IMPLEMENTAREA UNUI JOC MULTIPLAYER ÎN UNITY

# Contents

[Contents 1](#_Toc138321496)

[I. Introducere 5](#_Toc138321497)

[1.1. Importanța temei 5](#_Toc138321498)

[1.2. Obiectivele lucrării 6](#_Toc138321499)

[1.3. Metodologia de cercetare 7](#_Toc138321500)

[1.4. Rezultatele obținute 9](#_Toc138321501)

[1.5. Rezumatul lucrării pe capitole 9](#_Toc138321502)

[II. Jocuri Multiplayer 10](#_Toc138321503)

[1. Istoric 10](#_Toc138321504)

[2. Tipuri de jocuri 11](#_Toc138321505)

[3. Tendințe 12](#_Toc138321506)

[4. Studiu comparativ privind două dintre cele mai populare jocuri multiplayer 13](#_Toc138321507)

[III. Game engines 14](#_Toc138321508)

[1. Definire generală 14](#_Toc138321509)

[2. Soluții existente 14](#_Toc138321510)

[3.1. Unity 14](#_Toc138321511)

[3.2. Unreal Engine 15](#_Toc138321512)

[3.3. Godot 16](#_Toc138321513)

[3.4. Tabel de comparație 16](#_Toc138321514)

[Tehnologii Folosite 17](#_Toc138321515)

[1. .NET 17](#_Toc138321516)

[1.1. Limbajul C# 18](#_Toc138321517)

[1.2. Arhitectura .NET 18](#_Toc138321518)

[2. Visual Studio 19](#_Toc138321519)

[3. Blender 20](#_Toc138321520)

[4. Unity 20](#_Toc138321521)

[Unity 21](#_Toc138321522)

[1. Viziunea 21](#_Toc138321523)

[2. Arhitectura modulară bazată pe componente 21](#_Toc138321524)

[3. Unity Scripting API 22](#_Toc138321525)

[4. Sistemul de simulare fizică 24](#_Toc138321526)

[5. Sistemul de randare 25](#_Toc138321527)

[6. Sistemul de animații 26](#_Toc138321528)

[7. Sistemul de editare e terenului 27](#_Toc138321529)

[8. Scriptable Objects 29](#_Toc138321530)

[IV. Biblioteca Photon Fusion 29](#_Toc138321531)

[1. Modul ’’Hosted’’ 30](#_Toc138321532)

[2. Componente principale 30](#_Toc138321533)

[3. Bucla de simulare Fusion 31](#_Toc138321534)

[3.1 Tick-uri 31](#_Toc138321535)

[3.2. Predictia 32](#_Toc138321536)

[3.3. Ordinea de execuție 34](#_Toc138321537)

[V. Descrierea implementării 35](#_Toc138321538)

[1. Definirea cerințelor 35](#_Toc138321539)

[2. Prezentarea arhitecturii soluției 37](#_Toc138321540)

[2.1. Prezentare a aplicației integrale 38](#_Toc138321541)

[2.2. Prezentarea aplicației Client și Host 38](#_Toc138321542)

[3. Descrierea modulelor componente 41](#_Toc138321543)

[3.1. Photon Cloud 41](#_Toc138321544)

[3.2. Aplicatia Host și Client 43](#_Toc138321545)

[3.2.1. Creearea interfeței 43](#_Toc138321546)

[3.2.2. Sistemul de meniuri 44](#_Toc138321547)

[3.2.3. Sistemul de UI din timpul jocului 45](#_Toc138321548)

[3.2.4. Caracterul 46](#_Toc138321549)

[3.2.4.1. Sistemul de stări 47](#_Toc138321550)

[3.2.4.2. Punctul central 49](#_Toc138321551)

[3.2.4.3. Sistemul de mișcare 49](#_Toc138321552)

[3.2.4.4. Sistemul de animații 50](#_Toc138321553)

[3.2.4.5. Personalizarea animațiilor 51](#_Toc138321554)

[3.2.4.6. Sistemul de viață 54](#_Toc138321555)

[3.2.4.7. Sistemul de inventariere 54](#_Toc138321556)

[3.2.5. Armele și proiectilele 54](#_Toc138321557)

[3.2.5.1. Sistemul de arme 55](#_Toc138321558)

[3.2.5.2. Sistemul de proiectile 56](#_Toc138321559)

[3.2.6. Stilurile de joc 57](#_Toc138321560)

[3.2.7. Începerea sesiunii de joc 57](#_Toc138321561)

[3.2.8. Obiecte interactive 58](#_Toc138321562)

[3.3. Web API 59](#_Toc138321563)

[3.4. Procesul de autentificare 59](#_Toc138321564)

[3.5. Baza de date 60](#_Toc138321565)

[Rezultatele testelor efectuate și interpretarea acestora 61](#_Toc138321566)

[VI. Concluzii 61](#_Toc138321567)

[Sinteza principalelor idei din lucrare 61](#_Toc138321568)

[Direcții pentru continuarea cercetării 61](#_Toc138321569)

[VII. Bibliografie 61](#_Toc138321570)

[Bibliography 61](#_Toc138321571)

Abrevieri si chestii

Snapshot = captură a stării curente a aplicației, luând in considerare doar obiectele marcate ca facand parte din bucla de simulare Fusion

Punch through = patrundere prin firewall ??

Relay server ??

Pipeline

State machine – masina de stari

Multiplayer ?

Debugger ? - depanator

Bug ? = erori

Tool ? = instrumente, unelte

Cross-paltform =multiplatforma

multiplayer first-person shooter?

Singleplayer ?

High-level ? in contextual: biblioteca ce foeră control high-level

Anti-cheat ?

UI

REST

MVC

HTTPS

JWT

JSON

SQL

CRUD

GNU

NES

MOBA

# Introducere

## Importanța temei

În ziua de astăzi, mediul digital devine un nou teren de antrenament, unde oamenii își dezvoltă abilitățile în diferite domenii: de la capacitatea de a lua decizii sub influența stresului, la realizarea unor sarcini fizice complexe precum o operație chirurgicală sau executarea focului la țintă. Mai mult, prin intermediul experiențelor digitale multiplayer, acest mediu aduce beneficii semnificative și în dezvoltarea abilităților colective. Având posibilitatea să interacționeze cu jucători din întreaga lume, utilizatorii sunt expuși la o gamă variată de perspective, care stimulează dezvoltarea abilităților de comunicare, cooperare și conducere.

În domeniul IT, o aplicație de tip joc este foarte asemănătoare cu una de tip simulator. Ambele reprezintă o aplicație software interactivă în cadrul căruia input-ul oferit de utilizator este tratat în timp real pentru a da contur unei serii de evenimente ce presupun simularea fizicii și sincronizarea cu efecte vizuale sau sonore complexe. Toate acestea sunt obținute prin eforturi intense computaționale și au ca rezultat creșterea calității experienței utilizatorului.

Jocul multiplayer first-person shooter reprezintă un punct de pornire esențial în dezvoltarea unui simulator de succes. Prin crearea unei experiențe multiplayer captivante, care permite jucătorilor să se conecteze și să interacționeze în timp real, se creează fundația pentru o experiență de simulare mai avansată.

Produsul dezvoltat în prezent are în centrul atenției realizarea și cercetarea metodelor de sincronizare în timp real, a tehnicilor de compensare a latenței introdusă de rețea și a opțiunilor valabile pentru asigurarea securității și integrității datelor și a stării sistemului. Astfel, produsul se concentrează pe implementarea funcționalităților multiplayer, oferind o bază solidă pentru adăugarea ulterioară a caracteristicilor specifice tehnologiei VR, în vederea migrării spre o aplicație de tip simulator.

## Obiectivele lucrării

Scopul acestei lucrări este de a explora procesul de proiectare și dezvoltare a unei aplicații de tip joc, cu accent pe aspectele legate de multiplayer și sincronizarea în timp real. Îmi propu să realizez o platformă solidă, care să permită jucătorilor să se conecteze și să interacționeze într-un mediu virtual comun, luând parte la o experiență multiplayer captivantă. Pentru a atinge acest obiectiv, vom explora diverse tehnici și tehnologii care să ne ajute în realziarea sincronizării în rețea și în gestionarea evenimentelor în timp real.

Utilizând platformele Unity și .NET, se va dezvolta o arhitectură robustă și scalabilă, care va fi accesibilă prin intermediul aplicației client disponibilă pe sistemul de operare Windows. Implementarea funcționalităților multiplayer va implica gestionarea conexiunilor, sincronizarea acțiunilor jucătorilor și asigurarea unei experiențe fără întreruperi și cu o latență minimă.

Un obiectiv important este integrarea funcționalităților care permit jucătorilor să vadă în timp real diferite statistici și măsurători relevante pentru joc. Aceste informații includ scorul, timpul rămas, punctele de viață a caracterelor. Prin implementarea acestor funcționalități, utilizatorii vor putea monitoriza și evalua performanța lor și a celorlalți jucători în timpul jocului, ceea ce va contribui la implicarea și competiția în mediul multiplayer.

Se are în vedere crearea unei paltforme care să permită gestionarea utilizatorilor și să păstreze un istoric al meciurilor jucate. Aceasta implică implementarea unui sistem de autentificare și înregistrare a utilizatorilor, stocarea și gestionarea datelor asociate fiecărui utilizator și a rezultatelor meciurilor jucate. O platformă bine dezvoltată și intuitivă va oferi jucătorilor o experiență personalizată și le va permite să își monitorizeze progresul și să se angajeze în competiții și clasamente.

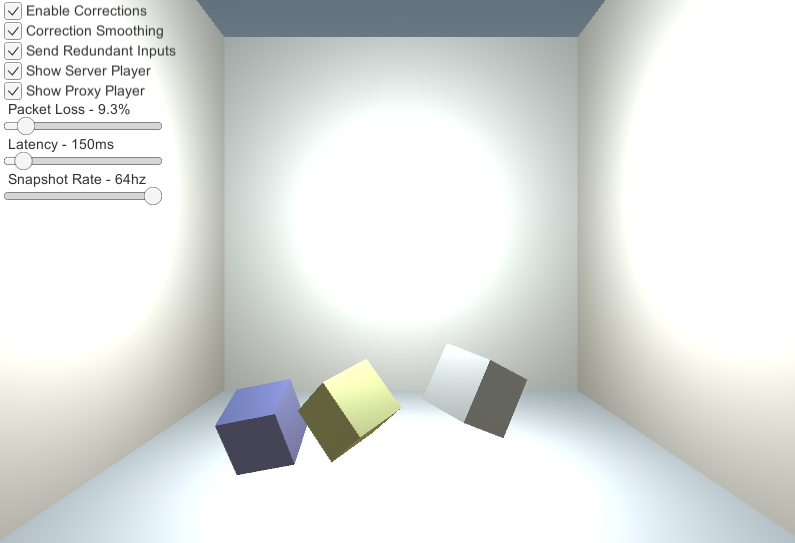
Mai mult, pentru a asigura o experiență sigură și protejată, se vor respecta standerdele de sucuritate în cadrul implementării aplicației. Comunicația va fi securizată și se vor folosi tehnici dedicate pentru păstrarea integrității stării sistemului în timpul desfășurării sesiunilor de joc.

