A picture containing text

Description automatically generatedLogo, company name

Description automatically generated

3D RPG voxel-based in Unity

Autor: Ciuciu Anca-Maria, 333 AA

An: III

Cuprins

[**1.** **INTRODUCERE** 4](#_Toc104208710)

[**1.1** **Context** 4](#_Toc104208711)

[**1.2** **Problemă** 4](#_Toc104208712)

[**1.3** **Obiective** 4](#_Toc104208713)

[**2.** **SUPORT TEHNIC ȘI ABORDĂRI EXISTENTE** 5](#_Toc104208714)

[**2.1 Tehnologiile folosite** 5](#_Toc104208715)

[**2.2 Implementări similare** 6](#_Toc104208716)

[**3.** **IMPLEMENTARE ȘI MULTIMEDIA** 8](#_Toc104208717)

[**3.1 Elemente de GUI, design și audio** 8](#_Toc104208718)

[**3.2 Networking și Multiplayer** 12](#_Toc104208719)

[**4.** **INTERACȚIUNEA CU UTILIZATORUL** 13](#_Toc104208720)

[**4.1 Mecanica jocului** 13](#_Toc104208721)

[**4.2 Decor** 15](#_Toc104208722)

[**4.3 Inamici AI** 17](#_Toc104208723)

[**5.** **CONCLUZII** 17](#_Toc104208724)

**SINOPSIS**

Dezvoltarea de jocuri video a inceput să ia amploare în ultimii ani prin prisma apariției unei serii de game engine-uri ce reușesc să abstractizeze noțiuni avansate de programare. Astfel, dezvoltatorii începători dispun de avantaje precum folosirea Drag & Drop pentru inserarea unui obiect într-o scenă în loc de încărcarea modelului și randarea efectivă a acestuia, autogenerarea în mod dinamic a unei structuri orientate obiect, generarea unei instanțe de networking și multe altele. Printre cele mai cunoscute framework-uri destinate acestui subiect, regăsim: Unity, GameMaker și Godot. Totuși acest avantaj al facilitării procesului de dezvoltare trebuie analizat în contextul afectării performanței jocului.

Având în vedere faptul că perspectiva abordată în cadrul jocului este cea 3D, folosind Unity, se va analiza geometria abordată în contextul procesării pe GPU ce implică folosirea triunghiurilor pentru generarea modelelor. De asemenea, va fi abordată și problema sincronizării pe rețea pentru propagarea în joc a alegerilor jucătorului, animații si comunicare prin chat.

Concluziile rezultate din acest proiect vor urmări o analiză din punct de vedere al perfomanței și posibile îmbunătățiri, având drept reper jocul ce a reprezentat principala sursă de inspirație.

# **INTRODUCERE**

## **Context**

Tema acestui proiect face referire la jocurile de tip Role Playing, voxel based, astfel că va fi dezvoltat în maniera unei reinterpretări a unui joc deja existent, destul de popular. Obiectivele principale ale acestei inițiatitve constau în folosirea unui framework, mai degrabă high level, relativ ușor de folosit pentru amploarea unui joc 3D multiplayer, ce dispune și de sincronizări pe rețea. Desigur că acest avantaj al abstractizării unor concepte mai profunde precum protocoalele de comunicație, pipeline-ul de grafică (trecerea de la vertex shader la pixel shader și mai apoi la pixel rendering prin procesare geometrică, noțiune care în Unity este cuprinsă de conceptul de render pipelines[24]), importarea și încărcarea modelelor 3D la runtime și multe altele ce vor mai fi amintite și pe parcurs, atrage după sine un dezavantaj consistent de performanță, ce poate trece neobservat de un dezvoltator neexperimentat ca mine, existând anumite repercursiuni în unele scenarii.

## **Problemă**

Toate aceste neajunsuri ale game engine-ului Unity, observate de către mine, vor fi analizate atent pentru a contura și o comparație între procesul de dezvoltare folosind un game engine free of use, open-source, pus la dispoziția oricui, acoperind o gamă extrem de largă de jocuri și procesul anevoios ce presupune programarea unui game engine custom, simplist în primă instanță, după care urmează procesul efectiv de programare a jocului.

Prin urmare, punctul principal de plecare al acestui proiect este reprezentat de preluarea modelelor 3D și a elementelor ce țin de GUI din *Veloren* [29], un action-role playing game realizat în limbajul de programare Rust cu un game engine/framework construit în cadrul companiei Picroma [3], ce presupune explorarea unei lumi voxel-based alcătuită din mai multe biomuri, prin prisma unui personaj aparținând uneia dintre clasele: Om, Orc, Pitic, Elf și Danari.

## **Obiective**

Obiectivele stabilite referitoare la gameplay constau în întâmpinarea utilizatorului cu o scenă de Menu ce îi va permite să modifice o setare a jocului, volumul sunetului, prin intermediul unui slider, să înceapă jocul efectiv și să poată ieși din joc. Odată ce utilizatorul a intrat în jocul efectiv, acesta își poate alege tipul de jucător (Knight, Archer, Mage), determinat de arma folosită, genul (Male / Female), cât și rasa (Human, Orc, Dwarf, Elf și Danari). De asemenea, pentru a evita coliziunea de identitate, fiecare jucător își poate seta un username. Odată creată această instanță, utilizatorul va fi redirecționat spre un Offline Scene tipic jocurilor multiplayer în care va putea alege dacă să se conecteze la un server prin intermediul unui IP, fiind Client sau să facă Host la joc – server-ul fiind ținut pe mașina locală în acest caz. Spawn-area instanței în joc va fi realizată cu o întârziere de câteva secunde prin intermediul unui Network Manager. Player-ul va avea posibilitatea de explorare a mediului și eliminare a inamicilor pentru avansarea în nivel. Totodată, în contextul conectării mai multor persoane, acestea pot comunica între ele prin intermediul unui chat.

