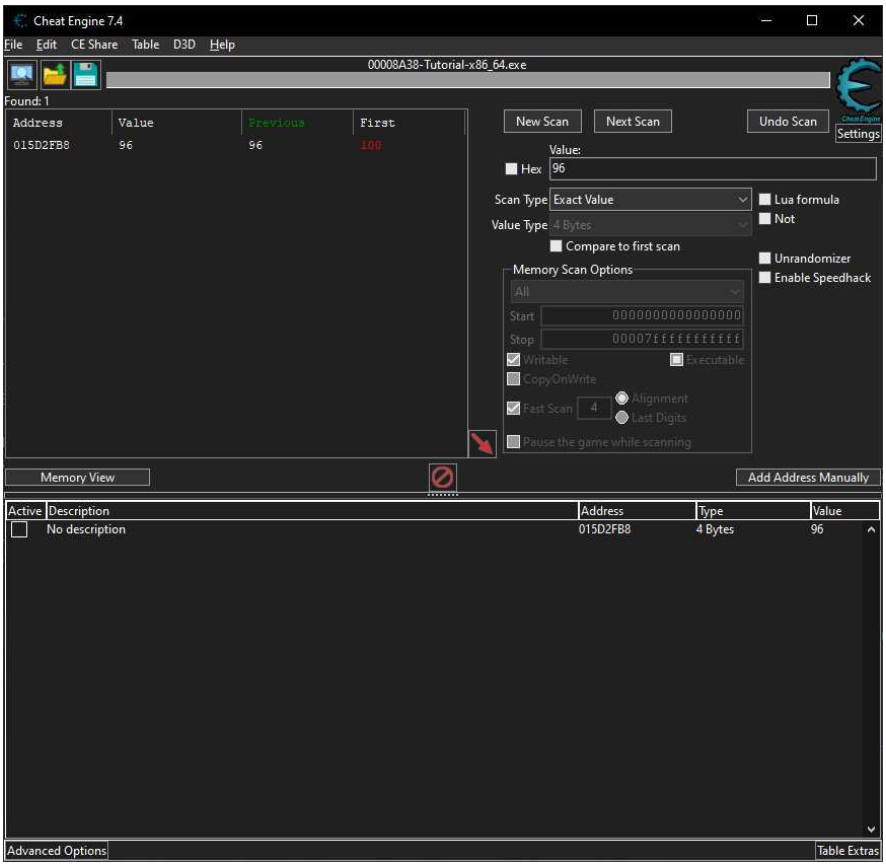
čitaju sliku zaslona, povezuju se na eksterno računalo koje radi izračune najboljih poteza i prosljeđuje unose računalu na kojoj je pokrenuta igra. Ovaj tip varanja je skup, ali u principu jednostavan i vrlo težak za otkriti.

Ako je u u interesu kompleksna promjena funkcionalnosti, potrebno je mijenjanje ili dodavanje novih datoteka igre. Tada se koriste alati koji imaju mogućnost dekompilacije i ponovne kompilacije [39]. Izvršna datoteka ili biblioteka igre naprije se učita u program. Zatim se analizira i mijenja kod blizak izvornome. Kada su dodane željene funkcionalnosti, kod se ponovno kompilira kao da je tu bio od početka. Ovakav tip varanja demonstrirat će se u potpoglavlju 6.4. Alternativno ako nije dostupno ponovno kompiliranje, ali su poznati identifikatori funkcija ili varijabli koje se žele mijenjati, koriste se mod loaderi koji pružaju svoje pozive za ugniježđivanje dodatnih datoteka.

6.2. Primjeri alati za varanje

Danas je razvijeno mnogo pomoćnih alata za varanje u igrama, no većina njih oslanja se na funkcionalnosti koje pružaju osnovni alati. Neki od njih navedeni su u ovom potpoglavlju.

6.2.1. Cheat Engine



Slika 6.1. Korisničko sučelje Cheat Engine alata [37]

Cheat Engine jedan je od najpoznatijih alata za varanje u igrama. Njegovo korisničko sučelje, vidljivo na 6.1., omogućuje korisnicima skeniranje i izmjenu vrijednosti u memoriji procesa igre, što može uključivati varanje poput modificiranja razine života, novca, municije i bodova. Također podržava ubrzanje ili usporavanje vremena u igri te automatsko pronalaženje vrijednosti koje se dinamički mijenjaju [37].

S obzirom na to da ne ovisi o razvojnim alatima igre, to ga čini generalnim alatom za varanje. Zbog toga je Cheat Engine vrlo fleksibilan, a podržava i skriptiranje, što omogućuje automatizaciju manipulacija u igri.

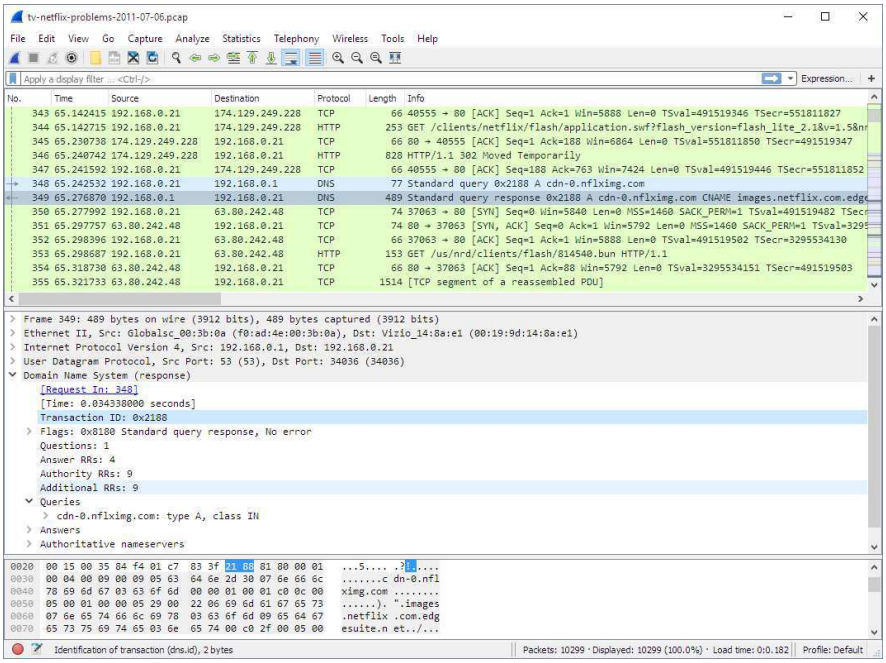
6.2.2. Ghidra

Ghidra je besplatni alat za reverzni inženjering razvijen od američke Nacionalne sigurnosne agencije (engl. National Security Agency, skr. NSA) [40]. U kontekstu varanja u igrama, koristi se za dubinsku analizu binarnih datoteka – izvršnih datoteka igre ili