## Metodologia de cercetare

Dată natura multiplayer a produsului software dezvoltat, un pas important este alegerea unei soluții scalabile și eficiente pentru realizarea sincronizării în rețea. Pentru a stabili cea mai potrivită opțiune, am realizat o serie de prototipuri prin care să explorez variantele disponibile.

Prima idee a fost folosirea bibliotecii Netcode, integrată în Unity. Aceasta facilitează trimiterea datelor importante despre lume și despre obiectele din scenă între mai mulți jucători în cadrul aceleiași sesiuni de joc. Însă, chiar dacă biblioteca oferă o soluție high-level pentru realizarea sincronizării datelor, lipsesc suport pentru tehnici esențiale în dezvoltarea unui joc multiplayer de acțiune, precum predicția la nivel de client și compensarea latenței în coliziuni. Aceste tehnici sunt detaliate în cadrul documentului în capitolul \*INSERT CAPITOL HERE\*.

Mai apoi, pentru explorarea și înțălegerea acestor tehnici am dezvoltat un prototip care simulează comunicarea în rețea și sincronizarea folosind tehnicile menționate anterior. Acesta funcționează pe bază de tick-uri discrete pentru realizarea sincronizării, introduce manual problemele întâlnite în comunicarea pe rețea precum latența și pierderea de pachete și realizează simularea clientului (blocul alb), a serverului (blocul galben) și a clientului văzut de un alt jucător pe care îl vom numi proxy (cubul albastru).



Prototipul gestionează intrările utilizatorului, acestea fiind stocate în cadrul unui vector local de intrări. Simultan, acestea sunt folosite și pentru realizarea simulării mișcării clientului local.

La intervale regulate de timp, clientul trimite mesaje către server conținând intrările și tick-ul asociat acestora. Aceste mesaje sunt adăugate în coada de mesaje a serverului, urmând să fie procesate si să se realizeze simularea mișcării și la nivel de server. În urma procesării, acesta trimite mesaje de stare către client, conținând informații despre poziția, rotația și viteza jucătorului pe server la un anumit tick.

Atunci când clientul primește mesajele de stare, se compară starea prezisă cu starea reală de la server și se calculează erorile de poziție și rotație. În cazul unor erori semnificative se realizează resimularea fizicii din ultimele tick-uri folosind ultima stare primită de la server și vectorul de intrări stocat local. Astfel se realizează sincronizarea, integrând predicția la nivel de client.

O lume de joc este compusă dintr-un număr mult mai mare de obiecte ce necesită sincronizare, iar fiecare obiect conține o plajă largă de proprietăți pe lângă poziție, rotație și viteză. Astfel, integrarea acestei variante cu biblioteca Unity Netcode poate face subiectul unui întreg proiect de cercetare care să dezvolte compresia, filtrarea și transmiterea datelor într-un mod scalabil și să complementeze sistemele existente în cadrul Unity pentru captare input, simulare fizică, animații.

Astfel, în urma acestei analize, am ales să folosesc biblioteca Fusion de la Photon care oferă integrare cu Unity și suport pentru tehnicile necesare, menționate anterior. Am dezvoltat un prototip 3D al unui joc multiplayer shooter, cu scopul de a experimenta capacitățile și funcționalitățile oferite de bibliotecă, urmând ca acest prototip să servească drept bază proiectului principal. 

În acest proiect, am implementat o variantă simplificată a sistemelor de mișcare, de acționare a armelor și de puncte de viață, materializate sub forma unui caracter, un mini-tanc, ce poate fi controlat de utilizator pentru a ataca ceilalți jucători conectați.

În urma analizei prototipurilor descrise mai sus, am început structurarea și dezvoltarea proiectului principal.

## Rezultatele obținute

Aici descriem in mare cam ce face proiectul acum, din puncutl de vedere al clientului. Atingem putin si gradul de sincronizare, nr de conexiuni

## Rezumatul lucrării pe capitole

Rezumat pe capitole…

# Jocuri Multiplayer

## Istoric

Istoria jocurilor video începe în anii 1950, când a fost dezvoltat primul joc electronic interactiv, numit “Tennis for Two”. Acesta ilustra un joc simplificat de tenis. Folosea doar două elemente grafice, o linie verticală pentru fileu și un punct pentru minge și era jucat pe un osciloscop. Totuși, la acel moment, jocurile video nu au părăsit laboratoarele oamenilor de știință.

În 1972, a fost înființată compania Atari. Compania nu numai că a dominat industria jocurilor video în deceniul următor, a dezvoltat și Pong, primul joc care a devenit un succes global. Principiul de joc al lui Pong este foarte similar cu cel al predecesorului său, Tennis for Two, și nu putea fi mai simplu. Instrucțiunile se limitează la afirmația: „Evitați să ratați mingea pentru un scor mare”. Cu toate că ideea jocului nu era nouă, Atari a integrat computerul și un ecran, într-o cutie cu un slot pentru monede, inventând astfel mașina de jocuri video. Pentru prima dată, un joc video era disponibil publicului larg pentru foarte puțini bani.

Jocul Space Invaders (1978) a marcat începutul epocii de aur a sălilor de jocuri, unde adolescenții își risipeau banii de buzunar pe mașinile de jocuri video. În același timp, Atari a lansat prima consolă ce permitea rularea mai multor jocuri diferite folosind casete. Peste 30 de milioane de oameni au cumpărat consola, aducând jocurile video în casele lor. La acest moment, jocurile prezentau povești și grafică simple, jucătorii se bucurau de faima adusă de clasamentele globale.

Perioada anilor 80 a adus la viață multe jocuri clasice, populare și asităzi: Pac-Man (1980), Ultima (1980), Mario Bros (1983), Tetris (1984) și SimCity (1989). Acestea au introdus povești complexe și caractere unice, iconice, care aveau să rămână în memoria oamenilor. În această perioadă a prins amploare prea bine cunoscută companie Nintendo, lansând consola de jocuri NES.

Dezvoltarea și inovațiile au început să apară. Anii 90 au introdus grafica 3D, iar odată cu trecerea în noul mileniu s-a trecut și la mediul online. Jucătorii puteau concura unii cu alții, experimentând aceeași lume de joc. Titluri precum Counter Strike sau World Of Warcraft erau jucate în principal doar pe internet. Jocurile au început să prezinte o grafică de calitate ridicată, personaje și povești complexe și lumi imense de explorat, chiar infinite.

Datorită avansului tehnologic și al popularizării telefoanelor mobile, jocurile au cunoscut schimbări la nivel demografic. Copiii și adolescenții nu mai erau singurii care petreceau timp în aceste lumi virtuale. În anul 2010, jocurile erau prezente în viața tuturor categoriilor de vârstă și pe toate platformele existente, iar capacitățile tehnologice hardware au început să nu mai reprezinte o problemă.

Astăzi, lumea virtuală și cea reală devin tot mai apropiate. Inovațiile tehnologice din domeniul realității virtuale permit simularea senzațiilor, a interacțiunilor și a fizicii într-o manieră foarte apropiată de cea reală. Jocurile au părăsit sfera divertismentului și sunt întâlnite într-o varietate de domenii, precum medicină, educație, arhitectură, fiind cunoscute sub numele de simulatoare virtuale.

## Tipuri de jocuri

Pe parcursul evoluției lor, jocurile video au dat naștere unei plaje variate de tipuri și genuri, fiecare oferind jucătorilor experiențe și provocări unice:

* Jocuri de acțiune – se pune accent pe rezolvarea de probleme în timp real, într-un ritm alert
* Jocuri de aventură – se concentrează pe explorare în ritmul ales de jucător și crează o lume narativă de poveste
* Jocuri tip paltformă – implică navigarea printre obstacole cu scopul de a ajunge la finalul nivelului, evitând pericolele
* Jocuri de supraviețuire – se bazează pe gestionarea și colectarea resurselor într-un mediu ostil
* Jocuri de rol – permit jucătorilor să își definească un personaj propriu și să se implice activ în modul sau ordinea de desfășurare a evenimentelor
* Jocuri de strategie – implică planificarea și luarea de decizii pentru a atinge obiective specifice
* Jocuri de simulare – imită diverse aspect ale vieții reale

Lista nu se oprește aici. Fiecare gen are subcategorii specifice, iar orice combinație între acestea dă naștere unui joc complet diferit. În plus, posibilitatea implementării acestuia într-o manieră singleplayer sau multiplayer crește mai mult diversitatea industriei. Iar fiecare paltformă pe care acesta poate fi implementat (desktop, mobil, tabletă, consolă, set VR), introduce un alt aspect distinctiv în cadrul experienței utilizatorilor.

## Tendințe

Realitatea virtuală (VR) și realitatea augmentată (AR) au fost două dintre cele mai semnificative avansuri în tehnologia de jocuri din ultimii ani. Potențialul acestor tehnologii este practic nelimitat, cu jocuri de societate interactive, vânători de comori virtuale, simulatoar. Realitatea virtuala si cea augmentată permit jucătorilor să se scufunde în lumi extraordinare.

Lumea jocurilor VR este deja o industrie înfloritoare, cu căștile Quest 2 de la Meta depășind vânzările consolelor Xbox Series X. Jocuri precum Beat Saber au demonstrat și ele rentabilitatea lor. AR cuprinde diverse aplicații, cum ar fi Pokémon Go, care a generat aproximativ 1 bilion de dolari în fiecare an de la lansare. Distribuția extinsă de senzori LiDAR de către Apple pe milioane de dispozitive în întreaga lume și lansarea iminentă a căștii lor ar putea deschide calea către o evoluție semnificativă în 2023.

VR are un potențial promițător pentru creștere. Conform cercetărilor efectuate de Newzoo, numărul de instalații active de seturi VR crește într-un ritm mai rapid ca niciodată, estimându-se că va crește de la 27,7 milioane în acest an la 46,0 milioane până la sfârșitul anului 2024.

Altă direcție este integrarea inteligenței artificiale (IA) într-un personaj non-jucător (NPC) care poate îmbunătăți semnificativ experiența de joc, permițând interacțiuni mai realiste și dinamice. În plus, pe măsură ce tehnologia IA avansează, putem să ne așteptăm să vedem NPC-uri care se adaptează la comportamentul jucătorilor, făcând gameplay-ul mai captivant și experiența unui simulator mult mai aproape de realitatea cotidiană.

IA nu este doar parte a experienței de joc, ci și parte a procesului de creare a jocului. De mai mulți ani, designerii folosesc IA pentru a-i ajuta să genereze elemente de joc, ceea ce îi eliberează de munca de a desena fiecare copac într-o pădure sau fiecare formațiune stâncoasă într-un canion. În schimb, designerii pot transfera această muncă la computere utilizând o tehnică numită generare procedurală de conținut, care a devenit destul de des întâlnită în industrie.