# **SUPORT TEHNIC ȘI ABORDĂRI EXISTENTE**

## **2.1 Tehnologiile folosite**

Având ca reper jocul menționat anterior, proiectul pe care l-am dezvoltat este realizat in game engine-ul Unity, folosind C#.

Unity, unul dintre cele mai cunoscute și folosite game engine-uri, permite utilizatorului dezvoltarea jocurilor în maniera interacționării în mod direct cu obiectele, crearea sau importarea animațiilor, sau chiar a unui sistem complex de animații. Acesta are suport cross-platform astfel că jocul realizat poate fi exportat pentru Android, întrucât îndeplinește criteriile ce țin de UI responsiveness. Totodată, Unity integrează diverse third-party plug-ins precum Mirror Networking [14], ParrelSync [17] – pentru testarea sincronizării, Performance Profiler [18]– care va fi utilizat atât pentru dezvoltare cât și pentru analizarea performanței și MagicaVoxel [11] – pentru crearea și editarea modelelor 3D.

Unul dintre primele dezavantaje pe care le-am observat la acest framework a fost faptul că, în momentul creării unui proiect nou, se generează: fișiere meta, deoarece Unity asociază fiecare asset utilizat cu un ID unic pentru a păstra referințe în momentul în care vor fi folosite, biblioteci ce se ocupă cu partea de randare, iluminare, etc., setările proiectului și ale utilizatorului, fișiere ce se ocupă cu managementul dependințelor – toate acestea ocupă aproximativ 200 MB de memorie. Dimensiunea finală a acestui joc ar fi fost 1.6 GB. Bineînțeles că spațiul ocupat în memorie reprezintă un aspect care trebuie luat în considerare astfel că am folosit Managed code stripping [12] pentru eliminarea codului assmebly generat de script-urile în C#, de pachete, plugins și .NET Framework. Acest lucru a fost posibil prin intermediul linker-ului Unity ce face o analiză statică asupra claselor și funcțiilor care nu sunt folosite la execuție. Cu această modificare, s-a reușit reducerea dimensiunii proiectului de la 1.6 GB la 1.4 GB.

Toate modelele, prefabricatele, sprite-urile, texturile și fonturile au fost stocate în folder-ul Resources [25], care în Unity capătă o valență specială. Se consideră o bună practică păstrarea tuturor resurselor în acest director deoarece, la build, Unity va include doar obiectele care chiar sunt utilizate în proiect.

Pentru partea de networking am folosit versiunea 40.0.9 de la Mirror Networking, ce presupune adăugarea în scena Offline (în momentul în care utilizatorul și-a ales deja jucătorul cu care să exploreze) un Game Object, denumit sugestiv Network Manager ce va conține componenta de KCP (Keyset Call Processing) Transport [19]– care este un protocol rapid care poate obține efectul de transmisie a datelor cu o reducere a latenței de până la 40% față de TCP, costul de lățime de bandă fiind cu 10-20% mai redus. Pentru integrarea efectivă, s-a adăugat și un HUD custom ce solicită utilizatorului tipul de conectare la rețea: Host / Client – IP Address / Server Only.

Fără un server deployment, ar fi imposibil de testat feature-urile ce țin de sincronizare. Tocmai din acest motiv, ParrelSync este o extensie ce permite utilizatorilor să testeze jocul fără a face build la fiecare schimbare. Practic, ce se întâmplă este că, se clonează folderul proiectului de fiecare dată când apar modificări. Astfel, dacă proiectul inițial este MyUnityProject, va face automat un folder MyUnityProject\_clone0. Aceste clone se suprascriu de fiecare dată când este accesat ParrelSync.

Performance Profiler-ul va fi utilizat pentru diagnosticarea problemelor ce țin de cât de mult timp este folosit pe CPU pentru fiecare task, procesul de randare și folosirea GPU-ului, mecanicile jocului, networking și audio source. Desigur că analiza obținută va fi în concordanță cu specificațiile mașinii pe care este executat jocul, acestea fiind:

* CPU: Intel ® Core ™ i7-10750H @ 2.60 GHz
* GPU: NVIDIA GeForce GTX 1650

Pentru crearea modelelor ce compun decorul cât și pentru editarea eroilor acestui joc și a GUI-ului, s-a folosit MagicaVoxel, un editor special dezvoltat pentru crearea de obiecte voxel / pixel art. Acesta cuprinde și o secțiune care presupune aplicarea unui algoritm de Marching Cubes pentru redarea modelelor în stilul low poly. Din păcate, acest efect de netezire este aplicat peste un model sub forma unui .obj doar local, modificând randarea în cadrul editorului. MagicaVoxel nu poate exporta modelul creat împreună cu informația de rendering în stil Marching Cubes.