njenih modula.

Iako je Ghidra primarno namijenjena analizi nativnih aplikacija, može se koristiti za razumijevanje ponašanja igre, identificiranje ključnih funkcija i potencijalnih točaka za manipulaciju. Za razliku od Cheat Enginea, Ghidra se ne koristi za izmjenu vrijednosti u stvarnom vremenu, već za statičku analizu koda [40].

6.2.3. Wireshark



Slika 6.2. Korisničko sučelje Wireshark alata [41]

Wireshark je alat za analizu mrežnog prometa koji se koristi za presretanje i ispitivanje paketa koje aplikacije šalju i primaju putem mreže [42]. U kontekstu igara, posebno umreženih igara, može se koristiti za analizu komunikacije između klijenta i poslužitelja.

S pomoću korisničkog sučelja Wiresharka (6.2.) moguće je otkriti kako igra komunicira s backendom, identificirati moguće ranjivosti, te u nekim slučajevima manipulirati mrežnim prometom putem dodatnih alata (npr. mitmproxy). Ipak, većina modernih igara koristi enkripciju i druge sigurnosne mjere koje otežavaju ovu vrstu analize.

6.2.4. dnSpy - odabrani alat

Za varanje u Unity igrama, dnSpy je jedan od najmoćnijih i najpraktičnijih alata. Radi se o .NET dekompilatoru i debuggeru koji omogućuje korisnicima da pregledaju, analiziraju, mijenjaju i ponovno kompiliraju .NET aplikacije, uključujući one napravljene u Unityju.

Unity igre su najčešće kompilirane u programskom jeziku C# i koriste .NET okruženje, zbog čega je dnSpy izuzetno prikladan alat za njihovu analizu. Omogućuje lako pronalaženje i izmjenu metoda, varijabli i logike igre bez potrebe za prijašnjim posjedovanjem izvornog koda.

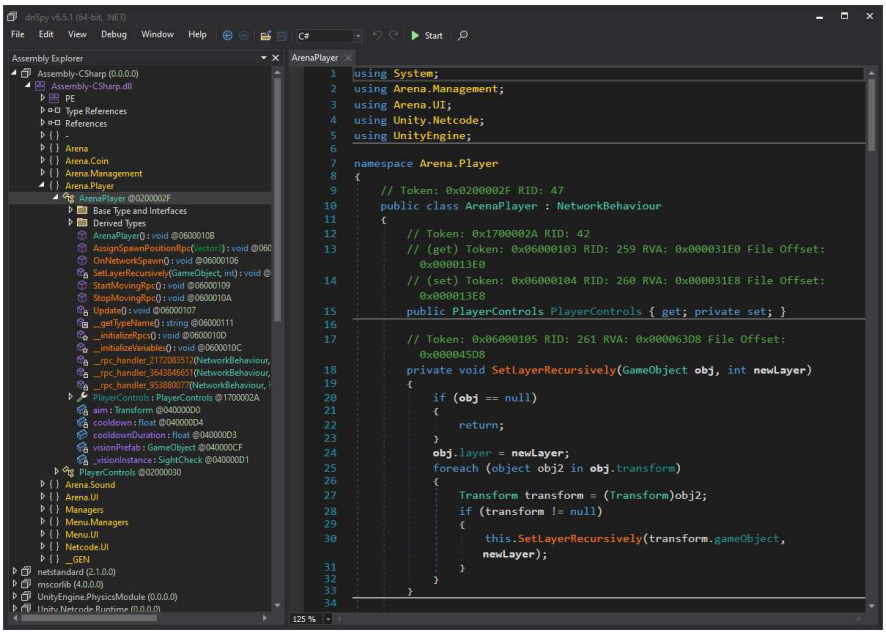
Kombinacija jednostavnosti korištenja, potpune kontrole nad .NET kodom te mogućnosti debuggiranja čini dnSpy idealnim alatom za analizu i modifikaciju Unity igara zbog čega je on odabrani alat za demonstraciju varanja u ovome radu.

6.3. Korištenje dnSpy alata

Program dnSpy slobodno je dostupan za preuzimanje i korištenje. Originalni alat je arhiviran pa se danas koriti njegov nasljednik dnSpyEx koji se preuzima s njegove Github stranice [39].