Generarea procedurală de conținut este de asemenea folosită pentru a crea niveluri de joc - uneori în mod aleatoriu - astfel încât jucătorul să se bucure de o experiență nouă de fiecare dată.

Combinația dintre evoluțiile tehnologice în aceste domenii și interesul crescut al populației în direcția jocurilor si simulatoarelor a dus la dezvoltarea a cee ace uni numesc ’’următoarea iterație a internetuluiְ’’, Metaverse. Acesta e un spațiu unic, persistent, 3D, al realității viruale unde oamenii pot interacționa ți experimenta viața în moduri imposibile în realitate.

## Studiu comparativ privind două dintre cele mai populare jocuri multiplayer

Jocurile multiplayer se află pe o curbă ascendentă din punct de vedere al popularității lor. Fortnie, un joc multiplayer third-person shooter dezvoltat de Epic Games, a înregistrat în luna iunie 2023 un număr de 242.989.948 jucători, iar Dota, un joc de tip MOBA dezvoltat de Valve, raportează că a avut în luna ianuarie, a aceluiași an, 9.821.803 jucători (1).

Am ales aceste două jocuri pentru a evidenția diferențele în cadrul tehnicii de sicnronizare și pentru a observa cum diferiți producători abordează problema trișorilor sau a găzduirii sesiunilor de joc.

Primul, fiind un joc de acțiune de mare viteză, folosește pentru sincronizare predicția la nivel de client. Pe baza input-ului utilizatorului se realizează simularea la nivel local pentru a diminua efectul de latență al rețelei. În același timp, acesta e trimis serverului care simulează și el aceeași lume de joc. Serverul trimite clientului un flux de date care reprezintă starea reală a lumii și jucătorilor, iar clientul reia simularea lcoală pentru a corecta orice eroare apărută.

Pe de altă parte, jocurile de tip MOBA au o abordare puțin diferită. În cazul acestora nu este dorit un răspuns instantaneu la input, o mica întărziere întărind ideea că utilizatorul este o entitate ce controlează o unitate, ci nu este unitatea ca în cazul primului joc. Astfel, input-ul utilizatorului este capătat și trimis la server. Acesta realizează simularea tinând cont de ordinea cronologică a eveniemntelor și trimite noua stare clietului, care actualizează lumea local.

Din punct de vedere al soluțiilor anti-cheat, Fortnite folosește o soluție numită Easy Anti-Cheat (EAC) dezvoltată de Epic Games, care este accesibilă și altor dezvoltatori. Dota folosește o soluție proprie Valve, care e folosită doar în jocurile dezvoltate de acest producător, numită Valve Anti-Cheat (VAC). Ambele variante au ca scop detectarea și sancționarea jucătorilor care utilizează software sau metode nelegitime pentru a obține un avantaj în jocuri. Utilizatorii pot raporta alți utilizatori care par suspecți. Sistemul filtrează cazurile raportate, urmând să fie verificate manual de alți jucători sau de indivizi specializați. Cazurile în care sistemul sancționează automat sunt puține și doar în cazurile metodelor clasice și bine cunoscute de trișat.

În ceea ce privește modalitățile de găzduire a sesiunilor de joc, există diferențe semnificative între cele două jocuri. Fortnite este un joc care se bazează pe un model situat strict în cloud, ceea ce înseamnă că toate meciurile sunt găzduite pe serverele Epic Games și nu există o opțiune directă pentru jucători de a rula local propriile lor meciuri. Pe de altă parte, Dota 2 oferă mai multă libertate. Jucătorii pot alege între opțiunea de a găzdui jocuri în rețeaua locală sau de a utiliza serverele dedicate Valve. Există și opțiuni pentru jocuri personalizate, unde jucătorii pot crea propriile lor lobby-uri și să invite alți jucători să se alăture.

# Game engines

## Definire generală

Un game engine este un produs sau un mediu software folosit original pentru a facilita dezvoltrea jocurilor video. Astăzi, un game engine oferă o plajă mult mai variată de utilizări integrand tool-uri pentru randare, simularea fizicii, detecția coliziuniilor, animare, inteligență artificială, managementul memoriei, rețelistică. (1)

În majoritatea cazurilor, scopul principal este înaintarea unor suite de dezvoltare vizuală si a unor componente software cu grad mare de reutilizabilitate. Dacă abordarea inițială asupra dezvoltării unui joc video era creearea unei aplicații de la 0, cu posibile reutilizări de cod în cadrul proiectelor similare, game engine-urile permit dezvoltarea unui număr nelimitat de aplicații folosind aceeasi baza, aceleași sisteme si aceleași tool-uri reducând semnificativ timpul si complexitatea dezvoltării unui proiect.

Fiecare game engine are focusul său propriu. Unele oferă support pentru randare 2D, 3D sau ambele. Unele prezintă un sistem de simulare a fizicii foarte dezvoltat care permit dezvoltarea simulatoarelor performante.

## Soluții existente

### Unity

Este o platformă de dezvoltare care se află pe piată din 2005, fiind dezolvtată de Unity Technology. Game engine-ul oferă un set bogat de tool-uri integrate, acoperind o plajă variată de funcționalități fără a fi nevoie de descărcări adiționale.

Avantajele principale ale sistemeului sunt:

* Compatibilitatea aplicațiilor rezultate cu cel puțin 20 de platforme diferite printre care putem enumera Windows, mobil, web, consolă.
* Comunitatea este imensă oferind o gamă largă de tool-uri suplimentare, resurse, tutoriale și forum-uri
* Asset store prorpiu
* Intuitiv pentru începători
* Limbaj de dezvoltare C#
* Gratis, până câștigurile produsului realizat depășesc o anumită sumă
* Sistemul oferă un pipeline pentru randare de înaltă calitate

Singurele dezavantaje de menționat sunt faptul că soluția nu este open-source și ca pipeline-ul de randare este inferior celui oferit de UnrealEngine, competitorului direct de pe piața.

### Unreal Engine

Este o platformă oferită de Epic Games cu focus pe dezvoltarea unor jocuri de calitate AAA pentru PC si console, care a apărut pe piață intrucât compania a decis să comercializeze game engine-ul folosit intern.

Avantajele principale ale soluției sunt:

* Suport pentru grafică HD, ajungând la un nivel fotorealistic
* Sistem de luminozitate și particule avansate
* Performanțe excelente când se lucrează cu resurse complexes sau abundente ca număr
* Limbajele de dezvoltare sunt C++ și Blueprint Visual Scripting, un limbaj care permite dezvoltarea fără să necesite scrierea de cod
* Compatibilitate cu o gamă variată de paltforme
* Gratis de folosit, cu mentiunea că 5% din câștiguri sunt reținute de companie
* Open-source

Printre dezavantajele sistemului putem enumera complexitatea interfeței, curbă de învățare abruptă și necesitatea unor resurse de sistem puternice pentru a folosi intregul potential. (2)

### Godot

O altă solutie care câștigă popularitate, apărută în 2014 în umbra celor două mari platforme prezentate anterior, este Godot.

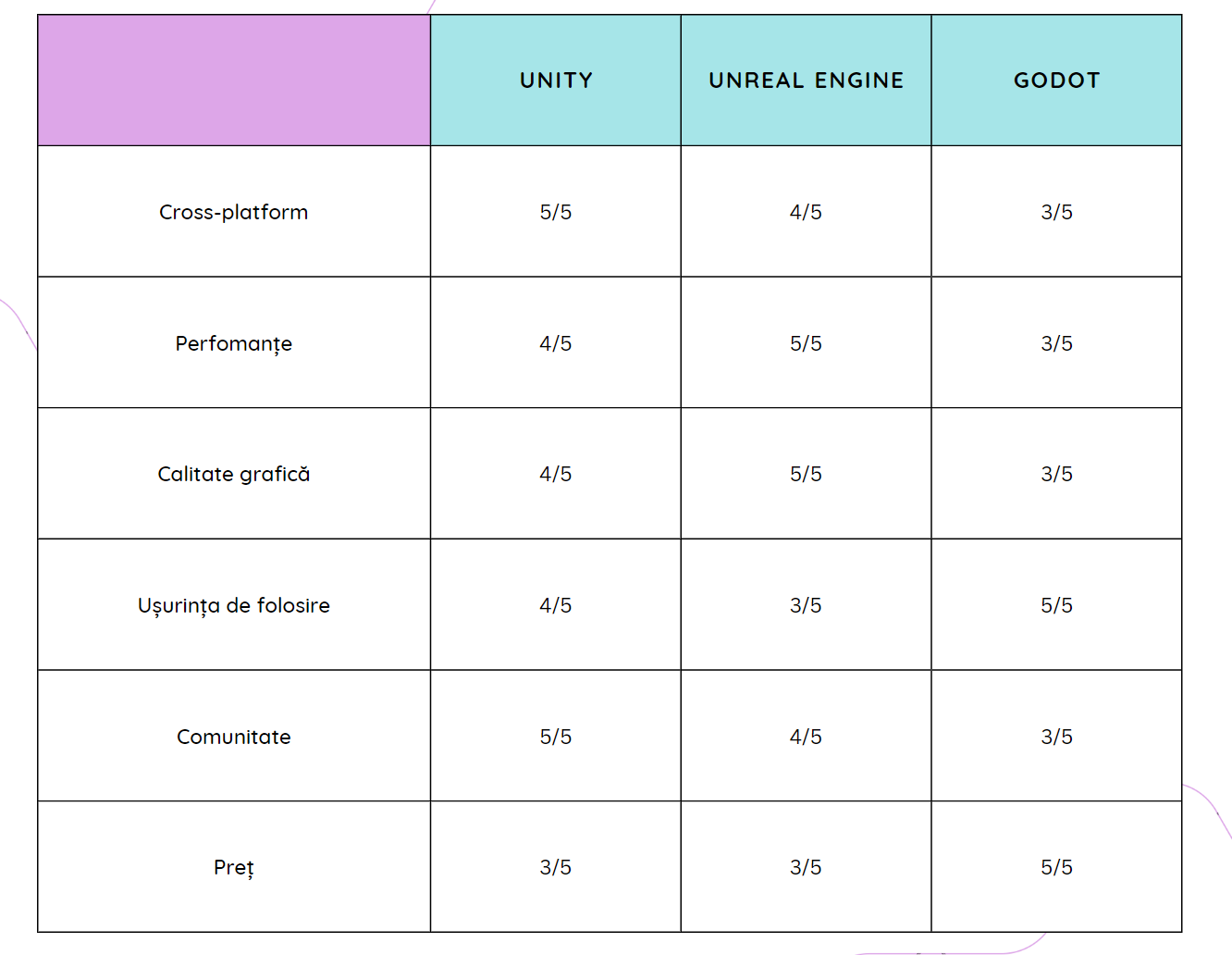
Aceasta se remarcă prin caracterul complet gratuit și open-source, reușind să ofere suport 2D și 3D pentru aplicații compatibile cu o gamă decentă de paltforme. Comunitatea, chiar dacă are o istorie mai scurtă, este activă și aflată intr-o continuă creștere.