## **2.2 Implementări similare**

Principala sursă de inspirație a acestui joc este, după cum a fost menționat deja, RPG-ul Veloren. Având în vedere faptul că Veloren este un proiect open-source, toate resursele acestuia se pot găsi pe GitLab [1], oricine având posibilitatea de a contribui, de a realiza mod-uri și de a aduce îmbunătățiri. În momentul creării personajului, echipa care a dezvoltat acest joc a optat pentru schimbarea în mod dinamic a eroului, utilizatorul personalizându-l prin diverse accesorii, stilul părului, detalii ale ochilor, culoarea pielii.

Totodată, acest joc dispune de două lumi izolate pentru explorare – Singleplayer și Multiplayer. Lumea ce urmează a fi explorată este generată procedural. Totuși, pentru evitarea unor lungi perioade de timp cauzate de acest proces, utilizatorii își pot încărca o hartă pre-generată prin intermediul unui seed (ex: world\_seed = 40382), pe care îl pot seta din cod [22]. Toate modelele 3D ale acestui joc au fost realizate cu ajutorul MagicaVoxel, ce dispune de o interfață grafică destul de simplistă, fiind ușor de utilizat. Utilizatorul este întâmpinat de NPCs (Non-Player Character) de la care poate cumpăra diverse materiale, mâncare, arme. Atmosfera este completată de un bioritm caracterizat printr-o alternanță de 2 ore.

GUI-ul conține un inventar de tip hot-bar (din care jucătorul își poate lua elemente în mod rapid prin apăsarea unei taste de la 0 la 9, în funcție de cum sunt așezate), care este linkat la inventarul de bază al jucătorului, un mini-map pentru localizarea în spațiu și depistarea „obiectivelor turistice” sau a altor jucători, indicatoare de progres și un chat.

Pe lângă faptul că armele și componentele armurii se pot găsi la NPC, comercianți sau drept recompense în urma avansării în level, utilizatorul își poate construi propriile obiecte dacă dispune de resursele necesare, prin intermediul unui sistem complex de crafting.

De asemenea, la fel ca în alte jocuri ce se încadrează în aceeași tematică, precum bine-cunoscutul Minecraft [13], realizat în Java sau Cube World [7] (C++), se remarcă existența unor dungeons ce au rolul de a provoca jucătorul la o serie de atacuri pregătite de niște mobi avansați, finalitatea fiind obținerea unor cufere de tip mystery box.

Modelul după care a fost programat controller-ul personajului este cel din Minecraft. Totodată, am preluat și stilul chat-ului, respectiv font-ul lor consacrat [9].

Cube World, precursorul și principalul reper folosit de cei de la Veloren, a fost dezvoltat în aceeași manieră ca cea prezentată anterior, având în plus un sistem de paragliding complex.

De remarcat este faptul că Rust este din punct de vedere sintactic asemănător cu C++. Totuși, acesta prezintă performanțe de viteză mai bune și o securitate îmbunătățită a memoriei prin faptul că nu permite folosirea pointer-ilor nuli sau dangling (care nu pointează spre un obiect valid), ceea ce face ca Veloren să fie superior față de sursa sa de inspirație, Cube World.

# **IMPLEMENTARE ȘI MULTIMEDIA**

## **3.1 Elemente de GUI, design și audio**

Sfera de Multimedia a fost acoperită prin intermediul elementelor ce țin de UI, acestea fiind randate cu ajutorul unui Canvas, care in Unity reprezintă un Game Object atașat scenei. În esență, acesta reprezintă un quad în scena 3D pe care se vor atașa în continuare elemente de GUI precum butoane, logo, background – reprezentate sub formă de “copii” în ierarhia de obiecte a scenei. De menționat este faptul că, în Unity, primul copil din ierarhie este desenat primul pe ecran, o regulă importantă pentru evitarea overlap-ului.

Canvas-ul conține o componentă de randare care poate fi setat pe spațiu ecran (screen space) sau spațiu lume (world space). Pentru acest joc, am ales opțiunea screen space - overlay deoarece, Canvas-ul se va adapta automat la eventualele schimbări ale dimensiunii sau rezoluției ecranului. Utilizarea modului world space este relevant în contextul în care UI-ul ar fi un obiect al scenei 3D (cum ar fi o bară de viață deasupra capului pentru jucător). Totodată, am adăugat și o componentă de Graphic Raycaster.

A screenshot of a video game

Description automatically generatedCum funcționează raycast-ul în Unity? [28] Practic, este o funcție care proiectează o rază “prin” scenă, returnând o valoare booleană dacă target-ul a fost lovit. În acel moment, se vor extrage informații referitoare la distanța, poziția și componenta Transform, utile pentru detectarea apăsării butoanelor (click buton Play -> Raycast pentru (x, y) click -> găsit buton -> TRUE -> executare Play).

Figura 1 - Scena de Main Menu

Ce este componenta Transform în Unity? [4] Această componentă este folosită pentru păstrarea poziției, rotației și scalării. Toate Game Object-urile dispun de câte o componentă Transform deoarece se află undeva în spațiu la un anumit moment dat.