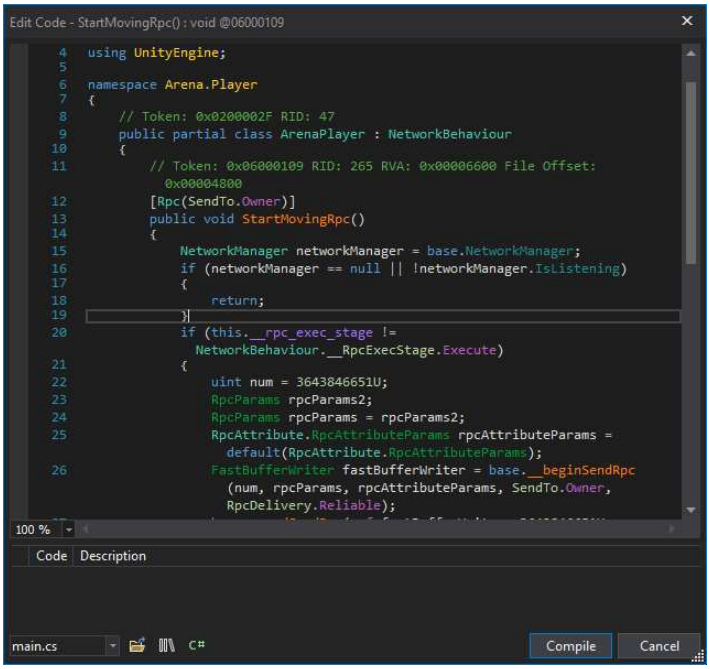
Nakon pokretanja alata potrebno je otvoriti potrebnu Dynamic-link library (skr. dll) datoteku. U slučaju Unity igara, to će gotovo uvijek biti Assembly-CSharp.dll koja se nalazi u build_Data/Managed mapi izgrađene videoigre. Otvaranjem te datoteke povezat će se i druge sustavne biblioteke koje omogućuju dekompiliranje i ponovno kompiliranje glavne datoteke, no one se neće mijenjati.

Većina funkcionalnosti razvijenih u C# skriptama biti će dostupna na izmjenu otvaranjem ove datoteke. Na slici 6.3. vidljiv je dekompilirani kod i hijerarhija skripta *Pa-Pa!* videoigre, koja je razvijena u sklopu ovog rada.



Slika 6.3. Uvid u dekompilirani kod s pomoću dnSpy alata

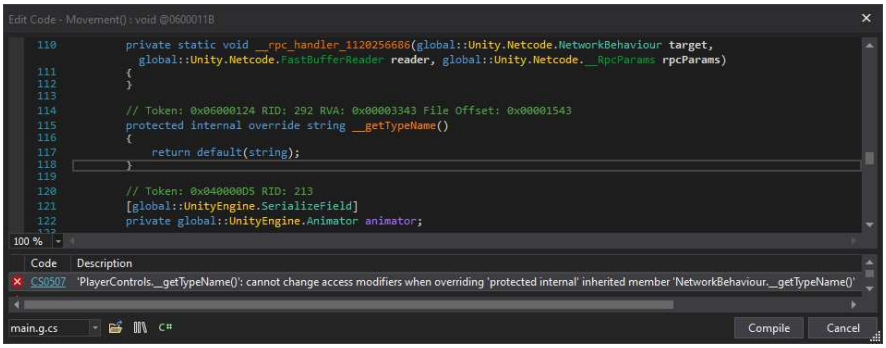
Nakon analiziranja koda i traženja nesigurnosti moguće je izmijeniti ili čak dodati nove funkcije i razrede. Desnim klikom na funkciju ili razred otvara se kontekstni izbornik koji, uz mnoge druge opcije, pruža uređivanje koda. Na slici 6.4. prikazan je prozor uređivača koda.



Slika 6.4. Mijenjanje koda u dnSpy-u

Kada je nova funkcionalnost spremna, potrebno je pritisnuti "Compile". U većini slu-

čajeva, tijekom isprobavanja alata, došlo je do pogreške pri kompilaciji prikazanoj na 6.5. Jednostavno rješenje je bilo zakomentirati problematičnu funkciju kojoj je, navodno, izmijenjen modifikator. Nakon uspješne kompilacije ta se funkcija sama ponovno generira pa ne dolazi do greške tijekom izvođenja igre.



Slika 6.5. Greška pri kompiliranju

Za kraj, potrebno je spremiti promijenjenu dll datoteku. Moguće je pregaziti originalnu ili spremiti ju kao novu. Ponovnim izgrađivanjem izvršnih datoteka u Unityu izgradit će se originalna, nepromijenjena igra, zato što dnSpy ne mijenja izvorne funkcionalnosti već samo izvršne datoteke.

6.4. Primjena varanja i rezultati

U ovome potpoglavlju, demonstrira se analiza koda i iskorištavanje ranjivosti. Priloženi kodovi programirani su s pomoću dnSpy alata i nadjačavaju izvorne funkcionalnosti.

6.4.1. Nesigurno programirana logika novčića

Logika novčića 5.9. poprilično je nesigurno isprogramirana. Iz koda je vidljivo da je skupljanje novčića zamišljeno tako da provjeru vrši samo poslužitelj. Međutim, skupljanje novčića ostvareno je tako da svaki klijent, ako nije ujedno i poslužitelj, svojevoljno preskače poziv poslužiteljske funkcije za dodjelu bodova. Micanjem te restrikcije, klijent može proizvoljno pozvati funkciju skupljanja, bez obzira na činjenicu je li ga zapravo pokupio. Ta se ranjivost može iskoristiti tako da klijent odmah pri stvaranju objekta novčića, na svojoj instanci pozove poslužiteljsku funkciju skupljanja "CoinCollectedRpc". To je demonstrirano na isječku koda 6.1.

Assembly-CSharp > CollectibleCoin > Start

```
12 private void Start()
13 {
14     this.CoinCollectedRpc(base.NetworkManager.LocalClientId, default(
       RpcParams));
15 }
```

Listing 6.1. Trenutno skupljanje novčića pri stvaranju

Dodatnim analiziranjem ponašanja novčića, utvrđeno je da je svaki novčić moguće više puta skupiti pozivanjem funkcije skupljanja. Kod 6.2. prikazuje iskorištavanje takve ranjivosti vrednovanjem svakog novčića nekoliko puta više.