Godot suportă pentru dezvoltare limbaje precum C# și C++, însa limbajul de scripting implicit este GDScript, un limbaj specific acestui game engine similar cu Python.

Limitările acestei soluții:

* Resurse limitate; desi există multe resurse disponibile, numărul lor nu se poate compara ce cele provenite de la competitorii direcți
* Performanțe limitate pentru jocurile mari
* Limitări din punct de vedere al calității grafice

### Tabel de comparație



# Tehnologii Folosite

## .NET

Compatibilitatea și complementaritatea dintre cele două medii de dezvoltare, Unity și .NET, dă formă unei tehnologii ce oferă flexibilitate și putere de dezvoltare. În spatele acestei integrări se află Mono, un runtime open-source pentru .NET, care servește ca punte între cele două platforme.

Mono permite Unity să utilizeze limbajul de programare C# și bibliotecile .NET pentru scripting și dezvoltarea aplicațiilor. Acesta asigură compatibilitatea între .NET Framework și Unity, permițând aplicațiilor Unity să ruleze cod C# și să acceseze bibliotecile .NET. Astfel, dezvoltatorii pot folosi toate facilitățile și funcționalitățile oferite de .NET în cadrul proiectelor Unity.

### Limbajul C#

C# este un limbaj de programare modern, orientat pe obiecte, care își are rădăcinile în familia de limbaje C. Acesta facilitează dezvoltarea unei varietăți de aplicații sigure și robuste care rulează pe .NET și .NET Core.

Limbajul a fost dezvoltat în cadrul companiei Microsoft, prima implementare fiind lansată pe scară largă în anul 2000, ca parte a inițiativei .NET Framework. Scopul a fost ca limbajul să fie unul simplu, modern și general. Cu timpul, acesta a devenit un limbaj portabil cu suport pe mai multe platforme, inclusiv sisteme embedded. (14)

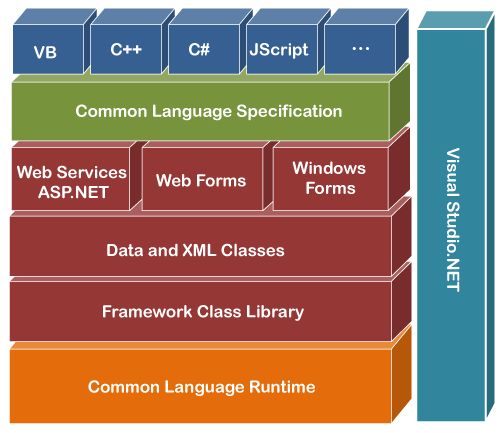
Mai multe caracteristici încadrează limbajul în categoria celor mai populare și folosite limbaje printre care se enumeră: managerierea automată a memoriei eliberată sau neutilizată, protecția oferită de tipurile nullable împotriva variabilelor care nu fac referire la obiecte alocate, susținerea tehnicilor de programare funcțională prin funcțiile lambda, creare unui model comun pentru lucrul cu date din orice sursă bazat pe LINQ, suportul pentru dezvoltarea sistemelor distribuite folosind operațiile asincrone. (15)

### Arhitectura .NET

Programele dezvoltate în limbajul C# rulează pe baza platformei .NET, a unui sistem virtual de execuție numit Common Language Runtime și a unui set de biblioteci. CLR este implementarea Microsoft a infrastructurii CLI, un standard internațional. CLI stă la baza creării mediilor de execuție si dezvoltare în care limbajele ți bibliotecile funcționează împreună într-un mod transparent.

Codul sursă scris în limbajul C# este compilat într-un limbaj intermediar (Intermediate Language - IL) care respectă specificațiile CLI. Codul IL și resursele, cum ar fi imagini și șiruri de caractere, sunt stocate într-o fișier executabil, de obicei cu extensia .dll.

La executarea programului C#, fișierul executabil este încărcată în CLR. Acesta efectuează compilarea Just-In-Time (JIT) pentru a converti codul IL în instrucțiuni native mașinii. CLR oferă și alte servicii legate de colectarea automată a zonelor de memorie eliberate, gestionarea excepțiilor și gestionarea resurselor. (16)



Sursă: https://static.javatpoint.com/tutorial/vb-net/images/vb-net-dot-net-framework-introduction.png

## Visual Studio

Visual Studio este un mediu de dezvoltare integrat (IDE) dezvoltat de Microsft. Este folosit pentru dezvoltarea de aplicații software precum site-uri web, servicii web, aplicații desktop și aplicații mobile. El folosețte platforme de dezvoltare Microsoft precum Windows API, Windows Forms, Windows Presentation, Windows Store și Microsoft Silverlight.

Oferă suport nativ pentru 36 de limbaje de programare diferite și permite editorului și debugger-ului să lucreze cu aproape orice limbaj, atât timp cât există un serviciu specific pentru acel limbaj disponibil. Printre limbajele native se regăsesc C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, XML, HTML și CSS. Suportul pentru alte limbaje precum Python sau Ruby este posibil prin instalarea plugin-urilor specifice.

Visual Studio include un editor code care facilitează dezvoltarea prin funcționalități precum evidențierea sintaxei, completarea codului folosind IntelliSense și generare automată de metode.

IDE-ul include, de asemenea, un debugger puternic care facilitează identificare și rezolvarea erorilor. Acesta permite oprirea temporară a rulării programului pentru analizarea memoriei curente, editarea codului în timpul rulării în modul debug și diferite metode de vizualizare a variabilelor. (14)

Diferite ediții sunt disponibile pentru a satisface nevoile diferite ale dezvoltatorilor: Community, Professional și Enterprise. Prima este gratuită și oferă o suită completă de instrumente necesare dezvoltării. Fiecare ediție oferă acces la caracteristici suplimentare, niveluri diferite de suport și integrare cu alte tehnologii Microsoft.

În cadrul dezvoltării aplicației a fost folosită ultima versiune disponibilă, Visual Studio Community 2022. Aceasta oferă serviciu de integrare a sistemului de control al sursei, Git, suport avansat pentru depanarea codului, integrare cu editorul Unity și plugin-uri care permit managementul bazelor de date.

## Blender

Blender este un produs software gratuit de modelare 3D, utilizat în industria de animație, vizualizare arhitecturală, jocuri video. Suportă întregul pipeline necesar unui dezvoltator: modelare, rigging, animare, simulare, randare, compoziție și urmărire a mișcării.

Blender este un proiect dezvoltat de comunitate sub licența GNU, oferind suport cross-platform pentru dispozitive Linux, Windows și Macintosh. Publicul poate aduce modificări, să repare bug-uri și să contribuie la dezvoltarea generală a aplicației. Scripturi Python pot fi folosite pentru personalizarea acesteia și dezvoltarea de tool-uri specializate. (15)

Am ales această soluție pentru editarea modelelor 3D datorită gamei largi de instrumente pentru modelare și editare, interfeței intuitive și personalizabile și comunității numeroase care oferă sprijin prin intermediul forumurilor și a tutorialelor.

## Unity

Unity este produsul software de tip motor de joc ales pentru dezvoltarea aplicației. Având în vedere importanța acestuia în cadrul proiectului, voi dedica un capitol acestui subiect.

# Unity

## Viziunea

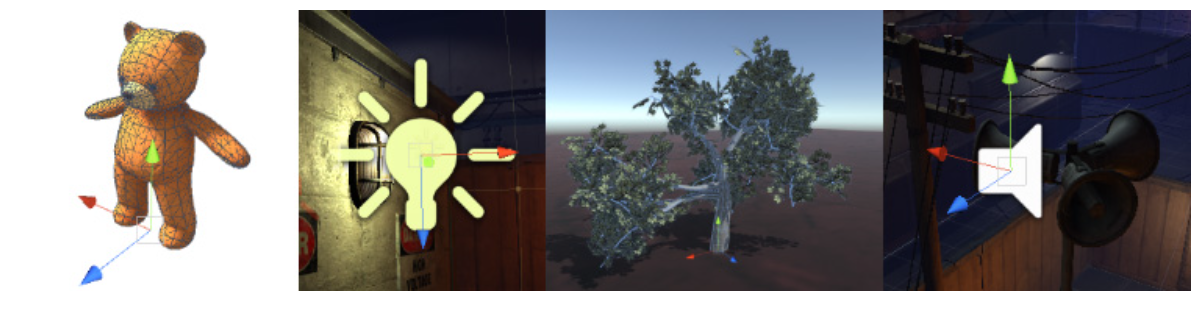
Unity și-a caștigat popularitatea fiind unul dintre cele mai dezvoltate game engine-uri cross-platform, însă este folosit pentru a crea o gamă mai largă de aplicații, nu doar jocuri.

John Riccitiello, CEO-ul Unity Technology, a afirmat ca vede Unity, și viitorul acestuia, ca o modalitate de a creea atât experiențe de realitate virtuală si auguemntată, simulatoare și jocuri, cât și experiențe interactive pentru industrii precum arhitectură, auto, filme, televiziune. Mai mult, acesta a evidențiat importanța accesului ușor la tehnologii si tool-uri pentru o plajă cât mai largă de creatori (3).

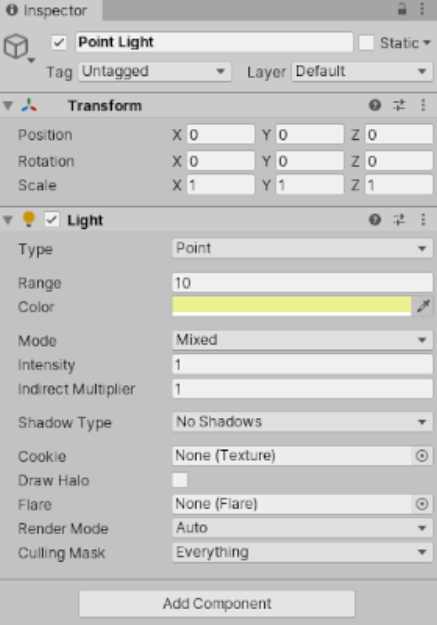
Această viziune se alinează cu modul intuitiv și ușor de folosit a mediului de dezvoltare pe care compania îl face accesibil gratuit.

## Arhitectura modulară bazată pe componente

Fiecare obiect din joc este un **GameObject**, de la caractere și item-uri colectabile, la lumini, cameră și efecte speciale. Totuși, un GameObject nu poate să facă nimic singur, are nevoie să primeasca niște proprietăți pentru a putea funcționa ca un caracter, obiect de mediu sau effect special.

Patru tipuri diferite de GameObejct: un caracter, o lumină, un copac și o sursă de sunet

Astfel, GameObjects sunt obiectele fundamentale din Unity, iar proprietațile care le oferă o funcționalitate se numesc componente. În funcție de tipul de obiect dorit și de comportamentul dorit pentru acesta, pot fi adăugate diferite componente. Unity oferă o paletă extinsă de componente pre-implementate, iar dezvoltatorul poate crea propriile componente folosint Unity Scripting API, prezentat mai jos. Mai mult, pe lângă funcționalitate, componentele oferă o metodă de a schimba valorile variabilelor din interfața grafică (4).