A picture containing shape

Description automatically generatedAstfel, pentru pagina de Menu, au fost adăugate pe Canvas un Background sub formă de imagine, un logo sub formă de sprite, trei butoane pentru Play, Settings și Quit. În momentul accesării Setărilor, se va activa în ierarhie copilul care permite vizualizarea slider-ului de volum pentru sunete.

Figura 2 - Setarea volumului

Sunetul. O altă componentă multimedia, a fost posibilă prin intermediul unui AudioManager [2], la care am atașat un script ce asigură actualizarea volumului și redarea unei melodii specifice în funcție de scena la care se află utilizatorul. Totodată, au mai fost adăugate sunete de tip “blip” în momentul în care se face hover cu mouse-ul pe butoane.

A screenshot of a video game

Description automatically generatedOdată ce a fost apăsat butonul de Play, jucătorul va fi pus în situația de a și alege eroul cu care se va juca. Această scenă conține un 3D Visualizer pentru previzualizarea eroului, realizat prin modificarea componentei Y din Transform-ul personajului, un Input Text Field, prin intermediul căruia se poate seta un nickname, schimbare care se va propaga și pe rețea.

Figura 3 - Alegerea eroului

Quad-ul pentru alegerea jucătorului a fost gândit după 3 aspecte: gen, rasă și arma folosită la luptă. Astfel, utilizatorul va avea de ales între următoarele: Male/Female, Human/Orc/Dwarf/Elf/Danari și Knight/Archer/Mage, schimându-se doar capul și arma jucătorului. Această funcționalitate a fost implementată cu ajutorul noțiunii de Scriptable Object[26], astfel că toate armele și capetele clasificărilor menționate anterior au fost păstrate într-un vector. În momentul accesării inițiale a scenei, jucătorul default este Male - Human – Knight, având atașat script-ul CharacterPreview, ce presupune starea de Idle a acestuia (animație de respirat). Scriptable Object în Unity este un container de date independent, folosit pentru reducerea memoriei proiectului evitând copierea prin valoare. Implementarea naivă ar fi constat în instanțierea fiecărui head prefab accesat si distrugerea (dezalocarea) acestuia de fiecare dată când utilizatorul ar fi solicitat un anumit erou, metodă care poate poate genera bug-uri și probleme de performanță.

Metoda implementată de mine constă în păstrarea prefabricatelor (capete și arme) în două containere separate și accesarea acestora prin referință. Prin urmare, va exista o singură copie a elementelor în memorie.

Totodată, această scenă conține pe Canvas două butoane Back și Create, ce determină tranziția la scena anterioară, respectiv la scena de joc și un Input Text Field pentru alegerea unui nume. Acest nume este salvat în PlayerPrefs, care în Unity, reprezintă o clasă ce se ocupă cu păstrarea tuturor setărilor făcute de utilizator între sesiunile de joc.

Vizualizarea 3D a personajului și randarea în același timp a eroului a fost posibilă prin intermediul situării componentei Transform a imaginii de fundal în spatele personajului. (Zbackground < Zpersonaj).

Multimedia mai înseamnă și text, astfel că pentru redarea acestuia am folosit font-ul original Minecraft la care am adăugat un Text Mesh pentru a obține un outline de culoare gri. În acest fel, se asigură lizibilitatea textului, în momentul în care acesta se suprapune cu alte obiecte de aceeași culoare.

Un aspect interesant pe care l-am descoperit prin prisma unui bug vizual a fost apariția fenomenului de Z-fighting [30], fenomen des întâlnit la jocurile cu perspectivă 3D. Acesta apare în momentul randării în contextul situării primitivelor la o distanță aproximativ egală față de cameră. Prin urmare, acestea vor avea valori aproape egale în Z-buffer – care se ocupă cu păstrarea informațiilor ce țin de reprezentarea unui obiect ca fiind mai apropiat sau mai depărtat de frustrum (depth information). În pipeline-ul de randare Z-buffer-ul este o matrice ce păstrează valoarea Z a fiecărui pixel. Dacă intervine un alt obiect care trebuie să fie randat la aceeași poziție, algoritmul de Z-buffer suprascrie valorile anterioare, acordându-i prioritate în fața camerei. La Z-fighting pixelii care sunt afectați provin de fapt din aproximarea pe care o face algoritmul între cele două obiecte, în urma rasterizării triunghiurilor.

Trecând mai departe, peste scena Offline Scene, în urma căreia se va crea instanța Network Manager, ce va conține prefabricata personajului deja creat, cât și prefabricatele mob-ilor ce urmează a fi generați, scena Level\_01 reprezintă scena efectivă de explorare a mediului. HUD-ul scenei de joc conține Health Bar, Mana Bar (viața și puterea jucătorului ce se regenerează treptat) Experience Bar (progresul în ceea ce privește învingerea inamicilor), un sistem de chat și un inventar a cărei funcționalitate nu am mai avut timp să o implementez.

Viața și puterea jucătorului au fost reprezentate ca niște elemente pe Canvas, compuse din dintr-un background gri sub formă de dreptunghi, un border și un slider ce își modifică lungimea în funcție vătămarea personajului. Acest slider revine la dimensiunea sa inițială treptat, simulând procesul de vindecare / regenerare.