Assembly-CSharp > CollectibleCoin > OnTriggerEnter

```
13 private void OnTriggerEnter(Collider other)
14 {
15     if (!this._pickupable)
16     {
17         return;
18     }
19     ArenaPlayer arenaPlayer;
20     if (other.TryGetComponent<ArenaPlayer>(out arenaPlayer))
21     {
22         this.CoinCollectedRpc(arenaPlayer.OwnerClientId, default(
       RpcParams));
23         this.CoinCollectedRpc(arenaPlayer.OwnerClientId, default(
       RpcParams));
24 
25         ...
26         this.CoinCollectedRpc(arenaPlayer.OwnerClientId, default(
       RpcParams));
27         this.CoinCollectedRpc(arenaPlayer.OwnerClientId, default(
       RpcParams));
28     }
29 }
```

Listing 6.2. Uvišestručena vrijednost novčića

6.4.2. Detekcija i wallhack

Vidljivo iz 5.8., detekcija vidljivosti provjerava se na klijentu, a u slučaju pozitivnog rezultata, poziva se funkcija poslužitelja. Zaobilaženjem uvjeta detekcije, moguće je odmah na početku runde "pronaći" protivnika i osvojiti 5 bodova. Igrača koji vara na ovaj način nemoguće je poraziti, ali zato je vrlo lagano saznati da se igrač koristi alatima za varanje. Puno suptilniji način varanja, a skoro pa jednako moćan u kontekstu ove videoigre, je korištenje wallhack varanja. Kao što je i opisano u poglavlju 3.2., mnoge igre ranjive su na wallhack, pa tako i Pa-Pa!.

U sljedećem kodu razvijena je funkcionalnost koja omogućava viđenje protivnika kroz zidove. Za ostvarenje te funkcionalnost upravlja se parametrima scenskog osvjetljenja i kamere s naglaskom na mijenjanje vidljivih "layera" (6.3.). Uključivanjem wallhacka, vidljiv je samo protivnik.

`Assembly-CSharp > SightCheck > SetWallhack`

```
11 public void SetWallhack(bool activated)
12 {
13     Camera component = base.transform.GetChild(0).GetComponent<Camera
       >();
14     int mask = LayerMask.GetMask(new string[] { "Default", "
       TransparentFX", "Ignore Raycast", "Water", "UI", "Enemy" });
15     int mask2 = LayerMask.GetMask(new string[] { "Enemy" });
16     RenderSettings.fog = !activated;
17     component.cullingMask = (activated ? mask2 : mask);
18     component.farClipPlane = (float)(activated ? 500 : 40);
19 }
```

Listing 6.3. Dodana wallhack funkcionalnost

Pozivanje novododane funkcije ostvaruje se u kodu 6.4. Dodane su nove provjere ulaza koje omogućuju praktično korištenje opisanog mehanizma. Dodatno, usporedbom s 5.8., više nema logičke provjere vidljivosti, već se neupitno poziva poslužiteljska funkcija za kraj igre. U stvarnosti, nema prevelike potrebe za ostvarenjem obiju ovih funkcionalnosti.

Assembly-CSharp > ArenaPlayer > Update

```
14 private void Update()
15 {
16     if (base.IsOwner && this._visionInstance.TurnedOn)
17     {
18         this._visionInstance.transform.rotation = this.aim.rotation;
19         this.cooldown -= Mathf.Max(0f, Time.deltaTime);
20         if (Input.GetMouseButtonDown(0) && this.cooldown <= 0f)
21         {
22             this.cooldown = this.cooldownDuration;
23             GameManager.Instance.PahPahRpc(base.NetworkManager.
    LocalClientId);
24             UIManager.Instance.FlashAnimation(new Color(1f, 1f, 1f,
    0.5f), 0.75f);
25             return;
26         }
27         if (Input.GetKeyDown(KeyCode.H))
28         {
29             this._visionInstance.SetWallhack(true);
30         }
31         if (Input.GetKeyDown(KeyCode.J))
32         {
33             this._visionInstance.SetWallhack(false);
34         }
35     }
36 }
```

Listing 6.4. Neprovjeravanje vidljivosti i upravljanje s wallhack funkcionalnosti

Sljedeće slike demonstriraju korištenje wallhack mehanizma. Pritiskom na tipke H i J, uključuje se (6.7.), odnosno isključuje mehanizam (6.6.).

Slika 6.6. Isključeni wallhack

Slika 6.7. Uključeni wallhack

Slika 6.8. prikazuje perspektivu igrača koji je upravo izgubio rundu. Iako gleda protivniku u leđa, protivnik ga je i dalje uspio vidjeti, što u normalnim okolnostima ne bi bilo moguće.

Slika 6.8. Falsificirana detekcija, perspektiva drugog igrača

6.4.3. Klijentska kontrola lika

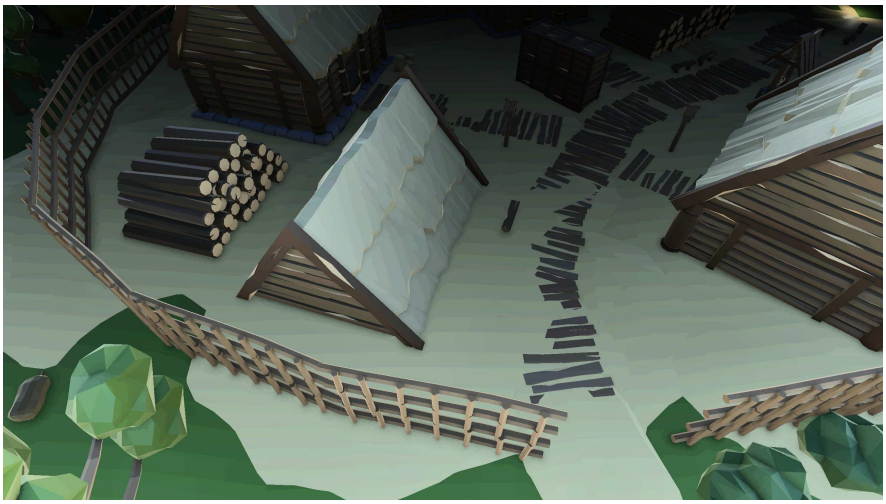
Zbog klijentski vođene logike kretanja lika (5.7.), mogu se mijenjati i parametri kretanja. U priloženom kodu 6.5., brzina kretanja i snaga skakanja upeterostručeni su.