De exemplu, o sursă de lumină este creată atașănd o componentă ’’Light’’ unui GameObject și toate obiectele conțin o component ’’Transform’’ care indică poziția, rotația si dimensiunea în lumea de joc.

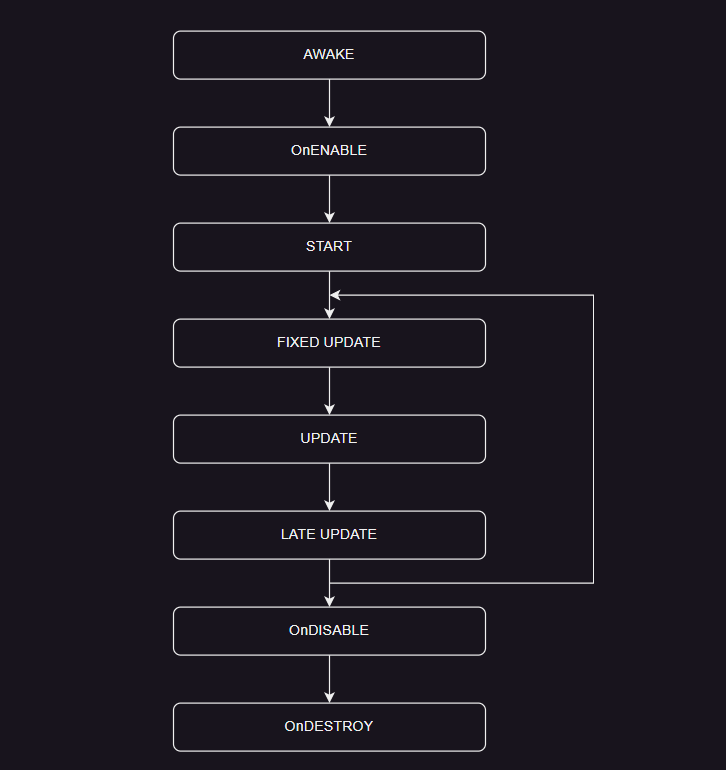
GameObject sursă de lumină

Unity Editor, componentă de lumină adăugată

## Unity Scripting API

Am stabilit că lumea de joc este compusă dintr-o multitudine de GameObjects, fiecare având funcționalități diferite in funcție de componentele atașate. Însă, ca dezvoltatori, avem nevoie de o metodă prin care să definim și să modificăm aceste funcționalități. Sistemul de scripting este cel care ne permite acest lucru, permițându-ne să creem intr-o manieră flexibilă si accesibilă o logică a jocului pe baza unui limbaj de programare, C# in cazul nostru.

Unity Scripting API este cosntruit în jurul clasei MonoBehaviour. Orice script trebuie sa conțină o clasa care moștenește MonoBehaviour, acest lucru asigură integrarea scriptului în ciclul de viață al obiectelor din joc. Principalele momente din ciclu de viață sunt reprezentate de metodole apelate o singură dată la creearea obiectului (Awake, OnEnable, Start), metodele apelate in mod repetitiv cât timp obiectul există in lumea de joc (FixedUpdate, Update, LateUpdate) și metode apelate o singură data în momentul distrugerii obiectului(OnDisable, OnDestroy). Toate aceste metode pot fi suprascrise pentru a creea comportamente personalizate pentru obiectele din joc. (5)



Unity Scripting API este organizat ierarhic folosind namespace-uri, printre care cele mai importante două fiind UnityEngine și UnityEditor. Metodele menționate mai sus, împreună cu alte metode care controleză fizica, animațiile, randarea și GUI, fac parte din namespace-ul UnityEngine și faciliteza creeare logicii de joc.

Cel de al doilea namespace important, aflat în partea superioară a ierarhiei, este UnityEditor care oferă modalități de a modifica însuși editorul pentru a permite dezvoltatorilor sa creeze tool-uri sau să adapteze interfața intr-un mod care să sporească nivelul de productivitate.

## Sistemul de simulare fizică

În general, produsele software de tip game engine, oferă posibilitatea de a simula fizica in cadrul proiectelor luând în considerare lucruri ca: legile mișcării, coliziuni, gravitație și o varietate de alte forte. Pentru a realiza acest lucru se folosește un alt produs software numit physics engine.

În cadrul jocurilor video și a simulatoarelor se preferă acele physics engine care procesează datele în timp real, în defavoarea celor cu o precizie mare. Această alegere se datoreză naturii interactive a aplicațiilor de acest gen, astfel folosindu-se calcule simplificate cu o acuratețe scăzută pentru a îmbunătății rata de actualizare a jocului și, simultan, expreința utilizatorilor (6).

Unity are integrate două opțiuni diferite pentru fizica jocurilor, folositoare în cazul proiectelor ce folosesc o abordare orientate pe obiect, Nvidia PhysX engine pentru 3D și Box2D engine pentru 2D, și două soluții ce pot fi instalate suplimentar pentru o abordare orientată pe date, unity Physics și Havok Physics for Unity. Astfel, platforma reușește sa crească eficiența și performanța pasând unității de procesare grafice calculele computaționale necesare simulării.

Prin aceste soluții software expune o diversitate de funcționalități precum:

* Simularea mișcării caracterelor și obiectelor într-un mod natural, permițându-le să cadă și să se rostogolească într-un mod realist – ragdoll physics
* Detectarea coliziunilor dintre obiecte
* Simularea comportamentului obiectelor moi – dinamica corpului fluid
* Simularea comportamentului corpului rigid – dinamica corpului rigid
* Conectarea mai multor obiecte, permițându-le să se miște sau rotească împreună – joints phyisics
* Propagarea sunetului în funcție de mediu și obiectele din scenă
* Transmiterea de raze în lumea jocului pentru a prelua informații despre obiectele existente – raycasting

## Sistemul de randare

Lumea virtuală expusă pe ecranele noastre 2D nu dispune cu adevărat de a 3a axă adâncimii, lumini și umbre, texturi mai riguroase sau mai fine. Toate aceste lucruri sunt reprezentate folosind calcule matematice complexe care permit proiectarea unei lumi 3D într-o imagine 2D și modificarea culorilor pixelilor acelei imagini astfel încât să creeze impresia unei texturi, lumini sau umbre într-o lume bidimensională. Sistemul de randare este componenta responsabilă pentru a implementa aceste calcule, a crea imaginea 2D si a o afișa pe ecran.

În cazul unui joc sau simulator, lumea ce necesită a fi randată se află într-o continuă schimbare. Imaginea este continuu actualizată, luând în calcul mișcarea și perspectiva camerei, care reprezintă punctul de vedere din cadrul aplicației, și sursele de lumină existente. Datorită acestui caracter dinamic, sistemul de randare recalculeaza în întregime imaginea specifică fiecărui frame afișat pe ecran. Se folosește o tehnica cu buffer dublu pentru a creea o tranziție plăcută între frame-uri: sunt folosiți doi vectori pentru randare; primul conține imaginea care este afișată pe ecran în prezent, iar al doilea este folosit pentru randarea următorului frame; în timp ce al doilea buffer este calculat, primul este afișat pe ecran; odată ce ranadarea următorului frame este gata, cei doi vectori sunt interschimbați și procesul continua să se repete.

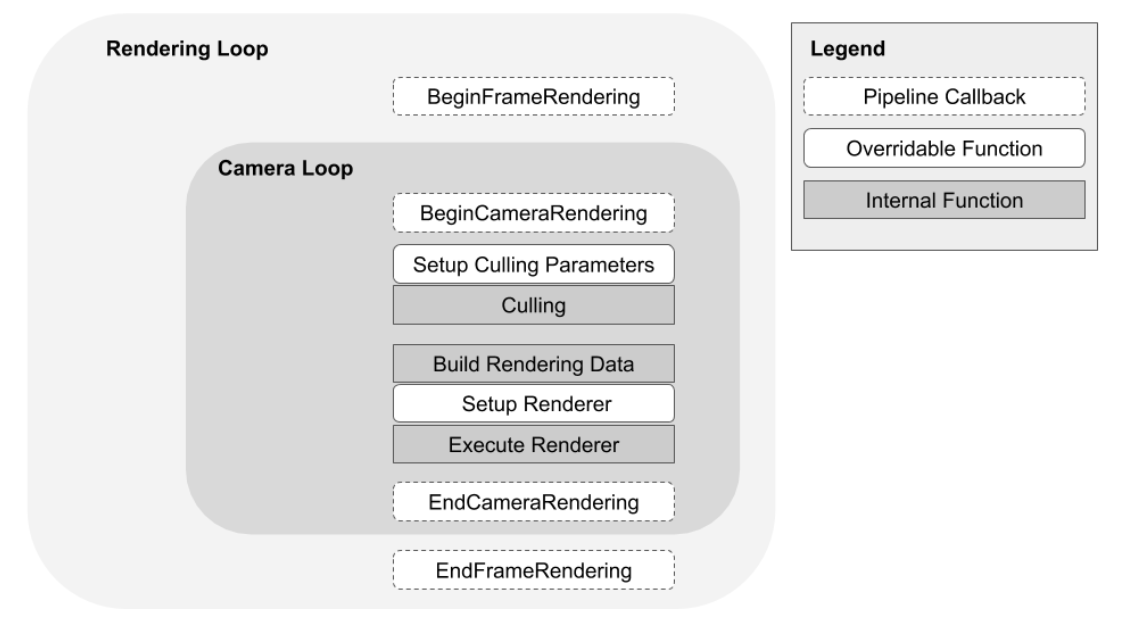
Întrucât aceste procese și calcule necesită existența unor tehnologii de randare în timp real optimizate pentru a obține un nivel de eficiență crescută, ele intră in componența librăriilor grafice low-level, care reprezintă o interfață pentru implementările propriu zise a producătorilor de plăci grafice.