Sistemul de chat a fost construit din două componente, cea de Input Field Text care se activează și devine disponibilă în momentul în care se apasă tasta “T” și un Chat Panel, ce are rolul de a afișa istoricul mesajelor. Pentru o evidență clară a contextului acestui istoric, fiecare mesaj urmează pattern-ul: [nume\_jucător]: <mesaj>

Un alt detaliu pe care l-am adăugat a fost crosshair-ul, asociat cu vederea telescopică, pentru a oferi o mai bună apreciere în momentul țintirii inamicului. Acesta este reprezentat printr-o imagine cu background transparent, poziționat în centrul Canvas-ului, având o animație de fixare a țintei – micșorarea crosshair-ului.

Pentru a atenționa jucătorul că este vătămat grav în timpul unei lupte, am decis să atașez un border strident, de culoare roșie, realizat în paint.net, la care am modificat componenta de alpha transparency pentru a reda un efect mai dramatic. Astfel, de fiecare dată când valoarea corespunzătoare vieții jucătorului scade sub un anumit prag, acest border se va activa pe Canvas, urmând să se dezactiveze odată cu vindecarea eroului.

A video game showing a video game

Description automatically generated with low confidence

Figura 4 - Activarea unui border de atenționare asupra nivelului scăzut de viață

O carateristică implementată care nu se observă chiar la prima vedere, deși se află în strânsă legătură cu partea de interfață, ar fi prezența unui Easter Egg [8]. Astfel, în momentul în care utilizatorul dă click pe pentagrama desenată pe cartea din logo, se vor declanșa niște artificii pentru o scurtă perioadă de timp. Această funcționalitate are rolul de a surprinde în mod plăcut utilizatorul, determinându-l să descopere și mecanici ascunse.

## **3.2 Networking și Multiplayer**

Desigur că, în alte circumstanțe, propagarea unui nume de utilizator în rețea ar fi fost un proces ce presupune utilizarea corespunzătoare a unor sockets cu protocolul TCP pentru evitarea pierderii datelor. Folosind framework-ul Mirror Networking, acest proces este realizat cu un SyncVar cu hook-ul corespunzător funcției de schimbare a numelui. Aceste SyncVars sunt folosite în cadrul comunicării Client->Server, funcționlitate redată de Mirror prin tag-ul [Command]. Alte tag-uri Mirror folosite pentru sincronizare ar mai fi ClientRPC sau TargetRPC pentru comunicarea Server-Client, apelat de pe server pentru a transmite date *tuturor* clienților. Acesta este întâlnit adesea în momentul sincronizării mișcărilor jucătorului.

Cum funcționează de fapt protocolul RPC? [20] Aplicațiile de apelare RPC funcționează sincron, astfel că serverul lansează o cerere sub forma unui apel de metodă. Serviciul de rețea este identificat printr-un port unde există un proces care rulează pe fundal și așteaptă cereri de conectare. În continuare, RPC traduce aceste apeluri și le trimite spre destinația setată. Această destinație prelucrează metoda trimisă și lista de argumente și va trimite înapoi un răspuns.

Text

Description automatically generatedToate preferințele utilizatorului ce țin de aspectul eroului au fost salvate cu ajutorul unor SyncVars la care au fost atașate hook-urile corespunzătoare metodelor de tip [Command] ce transmit informația de la Client la Server.

Figura 5 - Folosirea tag-urilor de Mirror Networking pentru sincronizare

MonoBehaviour reprezintă clasa de bază din care derivă toate clasele din Unity. Totuși, pentru a semnala faptul că respectivele funcționalități comunică cu serverul, Mirror impune derivarea din clasa NetworkBehaviour. Astfel, sincronizarea mișcării jucătorului s-a realizat exclusiv prin comunicare de la Client la Server, pentru a înregistra la fiecare frame poziția curentă după care, în sens invers, de la Server către toți ceilalți clienți cu ajutorul RPC-ului. Această implementare abstractizează procesul de serializare sub formă de biți și deserializare realizat în programarea low level pentru networking.

Spawnarea în joc a eroului cât și a inamicilor se face la aceeși poziție inițială de fiecare dată, prin prisma NetworkManager-ului în care am reținut un array cu prefabricate pentru corpurile acestora, cât și pentru efectele de atac ale fiecăruia (fireballs, sword trail, arrows). Generarea de obiecte în scenă s-a realizat cu funcția Instantiate pe Server. Totodată, afectarea nivelului de viață a jucătorului s-a realizat printr-un callback pe server care apelează funcția DealDamage ce aplică o decrementare la nivelul de viață actual. Același principiu este aplicat și pentru nivelul de putere al eroului.

O altă componentă ce se încadrează în această categorie ar fi sistemul de chat, pe care l-am implementat în clasele ChatManager și ChatSync. Clasa ChatManager se ocupă în principiu cu preluarea string-ului introdus de utilizator în InputField, ștergerea acestuia în momentul apăsării Enter. După preluarea string-ului, acesta este adăugat intr-o coadă, ce are o capacitate maximă de 10. În momentul în care s-a atins capacitatea maximă a acestei structuri, se vor scoate din coadă mesajele deja afișate în Chat Panel si se va face update la UI, mai exact, la Scroll View. Pentru această implementare am tratat și cazurile în care utilizatorul ar încerca să trimită un mesaj null. Clasa ChatSync se ocupă cu primirea mesajului pe server de la utilizator și trimiterea acestuia către celelalte instanțe conectate.