`Assembly-CSharp > PlayerControls > Start`

```
15 private void Start()
16 {
17     this._movementDirection = base.transform.rotation * Vector3.
       forward;
18     this.speed *= 5f;
19     this.jumpPower *= 5f;
20 }
```

Listing 6.5. Promijenjeni parametri kretanja

Veća snaga skakanja omogućuje dostizanje puno većih visina skakanjem. Iz sljedećih slika vidljivo je kako to izgleda iz perspektive igrača koji skače (6.9.) i iz perspektive njegova protivnika (6.10.).



Slika 6.9. Visoko skakanje omogućeno varanjem

Slika 6.10. Perspektiva drugog igrača tijekom visokog skakanja

6.4.4. Zlouporaba ovlasti domaćina

S obzirom na to da domaćin ima ulogu klijenta i poslužitelja, njegove su mogućnosti varanja gotovo neograničene. Moguće je u potpunosti preuzeti kontrolu nad svakim aspektom igre preuzimanjem vlasništva svih objekata. Uz to, domaćin može ignorirati sve naredbe koje šalje gost. Bez programa protiv varanja nije moguće ograničiti domaćinove mogućnosti varanja u topologiji klijent-poslužitelj. U nastavku se demonstriraju neki primjeri zlouporabe ove moći.

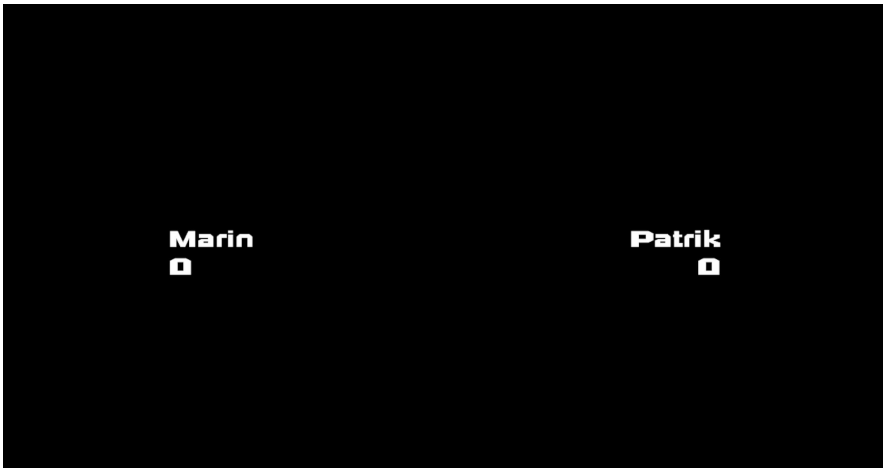
U prvom primjeru (6.6.), promijenjeno je bodovanje tako da gost nikada ne može dobiti bodove. Na kraju runde (6.11.) vidljivo je da nitko nije osvojio bodove, što je inače nemoguće. To je zato što je gost pobijedio rundu, ali nije nagrađen bodovima. Ovo nije

suptilan način varanja; mogao bi se učiniti suptilnijim tako da svaki skupljeni novčić ima određenu šansu da se ne pribroji bodovima gosta.

`Assembly-CSharp > GameManager > AssignPointsRpc`

```
38 if (clientId == this.HostId)
39 {
40     this._hostPoints += points;
41 }
42 else if (clientId == this.GuestId)
43 {
44     this._guestPoints = this._guestPoints;
45 }
```

Listing 6.6. Nevredovanje bodova gosta



Slika 6.11. Kraj runde bez dodijeljenih bodova

Programiranjem deterministički određenog redoslijeda postavljanja likova na početku runde, umjesto nasumičnog, domaćinu može dati veliku prednost jer uvijek zna gdje je protivnik postavljen, a gostu je teško prepoznati da se radi o varanju. U sljedećem primjeru (6.7.) demonstrirano je postavljanje protivnika na proizvoljnu lokaciju, a na slici 6.12. vidljivo je da je gost postavljen na vrh kuće do kojeg nije mogao doći kretanjem. Lagano se može isprogramirati i promjenjiv deterministički redoslijed na ovaj način.

Assembly-CSharp > GameManager > SpawnPlayerRpc

```
35 bool flag = base.NetworkManager.LocalClientId == senderClientId;
36 ArenaPlayer component = base.NetworkManager.SpawnManager.
       InstantiateAndSpawn(this.playerPrefab, senderClientId, true, false
       , false, default(Vector3), default(Quaternion)).GetComponent<
       ArenaPlayer>();
37 if (flag)
38 {
39     this._hostPlayer = component;
40     this._hostPlayer.AssignSpawnPositionRpc(this._hostSpawnPoint);
41     return;
42 }
43 this._guestPlayer = component;
44 this._guestPlayer.AssignSpawnPositionRpc(new Vector3(21.5, 10, -14.5))
       ;
```

Listing 6.7. Proizvoljno postavljanje gosta

Slika 6.12. Gost namjerno postavljen na vrh krova kuće

7. Sprječavanje demonstriranih varanja

7.1. Krpanje ranjivosti

Ranjivosti poput nesigurnih novčića i klijentske kontrole kretanja jednostavno se mogu riješiti tako što se promjene stanja odrađuju samo na poslužiteljskoj strani. Igrači bi samo trebali slati naredbe, a poslužitelj odlučuje kako će one utjecati na igru.

Za rješenje vidljivosti već postoji implementirana funkcija koja provjerava je li protivnik vidljiv igraču. Ako bismo ju koristili na poslužiteljskoj strani da odlučuje treba li nekom igraču poslati informacije o protivniku mogao bi se riješiti problem wallhacka. Međutim ona nije namijenjena da se koristi prečesto jer je performativno zahtjevna. Uz to, čim se klijentu pošalje informacija o protivniku, on je tada informiran da vidi protivnika i automatiziranom skriptom može odmah osvojiti rundu.