Unity soportă majoritatea soluțiilor mari existente, DirectX, Metal, OpenGL și Vulkan, în funcție de platforma de dezvoltare sau de paltforma țintă și permite selectarea uneia dintre acestea din cadrul editorului în timpul dezvoltării. (7)

Unity pune la dispoziție două soluții propri și anume două pipeline-uri de randare integrate:

* Unity Rendering Pipeline (URP), care oferă utilzatorilor un workflow artist-friendly pentru a crea grafică optimizată compatibilă cu o plajă mare de platforme
* High Definition Render Pipeline (HDRP), care este specializat în afișarea de grafică de înaltă calitate, cu suport pentru tehnici și efecte avansate și este optimizat pentru un număr limitat de platforme

De exemplu, acesta e loop-ul specific URP:

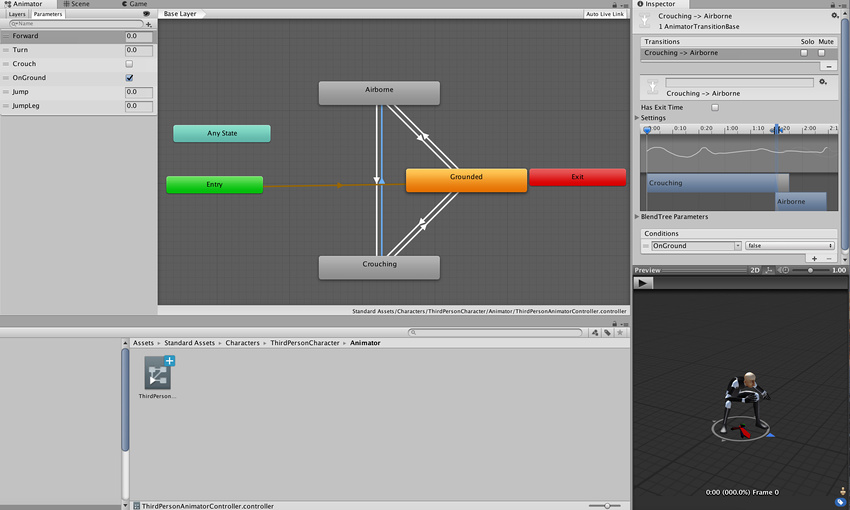


## Sistemul de animații

Animațiile sunt o componentă esențială în experianța utilizatorilor. Putem vorbi de animații 2D care să aducă la viață interfețele, animații 3D simple care să exprime mișcarea unei mingi sau a unei mașini sau chiar de animații 3D complexe care să imite într-un mod cât mai organic mișcarea unui caracter humanoid. Deci pentru a crea o experiență bine animată, atât din punct de vedere estetic cât și al perfermonței, necesită un nivel înalt de pregătire. De aceea, majoritatea game engine-urilor integrează o componentă menită să faciliteze animarea entităților din joc, deseori folosind atât o componentă vizuală cât și una bazată pe scripturi.

Componenta oferită de Unity poartă denumirea de ’’Mecanim’’ și se bazează pe conceptul de clipuri de animații, care conțin informație despre cum poziția, rotația și alte proprietăți a obiectelor din scenă la anumite momente de timp. Fiecare clip poate fi vazut ca o înregistrare lineară ce conține aceste proprietăți.

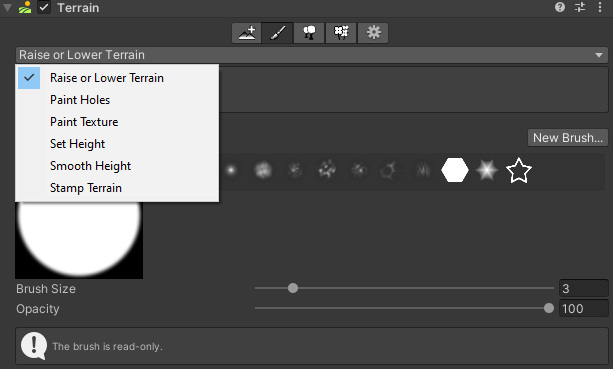
Clipurile de animații sunt organizate într-un sistem asemanător unei organigrame, numit ’’Aniamation Controller’’. Acesta acționează ca un state machine care ține evidența clipului curent, clipurilor ce urmează a fi redate sau a clipurilor ce trebuie contopite și redate împreună (8).



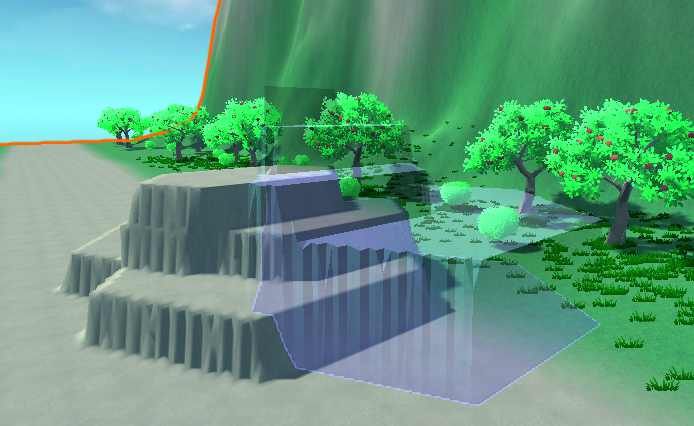
## Sistemul de editare e terenului

Unity conține o componentă integrată care permite creeare unui teren și a unui peisaj pentru lumea de joc. Componenta permite crearea unor ‘’parcele de teren’’ pentru care se pot edita înălțimea, textura, culoarea obținând rezultate asemănătoare munților, dealurilor sau văilor.

Editarea terenului este prezentată sub forma unor pensule care, pe baza unor filtre de dimeniune și opacitate, permit operațiuni precum; ridicarea sau coborârea nivelului, setarea terenului ca transparent, aplicarea de texturi, nivelarea terenului.



Componenta de editare a terenului cu modurile de funcționare disponibile

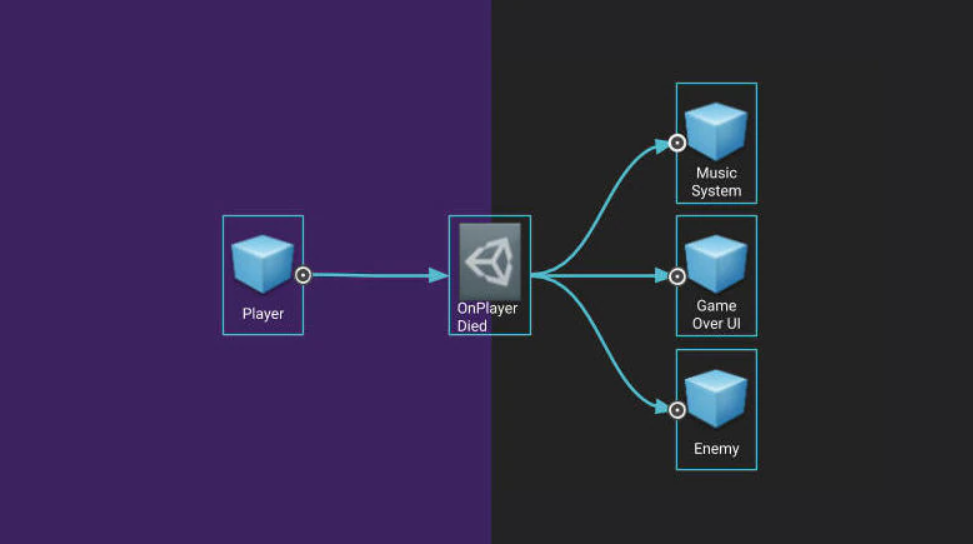


Pensula de ridicare a nivelului

## Scriptable Objects

ScriptableObjects este o entitate specifică Unity, este o clasă serializabilă care permite stocarea unor cantități mari de date și impărtășirea eficientă a acestora între diferitele sisteme ce compun aplicația aflată în dezvoltare.

Practic, pentru a utiliza aceasta tehnologie, trebuie sa creezi o clasa care să moștenească clasa ’’ScriptableObject’’. Obiectul nou creat este unic și referențiat în mai multe scripturi, acționând ca o memorie împărtășită între acestea.



Srusa: https://unity.com/how-to/architect-game-code-scriptable-objects

Printre beneficile aduse de acest tip de obiecte se regăsesc ușurința de implementare, facilitarea unei abordări modulare de dezvoltare, ușurința de schimbare și editare a componentelor, facilitarea procesului de debug (9).

# Biblioteca Photon Fusion

Fusion este o bibliotecă performantă care asigură comunicarea în rețea pentru aplicațiile dezvoltate în Unity. Este integrată intr-un mod natural în workflow-ul specific Unity și oferă funcționalități avansate precum compresia datelor, predicție la nivel de client și tehnici de compensare a întârzierilor în rețea. (11)

Biblioteca se bazează pe algoritmi de compresie de ultimă generație pentru a reduce lățimea de bandă necesară folosind resurse minime pe CPU, iar datele sunt transferate sub forma unui pachet compresat ce conține o captură a întregii stări a aplicației la un moment dat, pe care o vom numi pe snapshot. Desigur, snapshot-ul nu conține toate datele necesare rulării jocului, ci doar acelea marcate ca făcând parte din bucla de simulare specifică Fusion.

Pentru sincronizarea buclei de simulare, biblioteca implementează o simualre bazată pe ticuri și permite funcționarea în două moduri diferite: ’’Hosted’‘ și ’’Shared’’. În continuare mă voi axa pe detalierea primul mod, întrucât acesta este folosit în dezvoltarea aplicației prezentate.

#### Modul ’’Hosted’’

Modul ’’hosted’’ permite aplicației funcționarea fie ca un server de sine stătător, fie ca un client și un server combinate, îndeplinind funcționalitățile amândurora. În ambele variante, serverul are drepturi exclusive asupra stării obiectelor care sunt marcate ca facând parte din bucla de simulare Fusion. Pentru ca un client să modifice stare unui astfel de obiect este necesar să transmită un mesaj serverului.

Orice modificare adusă de client unui obiect înscris în rețea este, defapt, doar o modificare locală, care va fi suprascrisă odată ce este primit și procesat snapshot-ul de la server cu starea jocului validată de acesta. Acest proces poartă numele de reconciliere: starea clientului este derulată înapoi la cea validă, primită de la server, urmând să resimuleze toate stările până în prezent. Dacă starea procesată de client este identică cu cea primită de la server atunci procesul acesta este inutil, însă în caz contrar starea clientului va fi actualizată la una validă.

La rularea aplicației din spatele unui firewall sau a unui router, Photon Cloud acționează ca un relay server sau încearcă să creeze o breșă în firewall și să stabilească o conexiune directă. Totuși, sesiunea este deținută de aplicația ce rulează în modul host și va fi distrusă dacă acesta se deconectează.

#### Componente principale

API-ul oferit de Fusion oferă o gamă de componente care, atașate obiectelor din Unity, adaugă funcționalități specifice comunicării în rețea și o plajă de clase care, dacă sunt moștenite, înregistrează clasa derivată în ciclul de viață al buclei de simulare Fusion. Astfel, biblioteca adăuga funcționalităti păstrând obiectele obișnuite din Unity ca piesă centrală a dezvoltării.

Două dintre elementele componentele principale sunt NetworkRunner și NetworkObject. Prima poate fi văzută ca nucleu central al bibliotecii la nivelul aplicației aflate în derulare întrucât este unic și managerează atât comunicarea cât și simularea, atât pe servere cât și pe clienți. A doua reprezintă o modelitate de identificare a obiectelor la nivelul rețelei în timpul rulării. Atașând această componentă unui obiect Unity, el este marcat ca facând parte din simularea sincronizată în rețea.

În capitolul \*INSERT NUME CAPITOL\* am introdus clasa MonoBehaviour ca element central în cadrul sistemului de scripting din Unity. Fusion introduce alte două clase care extind comportamentul acesteia: SimulationBehaviour și NetworkBehaviour. Prima este clasa de bază folosită pentru a introduce un obiect în bucla de simulare Fusion prin simpla moștenire a clasei ce definește obiectul din aceasta, în locul moștenirii din MonoBehaviour. Cea de a doua este derivată din prima și introduce posibilitatea de a adăuga variabile de stare ce for fi sincronizate în rețea (12).