# **INTERACȚIUNEA CU UTILIZATORUL**

## **4.1 Mecanica jocului**

Controlul eroului se bazează pe implementarea unui sistem complex ce respectă principiile fizicii, luându-se în calcul coliziunile player-player, player-inamic, arme-inamic și player-decor. Astfel, eroul ales poate fi controlat cu ajutorul tastelor W/A/S/D, Space (pentru a sări) și Shift-W/S pentru a alerga.

Prin urmare, pentru deplasare, s-a utilizat un Quaternion [21], care în Unity reprezintă o structură de date bazată pe numere complexe, folosită pentru reprezentarea rotațiilor. În momentul apăsării uneia dintre tastele WASD, se va calcula un quaternion ce va lua ca parametru (cu ajutorul metodei Euler), vectorul direcție, dat de cif.GetLookDirection(). Mai apoi, această rotație se va pasa către metoda MovePosition [15] ce va determina mișcarea efectivă. Pentru a alerga, se va adăuga la amplitudinea acestei operații un shiftMultiplier. Partea interesantă intervine în momentul în care vrem să simulăm efectul de salt. Acest lucru s-a realizat folosind un Raycast care are centrul de proiecție egal cu centrul cutiei de coliziuni a jucătorului, destinația fiind planul pe care acesta stă. Deci, dacă raza realizează o coliziune cu pământul, atunci jucătorul este la sol, în caz contrar, acesta este desprins de la sol. Astfel, în momentul apăsării tastei Space, se va verifica dacă jucătorul este la sol și în caz afirmativ, i se va aplica o forță perpedinculară pe axa Oy (Vector3.up) simulând acel impuls.

Jocurile bine realizate conțin anumite detalii care probabil rămân neobservate de către mulți, dar care întregesc atmosfera redată. Amuzant este că absența acestora este imediat observată. Un exemplu în acest sens ar fi animația de respirat a eroului ce compune starea de Idle. Astfel, chiar din scena de alegere a eroului, această animație poate fi sesizată. Aceasta a fost implementată folosind o sinusoidă cu unghiul float angle = Mathf.Sin(t \* 2f). Acest unghi se va pasa în continuare la poziția locală pentru toate componentele din care este format jucătorul, mai puțin la membrele inferioare – acestea sunt statice în momentul respirării. La fiecare frame, se va adăuga o constantă, delta la timpul t.

Atacul este implementat folosind aceeași mecanică pentru toate tipurile de arme. Practic, această funcționalitate s-a realizat tot cu o sinusoidă aplicată de această dată pe axa de rotație Ox. Player-ul își va ridica brațul în care ține arma până la nivelul umerilor și inapoi. Pentru a ataca, utilizatorul se va folosi de click stânga. Tot la acest capitol, feedback-ul pe care îl dă atacul realizat atat de erou cât și de inamic constă în declanșarea unui sistem de particule personalizat. Atacul este tot un GameObject, ce conține drept componente: o sferă de coliziune, un RigidBody, pentru a interacționa cu gravitația jocului, un NetworkIdentity și un NetworkTransform pentru a exista în contextul multiplayer și un script dezvoltat de mine, Projectile. Acest script instanțiază un GameObject de acest tip, aplicându-i o forță pe direcția Oz, cu transform.forward. Pe acest GameObject se invocă funcția Destroy pentru a dispărea din scenă. Pentru verificarea interacțiunii cu ceilalți jucători / mobi, s-a folosit un LayerMask [10], care ajută la detectarea cutiei de coliziuni.

O altă caracteristică importantă care a fost implementată este controller-ul camerei de tip Third Person, ce dispune de următoarele funcționalități: zoom in/out asupra target-ului, orbitarea și rotirea în jurul target-ului și apropierea camerei de jucător în momentul în care, spre exemplu, acesta se află cu spatele foarte aproape de un GameObject ce dispune de o cutie de coliziune. Astfel, printr-un RaycastHit [23], se preiau informații despre coliziunea dintre raza ce unește centrul camerei cu player-ul și peretele respectiv, aplicându-se un SmoothDamp [27] care se ocupă cu deplasarea ușoară a camerei înspre player.

**A screenshot of a video game

Description automatically generated**Tipurile de coliziuni folosite pentru acest joc sunt de tip: sferă, capsulă, paralelipiped și prismă. Pentru iluminarea scenei s-au folosit surse de lumină direcțională care imită razele soarelui.

Figura 6 - Scena de joc

## **4.2 Decor**

Toate modelele 3D ce compun decorul jocului au fost împrumutate din arhiva open source Veloren, în format .vox, dezvoltatorii jocului aplicând un algoritm de optimizare a acestora în Rust, pentru reducerea numărului de triunghiuri. Deoarece Unity nu recunoaște acest tip de fișier am fost nevoită să convertesc toate resursele .vox în fișiere: .mtl, .obj și .png – ce compun contextul final descris de: textură, UV map, culoare, geometrie, cu ajutorul programului MagicaVoxel. Vom afla în cele ce urmează faptul că nu a fost o idee prea bună, deși, pe moment, rezultatul a fost unul satisfăcător.