Što se tiče ovlasti domaćina, jednostavno treba koristiti drugu arhitekturu mreže. Ako je igra namijenjena kao natjecateljska, onda poslužiteljska logika ne bi trebala biti u kontroli nijednog igrača.

7.2. Programi protiv varanja

Kako se ne može vjerovati klijentu da neće modificirati svoju instancu igre, a nije praktično u potpunosti osigurati autoritativnost poslužitelja, promotrit će se mogućnost implementacije programa protiv varanja.

Temeljne sigurnosti koje treba implementirati su otpornosti na debuggiranje, promjenu datoteka igre, ugrađivanje novih datoteka i promjenu memorijskih vrijednosti [43]. Unaprijed je poznato koje datoteke igra treba posjedovati, prema tome moguće je

proći kroz cijeli datotečni sustav igre i pronaći uljeze, ako ih ima. Za provjeru modifikacije legitimnih datoteka koristi se provjera kontrolnim zbrojem (engl. checksum). Ako je kontrolni zbroj jednak unaprijed izračunanom, gotovo je sigurno da datoteke nisu mijenjane. Protiv čitanja memorijskih vrijednosti potrebno je implementirati provjeru uključenih procesa tijekom rada igre, a za otpornost prema debuggiranju potrebna je ustrajna analiza pokrenutih procesa [43].

Težak dio posla je zaštititi sam program protiv varanja od napada. Za jaku zaštitu program bi trebao biti invazivan, privilegiran i koristiti napredne kriptografske mehanizme. Sve to čini implementaciju ovakvog programa težu od primjene dobrih sigurnosnih praksi na razvoju same igre.

7.3. Detekcija varanja na poslužitelju

Koristeći opisana varanja, moguće je osigurati da protivnik nema nikakve šanse pobijediti, no iz perspektive drugog igrača jasno je da se radi o varanju. Dodavanjem mehanizma prijave varanja, moderatori bi mogli provjeriti tijek igre i uvjeriti se da se radi o varanju. Ako bi se implementirao i program protiv varanja na klijentu, on bi također vodio komunikaciju s njegovim sugovornikom na poslužitelju.

Promatrajući samo stanje na poslužitelju, također je moguće razviti neke mehanizme za detekciju. Ako se kretanje izvršava na klijentu, poslužitelj može računati brzinu kretanja (ili skakanja) i ako procijeni da je igračev lik dosegnuo nemoguće brzine, može zaključiti da se radi o varanju. Dodatno, kao provjeru za kraj runde, može naknadno provjeriti je li protivnički igrač zaista bio u vidnom polju pobjedničkog igrača. Što se tiče zaštite protiv wallhacka, mogu se monitorirati smjer kretanja i pogleda klijenta. Igrači koji koriste wallhack mijenjat će svoje kretanje u skladu s time. Razvitkom umjetne inteligencije koja prati kretanje likova u usporedbi s bazom igrača za koju se zna da su varali, može se procijeniti vara li neki igrač ili ne.

7.4. Intermediate Language To C++

Većinski korišteni backend za skriptiranje u Unityju je Mono, međutim u postavkama izgradnje projekta moguće je promijeniti backend na Intermediate Language To C++

(skr. IL2CPP). Kao što mu ime nalaže, on prevodi međujezik kompiliranog C# koda u C++ kod. Od tuda, C++ prevodi se u nativni binarni kod. Korištenje IL2CPP-a kao posljedicu ima bolje performanse i kompatibilnost s različitim platformama [44].

Nenamjerna posljedica korištenja IL2CPP-a, iako ga neki koriste upravo u tu svrhu, je veća zaštita od dekompiliranja. Iako je dekompiliranje zaista otežano korištenjem ovog backenda, Unity uz binarne datoteke, sprema i određene meta podatke. Uz korištenje dodatnih alata za dnSpy ti meta podaci iskorištavaju se kako bi dekompilirali i ovako izgrađenu videoigru [45].

7.5. Alternativni pogonski sustavi za igre

Prirodno se postavlja pitanje: Je li korištenjem drugog pogonskog sustava za razvoj videoigre moguće otežati varanje? Odgovor je djelomično potvrdan. Primjerice Unreal Engine koristi C++ za razvoj videoigara te ga je time teže dekompilirati. Međutim s vremenom su se razvili mnogi alati koji pomažu pri reverznom inženjeringu igara napravljenim u Unreal Engineu [46]. Još jedan popularni pogonski sustav je Godot koji je posebice lagan za dekompilirati [47].

Odabir pogonskog sustava, dakle, ne može spriječiti varanje, niti ga značajno otežati. Za bilo kakvu vrstu izgrađenih datoteka videoigre, postoje alati koji ju mogu analizirati i mijenjati.

8. Zaključak

S pregledom rastućeg tržišta videoigara i aktualne igre nadmudrivanja između programera videoigara i igrača koji varaju, zaključuje se da je razumijevanje mehanizama varanja i njegovog sprječavanja od velike važnosti kod razvijanja uspješne umrežene videoigre.

Odabir arhitekture umreženosti i principi komunikacije među igračima i poslužiteljima predstavlja temelj za izgradnju sigurnosnih mehanizama. Najbitnije je osigurati autoritativnost poslužitelja, no ponekad to nije praktično u pogledu razvijane videoigre. U tim slučajevima klijenti će moći iskorištavati kontrolu koju imaju i izmijeniti zamišljenu logiku. Čak i slučajevima igara s autoritativnim poslužiteljem, varanje ne može biti sasvim spriječeno. Zbog teškoće osiguravanja da se klijentska aplikacija ne zloupotrebljava, razvijaju se programi protiv varanja s invazivnim i privilegiranim funkcijama.