Aceste variabile de stare sunt defapt simple proprietăți clasice din C# marcate cu tag-ul ’’Networked’’. Valorile acestora sunt integrate în cadrul snapshot-urilor și folosite în procesul de reconciliere.

#### Bucla de simulare Fusion

Fusion rulează o simulare în timp discret bazată pe tick-uri. Procesul general care asigură acest lucru poartă denumirea de ’’buclă de simulare Fusion’’ și implementarea acestuia se bazează pe clasele descrise anterior: NetworkBehaviour și SimulationBehaviour.

În continuare voi detalia conceptele care stau la baza funcționării acestui mod de simulare și care permit realizarea sincronizării în rețea.

##### 3.1 Tick-uri

Una dintre problemele întâlnite în cazul aplicațiilor ce necesită sincronizare la distanță este posibilitatea apariției incosistenței din cauza diferenței de viteză a conexiunii sau a puterii de procesare a dispozitivelor clienților. Pentru a rezolva această problemă, Fusion utilizează unități discrete de timp numite tick-uri țn locul sincronizarii clasice bazate pe timp. Rata acestor tick-uri nu are legătura cu viteza de trecere a timpului, iar folosirea lor în detrimentul unui ceas hardware permite clienților să aibă o referință comuna de timp. Intervalul de timp dintre fiecare tick este definit in hertzi și este accesibil în cod prin intermediul componentei unice ce stă la baza sesiunii de joc NetworkRunner.

Fiecare tick al simulării are asociat un snapshot. Pentru a trece la următorul tick și, implicit, a genera următorul snapshot asociat, Fusion va invoca metodele specifice pentru fiecare obiect derivat din SimulationBehaviour sau NetworkBehaviour în același mod în care și Unity invocă metodele din cadrul ciclului de viață al obiectelor. (13)

##### 3.2. Predictia

Unul din cel mai important aspect al unei experiențe reușite în cadrul unei aplicații în timp real este timpul de răspuns al sistemului, în special în cazul unei aplicații multiplayer.

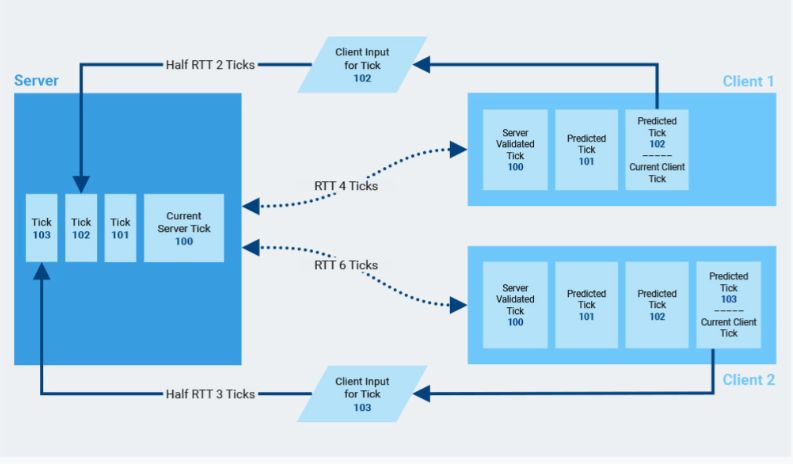
Din motive de securitate, acțiunile jucătorilor trebuie validate pe server. Această validare introduce latență în sistem, întrucât fiecare acțiune trebuie să ajungă la server, să fie validată și să fie trimisă înapoi la client. În funcție de locația clientului sau puterea concetivității acestuia la internet, latența poate avea valori insesizabile sau poate atinge valori care să distrugă complet experiența utilizatorului.

Pentru a reduce această latență, păstrând și validarea la nivel de server, se folosește o tehnică numită predicție la nivel de client. Tehnica presupune ca clientul să răspundă acțiunilor utilizatorului instant pentru ca efectele latenței să nu mai fie resimțite și, în paralel, să trimită input-ul acestuia la sever pentru validare.

Astfel, întârzierile cauzate de rețea sunt minimizate, însă apare problema desincronizării dintre client și server întrucât fiecare rulează propria simulare. Această problema este rezolvată folosind o tehnică numită reconciliere, pe care am descris-o anterior.

Pentru a întelege mai bine acest proces, voi parcurge un exemplu cu 2 clienți conectați la același server care doresc să miște un caracter, fiecare având o latență diferită.

Atunci când clientul 1 dorește sa miște caracterul, el prezice starea viitoare a acestuia bazându-se pe ultimul tick validat de server. În funcție de round-tripul comunicaării, serverul decide un număr de tick-uri necesar pentru a fi prezis (în cazul exemplului nostru round-tripul este de 4 tick-uri, deci trebuie prezise 2 tick-uri). Având în vedere ca ultimul tick validat este cel cu numărul 100, clientul 1 comunică serverului input-ul său pentru tick-ul 102 astfel încat să existe suficient timp pentru pachet să ajungă la server până în momentul simulării tick-ului 102 al nivel de server. Dacă nimic nu a interferat cu clientul in aceste tick-uri, starea pentru tick-ul 102 a clientului si cea validată a serverului for fi identice.



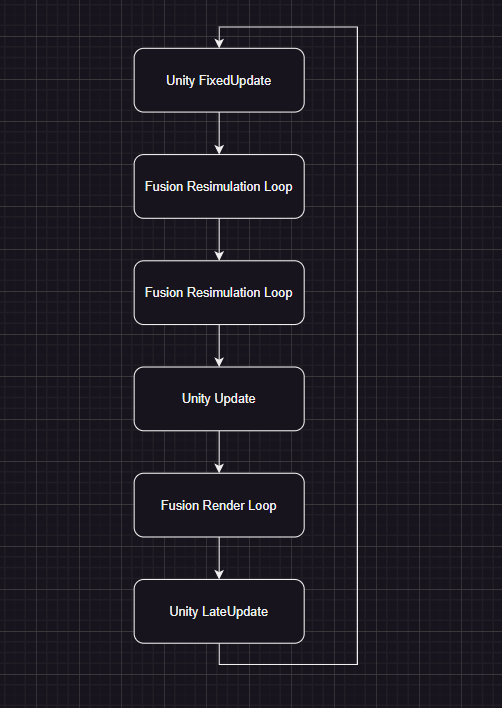
Sursă: https://doc.photonengine.com/fusion/current/manual/network-simulation-loop

Clientul 2 realizează aceleași operații, însă are round-tripul mai mare. După cum se observă pe figură, cei doi clienți nu sunt în exact aceași stare datorită acestei diferențe de latență. Aici intervine procesul de reconciliere care presupune resimularea stărilor clienților în funcție de starea validată de server, proces care asigură sincronizarea eveniemntelor. (14)

##### 3.3. Ordinea de execuție

Pentru realizarea proceselor descrise anterior, Fusion implementeaza o buclă de simulare intercalată cu bucla clasică din cadrul API-ului de scripting Unity. Această buclă oferă metode similare și poate fi împărțită în trei categorii:

* Bucla de resimulare, care se ocupă cu precesul de reconciliere
* Bucla de ’’Forward’’, care se ocupă cu prezicerea următoarelor tick-uri
* Bucla de randare, care este responsabilă cu aspectele ce nu au legătură cu logica jocului; este folosită în principal pentru efecte speciale, animații și interpolarea pozițiilor obiectelor



# Descrierea implementării

## Definirea cerințelor

Aplicația are ca scop principal crearea unei experiențe captivante pentru jucători atât din punct de vedere al gameplay-ului, cât și din punct de vedere vizual și prezintă următoarele funcționalități:

* Sistem de autentificare

Oferă posibilitate de creare de conturi din aplicație și nu permite folosirea acesteia fără logarea cu success pe bază de credențiale. Informațiile de logare vor fi păstrate în cadrul unei baze de date, cu mențiunea că pentru parole se va păstra doar hash-ul acestora.

* Creare unei camera de joc

Utilizatorii pot crea o cameră de joc și realiza setările aferente acesteia: id de conectare, hartă dorită, stil de joc și număr maxim de jucători acceptați. Aplicația va verifica ca id-ul să fie unic la nivel de sistem.

* Alăturarea în cadrul unei camere existente

Utilizatorii se pot conecta la o camera existentă pe baza id-ului acesteia. În cazul introducerii codului corect, aplicația redirectează utilizatorul la fereastra camerei. În caz negativ, aplicația afișează mesaj de eroare corespunzător.

* Prezentarea ecranului specific camerei pre-sesiune de joc

Odată conectați la o camera, utilizatorii pot vedea ceilalți jucători conectați și setările camerei curente, alese la crearea acesteia. Ecranul prezintă posibilitatea de părăsire a camerei sau de începere a sesiunii de joc.

* Două moduri diferite de joc

Aplicația va pune la dispoziție două moduri diferite de joc: unul pe echipe în care jucătorii nu își pot rănii coechipierii și unul în care fiecare jucător e pe cont propriu. Ambele moduri vor reînvia jucătorii după un scurt timp de așteptare, în momentul morții.

* Două hărți de joc diferite

Aplicația va permite utilizatorului să desfășoare sesiunea de joc pe cel puțin două hărți diferite pentru a diminua caracterul de repetabilitate.

* Sistem de mișcare a jucătorului

Aplicația va capta input-ul utilizatorului de la tastatură și mouse și, pe baza acestuia, se va actualiza caracterul jucătorului. Tastele W,A,S,D reprezintă mișcarea în plan orizontal, SPACE reprezintă mișcarea în plan vertical și mișcare mouse-ului actualizează orientarea camerei, care reprezintă perspectiva vizuală a jucătorului.

* Sistem versatil de arme

Va exista o verietate de arme disponibile în cadrul sesiunii de joc:

* armă de distanță mare cu efect instant și număr mic de gloanțe
* armă de distanță mare cu efect instant și număr mare de gloanțe
* armă de distanță mare cu proiectil lent
* armă de distanță mare cu proiectil care ricoșează la coliziuni cu mediul

Pe lângă arme, jucătorul poate folosi două tipuri de grenade: explozive care rănesc inamicii sau fumigene care creează o zonă netransparentă.

* Sistem de obiecte colectabile

Vor exista în anumite locații pe hartă obiecte ce pot fi colectate de jucător pentru a echipa o nouă armă, a reîncărca muniția armelor echipate, a reîncărca stocul de grenade sau a încărca punctele de viață a caracterului.

* Sistem de inventariere a armelor

Jucătorul va păstra maxim două arme simultan asupra sa și va putea schimba, în orice moment, care este cea folosită în momentul prezent prin apăsarea unor taste.

* Animații pentru caracter

Fiecare mișcare a carecturului este însoțită de o animație specifică. Animațiile caracterelor specifice inamicilor sunt vizibile și sincronizate local.

* Sistem de viață a jucătorilor

Jucătorii vor avea un număr de puncte de viață fix la începutul sesiunii de joc. Acesta va scădea în momentul coliziunii cu un proiectil inamic, va crește în momentul colectării unor obiecte specifice și va desemna moartea jucătorului dacă ajunge mai mic decât valoarea 0.