Ca o aprofundare, .mtl este un tip de fișier care reține informații despre setările materialului folosite de modelul 3D – aceste setări sunt proprii editorului în care modelul este creat/exportat. Totodată, referința acestuia este păstrată într-un fișier .mtl.meta care face legătura dintre .mtl și .obj, “explicându-i” editorului în care acesta este randat cum anume trebuie să aplice materialul/textura pe obiectul 3D. De cele mai multe ori, aceste fișiere sunt însoțite și de o imagine, un .png

Cum arată acest .mtl? Am atașat mai jos un exemplu de conținut .mtl pentru arcul cu săgeți. Ne putem da seama că lucrurile încep să prindă mai mult sens întrucât tot ce face acest tip de fișier este să păstreze următoarele componente aplicate pe suprafața și unghiurile modelului [5]: Ka (ambient color), care redă informații despre cum se reflectă lumina pe suprafața obiectului, Kd (diffuse color), ce contribuie la aplicarea culorii pe obiect, Ks (specular color) care aplică efectul de strălucire și imaginea atașată, care se ocupă de detaliile de design ale arcului. În continuare, fișierul .obj va referi către cel .mtl prin intermediul unei declarații de tip *mtllib*. Mai multe informații interesante despre acest subiect se pot găsi aici: [6].

newmtl palette

illum 1

Ka 0.050 0.100 0.030

Kd 0.620 0.200 1.000

Ks 0.500 0.500 0.500

map\_Kd bow.png

Totuși, un aspect destul de important pe care l-am omis din vedere realizând această conversie ar fi corelarea numărului de triunghiuri pe fiecare obiect în parte.

Terenul, cât și munții din joc au fost creați prin intermediul unui tool high-level din Unity- Paint Brushes – ce pune la dispoziție utilizatorului o gamă largă de texturi pe care le poate folosi. Skybox-ul (cerul cu nori mișcători) a fost implementat cu ajutorul unui cub peste care a fost atașat un material. Pentru a vizualiza cum funcționează acest concept, ne putem închipui că întreaga scenă a jocului se află în interiorul acestui cub peste care a fost aplicată o forță pentru a se roti, creând impresia de nori mișcați de vânt.

Moartea jucătorului are ca efect trecerea la scena de End Of Life care are atașată pe Canvas o imagine dramatică și un buton ce permite reînceperea altei sesiuni de joc.

A screenshot of a video game

Description automatically generated

Figura 7 - Scena de final de joc

## **4.3 Inamici AI**

În momentul în care player-ul se apropie de un inamic îndeajuns de mult încât să intre în zona acestuia de acțiune, mob-ul respectiv își va seta ca target eroul și va genera sfere de foc. Inamicul va urmări jucătorul și va încerca să îl atace până în momentul în care acesta va ieși din perimetrul setat ca trigger. Totodată, inamicul nu este static, acesta se mișcă după niște coordonate alese aleatoriu, perimetrul fiind setat cu ajutorul unui NavMesh [30]. NavMesh este o funcționalitate din Unity care preia date despre Render Meshes și Teren și permite utilizatorului să aleagă o zonă de pe hartă care să fie destinată patrulării unui GameObject. Inamicul dispune la rândul său de un LayerMask care înregistrează dacă acesta a fost lovit, pentru scăderea punctelor reprezentând viața. Spre deosebire de eroul principal, inamicul nu dispune de calitatea de regenerare automată a vieții. Astfel, la fiecare frame, în funcția Update, se vor obține informații despre distanța de la inamic la player și se va analiza dacă aceasta este mai mică decât o valoare setată de mine. În caz afirmativ, inamicul își va orienta privirea spre target, prin intermdiul funcției LookAt(targetPostition) și va începe să atace.

Atacul acestora este sincronizat pe rețea prin intermediul tag-urilor de tip [Command], care trimit callback-uri de la Client la Server. Odată ce server-ul a înregistrat atacul, va genera sferele de foc ce au atașate cutii de coliziuni, cu ajutorul funcției Network.Spawn(fireball).

# **CONCLUZII**

Toate obiectivele ce țin de Game Design-ul jocului au fost implementate. Rezultatul este o clonă după jocul Veloren, realizată în alt mediu de programare. Este această clonă mai eficientă, mai rapidă? Ar fi putut fi măcar? Răspunsul la aceste întrebări îl putem afla prin analizarea rezultatelor obținute prin intermediul unui Performance Profiler.

În momentul pornirii jocului, mai exact în scena de Menu, se poate observa faptul că CPU-ul nu este afectat prea mult, momentan. Era de așteptat având în vedere faptul că scena de Menu conține doar un Canvas pe care sunt atașate niște elemente de UI. Mai jos, putem observa faptul că un frame în acest context durează aproximativ 16 ms. O problemă pe care o putem semnala aici, totuși, ar fi solicitarea procesorului prin GarbageCollector. GarbageCollector-ul se ocupă cu dezalocarea memoriei pe heap. În Unity, alocările pe heap sunt cele realizate prin referință.

Graphical user interface

Description automatically generated with medium confidence

Figura 8 - Analiză performanță scenă Main Menu

Astfel, posibile îmbunătățiri ar fi: reducerea numărului de alocări pe heap și a creării obiectelor prin referință, evitarea alocării în metodele care sunt apelate des sau folosirea metodei Caching, ce se referă la păstrarea unei refeințe către un anumit tip de date stocate într-un array petrnu a face o singură alocare pe heap pentru toate acele obiecte. (exemplu: FindObjectsOfType<Renderer>(); ). De menționat este faptul că acest lucru ar fi destul de dificil de implementat în contextul instanțelor networked.