U kontekstu Pa-Pa! videoigre, koja je razvijena s ciljem istraživanja i demonstriranja nesigurnosti, prikazani su dijelovi koda za umrežavanje, upravljanje likom i bodovanje. Korištenjem bilo kakvog oblika reverznog inženjeringa, što je u slučaju ovog rada bilo jednostavno dekompiliranje korištenjem dnSpy alata, moguće je prepoznati ranjivosti u ključnim dijelovima koda. S izmjenom pojedinih dijelova logike u kontroli klijenta, igraču je omogućena zlouporaba sustava bodovanja, kretanja i vidljivosti drugog igrača. Skoro svaki tip varanja ovako razvijen omogućuje igraču nepobjedivost, a mogući su i suptilniji načini za varanje koji i dalje donose veliku prednost.

Retrospektivnim pogledom na razvoj igre i postupak varanja, razmatrali su se postupci krpanja određenih ranjivosti. S tim postupcima ograničio bi se opseg mogućnosti varanja, ali neki važni aspekti, kao što je detekcija vidljivosti drugog igrača, nemaju jednostavan popravak. U tim slučajevima razmatra se korištenje programa protiv varanja

na klijentu i poslužitelju, svaki od kojih može značajno pridonijeti, ali su i tehnički zahtjevni za implementaciju, dok je razmatranje korištenja drugih razvojnih alata zaključeno s time da to nije učinkovit način za sprječavanje varanja.

Literatura

[1] H. Pandey, https://blog.hathora.dev/peer-to-peer-vs-client-server-architecture/, [mrežno; stranica posjećena: lipanj 2025.].

[2] P. Vigier, https://pvigier.github.io/2019/09/08/beginner-guide-game-networking.html, [mrežno; stranica posjećena: lipanj 2025.].

[3] Development Relations Team, Pubnub, https://www.pubnub.com/guides/unity/, [mrežno; stranica posjećena: lipanj 2025.].

[4] Unity, https://docs-multiplayer.unity3d.com/netcode/current/terms-concepts/client-server/, [mrežno; stranica posjećena: lipanj 2025.].

[5] PlayerAuctions, https://www.playerauctions.com/about/sell-game-accounts/, [mrežno; stranica posjećena: lipanj 2025.].

[6] E. Conroy, M. Kowal, A. J. Toth, i M. J. Campbell, "Boosting: Rank and skill deception in esports", *Entertainment Computing*, sv. 36, str. 100393, 2021. https://doi.org/https://doi.org/10.1016/j.entcom.2020.100393

[7] G. Ma, https://medium.com/exponential-era/the-million-dollar-video-game-cheating-market-05150db8fe4c, [mrežno; stranica posjećena: lipanj 2025.].

[8] Market Research Intellect, https://www.marketresearchintellect.com/nl/blog/anti-cheat-solutions-drive-the-future-of-online-gaming-security/, [mrežno; stranica posjećena: lipanj 2025.].

[9] Manish Sharma, https://www.linkedin.com/pulse/securing-your-game-essential-security-protocols-manish-sharma-myyuf/, [mrežno; stranica posjećena: lipanj 2025.].

[10] zarr, https://forums.bhvr.com/dead-by-daylight/discussion/303696/busting-some-hit-registration-myths, [mrežno; stranica posjećena: lipanj 2025.].

[11] A. Ching, https://medium.com/mighty-bear-games/what-are-server-authoritative-realtime-games-e2463db534d1, [mrežno; stranica posjećena: lipanj 2025.].

[12] G. Gambetta, https://www.gabrielgambetta.com/client-server-game-architecture.html, [mrežno; stranica posjećena: lipanj 2025.].

[13] S. Park, A. Ahmad, i B. Lee, "Blackmirror: Preventing wallhacks in 3d online fps games", u *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020., str. 987–1000.

[14] Epiqua, Fab, https://www.fab.com/listings/15c947ba-a68f-4962-a14c-c3753484c6dc, [mrežno; stranica posjećena: lipanj 2025.].

[15] M. Chen, "Ai cheating versus ai anti-cheating: A technological battle in game", *Applied and Computational Engineering*, sv. 73, str. 222–227, 2024.

[16] S. A. Dhinge, S. G. Sukum, H. S. Ranjane, R. R. Rajput, i J. H. Jadhav, "Anti-cheat in fps game for aimbot detection", u *2024 Intelligent Systems and Machine Learning Conference (ISML)*, 2024., str. 124–127. https://doi.org/10.1109/ISML60050.2024.11007386

[17] L. Artichoke, https://www.seaofthieves.com/community/forums/topic/167498/provide-option-to-play-without-eac, [mrežno; stranica posjećena: lipanj 2025.].

[18] The Linux Information Project, https://www.linfo.org/kernel.html, [mrežno; stranica posjećena: lipanj 2025.].

[19] Andrew Hyatt, https://ritcsec.wordpress.com/2022/08/03/security-concerns-about-kernel-level-anti-cheat-in-video-games/, [mrežno; stranica posjećena: lipanj 2025.].

[20] C. Dorner i L. D. Klausner, "If it looks like a rootkit and deceives like a rootkit: A critical examination of kernel-level anti-cheat systems", u *Proceedings of the 19th*

International Conference on Availability, Reliability and Security, 2024., str. 1–11.

[21] S. McEvoy, https://www.gamesindustry.biz/respawn-blocks-apex-legends-on-platforms-using-linux-including-steam-deck, [mrežno; stranica posjećena: lipanj 2025.].

[22] N. Poor, "Computer game modders' motivations and sense of community: A mixed-methods approach", *New Media & Society*, sv. 16, br. 8, str. 1249–1267, 2014. https://doi.org/10.1177/1461444813504266

[23] T. Sihvonen, *CULTURAL AND COMMERCIAL APPROPRIATION*. Amsterdam University Press, 2011., str. 37–86. [Mrežno]. Adresa: http://www.jstor.org/stable/j.ctt46mt37.5

[24] CurseForge, https://www.curseforge.com/minecraft/search?page=1&pageSize=20&sortBy=relevancy&class=mc-mods, [mrežno; stranica posjećena: lipanj 2025.].

[25] Hypixel Support, https://support.hypixel.net/hc/en-us/articles/6472550754962-Hypixel-Allowed-Modifications, [mrežno; stranica posjećena: lipanj 2025.].