Rezultatul dramatic intervine în momentul scenei efective de joc, astfel, că, după cum se poate observa și din imaginea de mai jos, intervine un spike, provocat de momentul în care jucătorul ridică privirea “din pământ”, uitându-se înspre decorul din scenă (case, tarabe, flori, cutii) ce este compus dintr-o multitudine de triunghiuri. În mod normal, o față a unei cutii din joc ar fi trebuit să fie compusă, ideal, din două triunghiuri. Acest lucru nu se întâmplă deoarece, în momentul exportării modelelor 3D .vox în .obj + .mtl + .png, MagicaVoxel nu aplică algoritmul de decimare a triunghiurilor, rezultând o dispunere haotică și ineficientă a triunghiurilor.

Chart

Description automatically generated

Figura 9 - Analiză performanță scenă de joc

O posibilă îmbunătățire în acest sens ar fi fost preluarea în mod direct a unor modele în formatul suportat de Unity. Însă, cum acest aspect a fost descoperit destul de târziu, ar fi fost destul de complicat regândirea unui alt Game Design. Prin urmare, se observă solicitarea intensivă a CPU-ului din cauza Rendering-ului. Mai jos, am atașat o imagine cu vizualizarea dispunerii triunghiurilor în cadrul unei tarabe.

A screenshot of a video game

Description automatically generated

Figura 10 - Dispunerea haotică a triunghiurilor

Particularizând toate aceste aspecte, se poate afirma faptul că realizarea unui joc 3D multiplayer folosind Unity ar putea fi la îndemâna oricărui programator începător, care are noțiunile de bază în ceea ce privește Programarea Orientată pe Obiecte, Networking și Grafică pe calculator. Pentru ca acest joc să îndeplinească majoritatea cerințelor ce țin de performanță, recomandarea ar fi evitarea unui framework pre-built. Desigur că procesul construirii unui game engine este unul mai anevoios, însă totodată, este și mai sigur și mai puțin predispus la diverse probleme.

1. **REFERINȚE BIBLIOGRAFICE**

|  |  |
| --- | --- |
| [1] | "Arhiva Veloren de pe Gitlab" https://gitlab.com/veloren/veloren. |
| [2] | "Audio Manager in Unity" https://docs.unity3d.com/2018.1/Documentation/Manual/class-AudioManager.html. |
| [3] | "Compania Picroma" https://picroma.com/. |
| [4] | "Componenta Transform in Unity" https://docs.unity3d.com/Manual/class-Transform.html. |
| [5] | "Componentele luminii in jocuri" https://math.hws.edu/graphicsbook/c4/s1.html. |
| [6] | "Continutul unui fisier .mtl" http://paulbourke.net/dataformats/mtl/. |
| [7] | "Cube World" https://en.wikipedia.org/wiki/Cube\_World. |
| [8] | "Easter Egg" https://en.wikipedia.org/wiki/Easter\_egg\_(media). |
| [9] | "Font-ul folosit pentru chat-ul din Minecraft" https://www.dafont.com/minecraft.font. |
| [10] | "LayerMask" https://docs.unity3d.com/ScriptReference/LayerMask.html. |
| [11] | "Magica Voxel" https://www.voxelmade.com/magicavoxel/. |
| [12] | "Managed code stripping" https://docs.unity3d.com/Manual/ManagedCodeStripping.html. |
| [13] | "Minecraft" https://www.minecraft.net/en-us. |
| [14] | "Mirror Networking" https://mirror-networking.com/. |
| [15] | "Mutarea unui obiect in Unity" https://docs.unity3d.com/ScriptReference/Rigidbody.MovePosition.html. |
| [16] | "NavMesh" https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html. |
| [17] | "ParrelSync" https://github.com/VeriorPies/ParrelSync. |
| [18] | "Performance Profiler" https://docs.unity3d.com/Manual/Profiler.html. |
| [19] | "Protocolul KCP" https://mirror-networking.gitbook.io/docs/transports/kcp-transport. |
| [20] | "Protocolul RPC" https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC. |
| [21] | "Quaternion" https://docs.unity3d.com/ScriptReference/Quaternion.html. |
| [22] | "Randarea lumii folosind un seed" https://book.veloren.net/players/world-generation.html. |
| [23] | "RaycastHit" https://docs.unity3d.com/ScriptReference/RaycastHit.html. |
| [24] | "Render Pipelines" https://docs.unity3d.com/Manual/render-pipelines.html. |
| [25] | "Resources folder in Unity" https://blog.theknightsofunity.com/unity-resources-folder-how-to-use-it/. |
| [26] | "Scriptable Objects în Unity" https://docs.unity3d.com/Manual/class-ScriptableObject.html. |
| [27] | "SmoothDamp" https://docs.unity3d.com/ScriptReference/Vector3.SmoothDamp.html. |
| [28] | "Unity Raycast" https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-GraphicRaycaster.html. |
| [29] | "Veloren" https://veloren.net/. |
| [30] | "Z-fighting" https://viscircle.de/guide-for-beginners-what-is-z-fighting/?lang=en. |