[26] X Minecraft Launcher, https://xmcl.app/en/guide/modloader, [mrežno; stranica posjećena: lipanj 2025.].

[27] Microsoft, https://devblogs.microsoft.com/visualstudio/decompilation-of-c-code-made-easy-with-visual-studio/, [mrežno; stranica posjećena: lipanj 2025.].

[28] Stack Overflow, https://softwareengineering.stackexchange.com/questions/185602/how-is-c-c-more-difficult-to-decompile-than-c, [mrežno; stranica posjećena: lipanj 2025.].

[29] Microsoft, https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview, [mrežno; stranica posjećena: lipanj 2025.].

[30] Unity, https://unity.com/products/netcode, [mrežno; stranica posjećena: lipanj 2025.].

[31] ——, https://docs-multiplayer.unity3d.com/netcode/current/about/, [mrežno; stranica posjećena: lipanj 2025.].

[32] ——, https://docs.unity3d.com/Packages/com.unity.netcode@1.0/manual/index.html, [mrežno; stranica posjećena: lipanj 2025.].

[33] R. Lord, https://www.richardlord.net/blog/ecs/why-use-an-entity-framework.html, [mrežno; stranica posjećena: lipanj 2025.].

[34] Unity, https://unity.com/solutions/gaming-services, [mrežno; stranica posjećena: lipanj 2025.].

[35] ——, https://assetstore.unity.com/, [mrežno; stranica posjećena: svibanj 2025.].

[36] incompetech, https://incompetech.com/music/royalty-free/music.html, [mrežno; stranica posjećena: svibanj 2025.].

[37] PandaSt0rm, https://www.hackthebox.com/blog/intro-to-gamepwn-aka-game-hacking#mcetoc_1fump7bcql7l, [mrežno; stranica posjećena: lipanj 2025.].

[38] dma, https://cires1.colorado.edu/jimenez-group//QAMSResources/Docs/DMAFundamentals.pdf, [mrežno; stranica posjećena: lipanj 2025.].

[39] dnSpyEx, https://github.com/dnSpyEx/dnSpy, [mrežno; stranica posjećena: travanj 2025.].

[40] National Security Agency, https://github.com/NationalSecurityAgency/ghidra, [mrežno; stranica posjećena: lipanj 2025.].

[41] Wireshark, https://www.wireshark.org/docs/wsug_html_chunked/ChUseMainWindowSection.html, [mrežno; stranica posjećena: lipanj 2025.].

[42] ——, https://www.wireshark.org/about, [mrežno; stranica posjećena: lipanj 2025.].

[43] I. Segalla, https://dev.to/igorsegallafa/tips-for-writing-an-anti-cheat-4m6k, [mrežno; stranica posjećena: lipanj 2025.].

[44] Unity, https://docs.unity3d.com/6000.1/Documentation/Manual/scripting-backends-il2cpp.html, [mrežno; stranica posjećena: lipanj 2025.].

[45] Perfare, https://github.com/Perfare/Il2CppDumper, [mrežno; stranica posjećena: lipanj 2025.].

[46] UE Explorer, https://github.com/UE-Explorer/UE-Explorer, [mrežno; stranica posjećena: lipanj 2025.].

[47] Godot Mod Loader Wiki, https://wiki.godotmodding.com/guides/modding/tools/decompile_games/, [mrežno; stranica posjećena: lipanj 2025.].

Sažetak

Analiza i demonstracija alata za varanje u umreženim videoigrama

Marin Prusac

Inicijalnim pregledom literature na temu umrežavanja i varanja u umreženim videoigrama u industriji, analizirani su alati i metode za varanje u umreženim videoigrama te su demonstrirane najčešće ranjivosti kroz vlastitu mrežnu igru razvijenu u Unityju. Prikazano je kako alati poput dnSpy omogućuju izmjenu koda i iskorištavanje sigurnosnih propusta, primjerice manipulaciju bodovima, wallhack i zloupotrebu ovlasti domaćina. Predložene su osnovne mjere zaštite, uključujući jaču poslužiteljsku kontrolu i primjenu programa protiv varanja. Rad daje praktičan pregled problema varanja i mogućnosti njihove prevencije u modernim mrežnim igrama.

Ključne riječi: umrežene videoigre; varanje u igrama; Unity; klijent-poslužitelj arhitektura; reverzni inženjering; dnSpy; zaštita od varanja

Abstract

Analysis and demonstration of cheating tools in networked videogames

Marin Prusac

With an initial review of literature on networking and cheating in networked video games in the industry, tools and methods for cheating in such games were analyzed, and the most common vulnerabilities were demonstrated through a custom networked game developed in Unity. It was shown how tools like dnSpy allow code modification and exploitation of security flaws, such as point manipulation, wall hacks, and abuse of host privileges. Basic protection measures were proposed, including stronger server-side control and the use of anti-cheat software. The paper provides a practical overview of cheating issues and prevention strategies in modern networked games.

Keywords: multiplayer games; game cheating; Unity; client-server architecture; reverse engineering; dnSpy; anti-cheat protection

Privitak A: Akreditacija preuzetih slobodnih resursa

A1. Audio isječci

Sljedeći audio isječci akreditirani su na zahtjevani način.

Black Vortex Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

"Cool Vibes" Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 4.0 License http://creativecommons.org/licenses/by/4.0/

Nerves Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

Crypto Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

Hidden Agenda Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

Ossuary 7 - Resolve Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

River of Io Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

Satiate - only strings Kevin MacLeod (incompetech.com) Licensed under Creative Commons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

Clash Defiant Kevin MacLeod (incompetech.com) Licensed under Creative Com-

mons: By Attribution 3.0 License http://creativecommons.org/licenses/by/3.0/

A2. 3D modeli

https://assetstore.unity.com/packages/3d/environments/landscapes/rpg-poly-pack-lite-148410
https://assetstore.unity.com/packages/3d/characters/quarter-view-3d-action-assets-pack-188720