* Afișarea informațiilor esențiale desfășurării sesiunii de joc

În cadrul interfeței grafice din timpul unei sesiuni de joc, pe ecran vor fi afișate: punctele de viață ale jucatorului, arma curentă, muniția acesteia, timpul rămas din rundă și, pentru o durată scurtă de timp, anunțuri în momentul uciderii unui jucător.

* Actualizare setărilor video

Utilizatorii pot seta calitatea video dorită, fiind disponibile minim trei opțiuni diferite pentru a asigura performanța pe dispozitive mai puțin dotate.

* Gestionarea conexiunilor

Serverul păstrează o evidență a tuturor jucătorilor conectați și realizează operații specifice în momentul conectării sau deconectării.

* Asigurarea răspunsului rapid la input-ul utilizatorului

Folosirea unor tehnici care să ascundă sau să minimizeze latența introdusă de rețea și să creeze impresia că sistemul răspunde aproape instant.

* Păstrarea evidenței scorului

Pe timpul unei sesiuni de joc, se va ține evidența scorului fiecărui jucator. Acesta va fi vizibil tuturor în cadrul unei ferestre speciice, activate prin apăsarea unei taste.

* Părăsirea unei sesiuni de joc

Utilizatorul are acces la un meniu sumar pe timpul desfășurării jocului care permite părăsirea acestuia și întoarcerea la meniul principal.

* Terminarea automata a unei sesiuni de joc

La finalizarea unei runde, jucătorii vor fi redirectați automat spre ecranul specifc camerei la care sunt conectați.

* Istoricul rezultatelor

Există posibilitatea de a vizualiza jocurile desfășurate anterior, cât și anumite statistici referitoare la acestea, precum scorul utilizatorului.

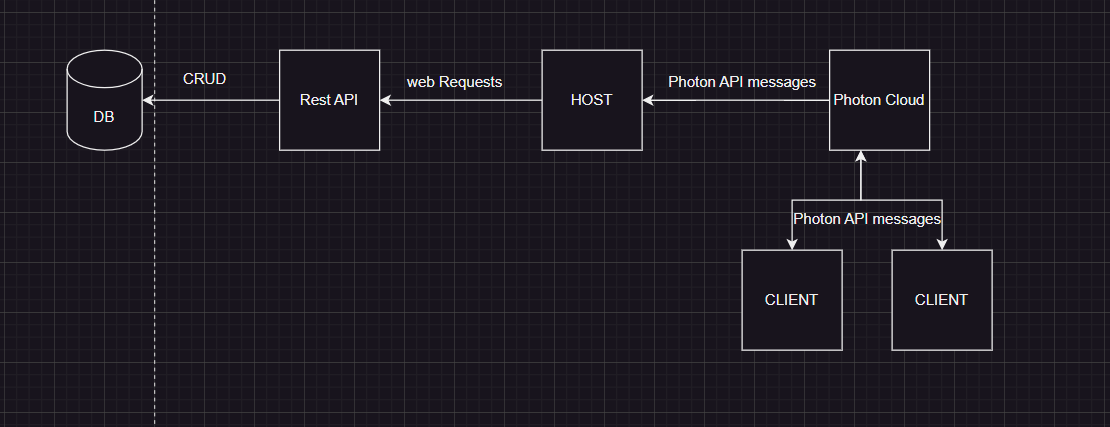
* Folosirea efectelor vizuale

Cu scopul de a îmbogății experiența utilizatorului, fiecare acțiune din joc va avea asociată un efect vizual specific.

## Prezentarea arhitecturii soluției

Aici avem fluxul general al aplicației.

### Prezentare a aplicației integrale



Sistemul este alcătuit din mai multe componente:

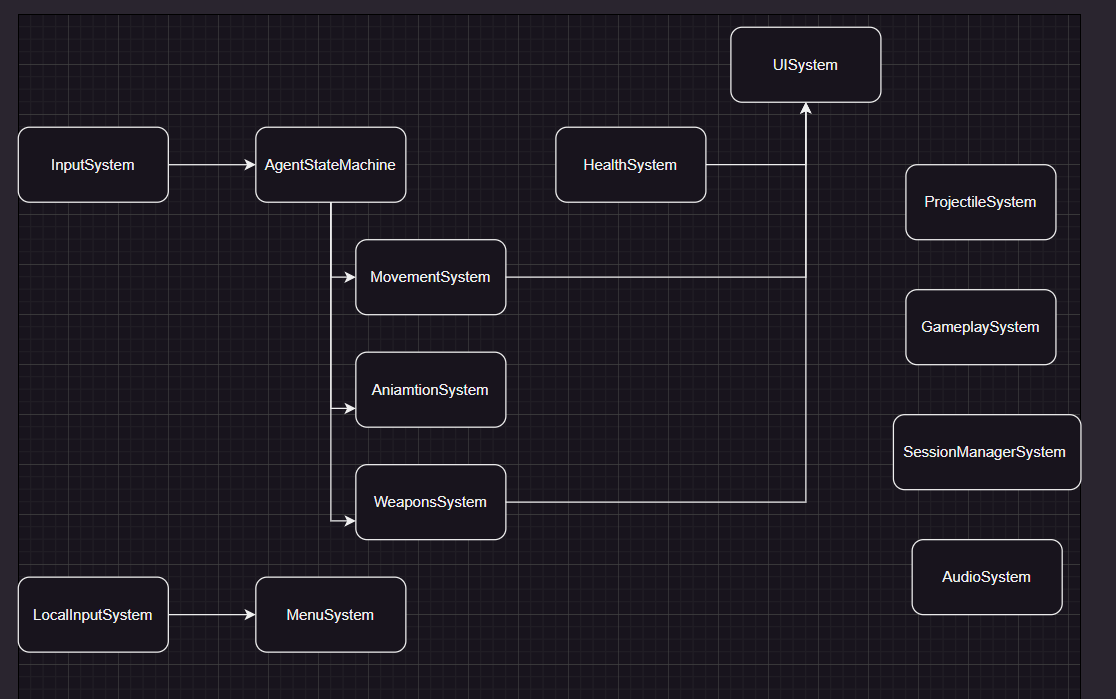
* Client – aplicația desktop pusă la dispoziția utilizatorului
* Host – aceeasi aplicație client, însă adoptă și rolul de server pentru a valida starea jocului
* Photon Cloud – serviciu oferit de Photon pentru a facilita conectarea si comunicarea clienților
* Rest API – aplicație care oferă o interfată de lucru cu baza de date și serviciu de autentificare a utilizatorilor
* DB – baza de date

Clienții comunică între ei folosind API-ul Photon Fusion de transmitere de mesaje, mesaje care sunt redirectate prin intermediul cloud-ului. Aceste mesaje conțin apeluri de evenimente sau snapshot-uri ale stării curente a aplicației din punctul de vedere al aplicației locale.

Host-ul este singurul care comunică cu microserviciul rest pentru a realiza autentificarea și pentru a interoga baza de date prin itnermediul acestuia. Comunicarea se face prin request-uri http, folosind formatul json pentru împachetarea mesajului transmis.

### Prezentarea aplicației Client și Host

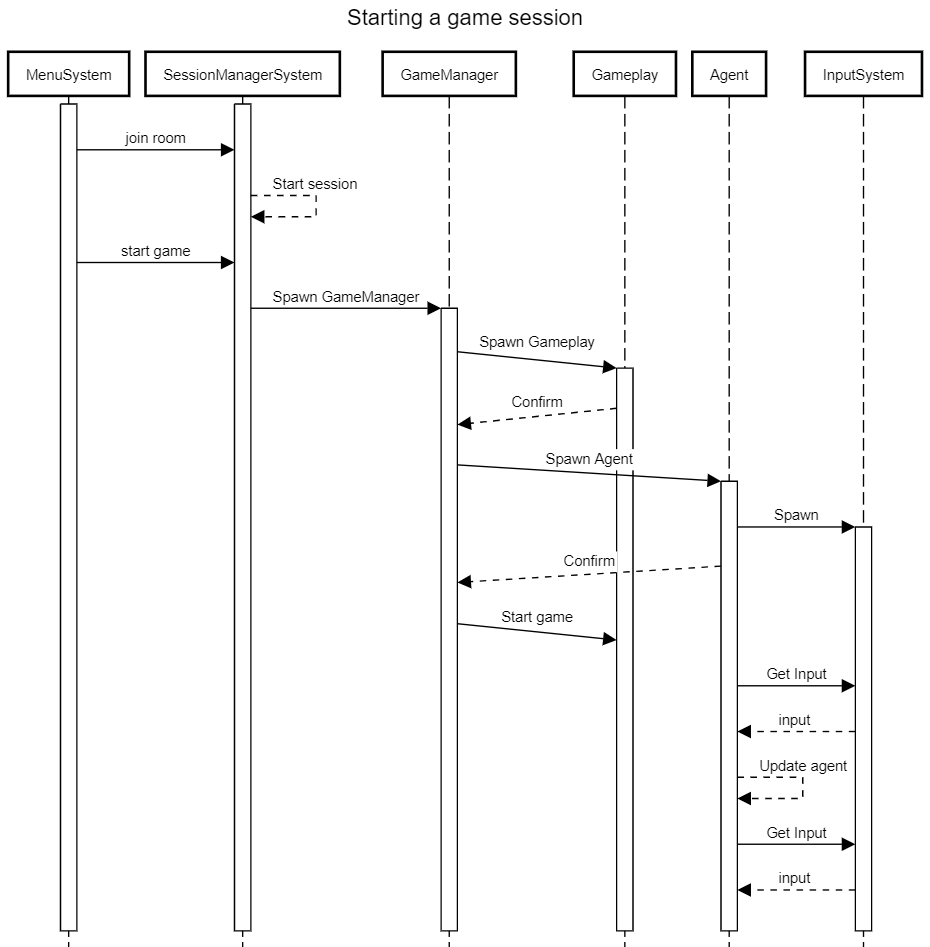
Host-ul realizează aceleași funcții ca aplicația client, însă îndeplinește și rolul de server. Mai multe detalii despre acest lucru în capitolul \*INSERT NUME CAPITOL\*. Arhitectura prezentată in continuare este cea comună ambelor aplicații.



Prezentarea sistemelor din aplicația client:

* LocalInputSystem – captează input-ul care afectează doar clientul local, folosit în general pentru interacțiunile cu sistemul de interfață cu utilizatorul
* MenuSystem – gestionează ordonarea și parcurgerea diferitelor menu-uri prezente în aplicație
* SessionManagerSystem – sistem care se ocupă cu inițializarea, respectiv distrugerea, sesiunii de joc și a tuturor sistemelor sau obiectelor necesare pentru realizarea acesteia
* InputSystem – captează input-ul utilizatorului în timpul desfășurării unei sesiuni de joc, îl înregistrează local și îl împărtășește cu serverul
* AgentStateMachine – actualizează și sincronizează starea curentă a caracterului și, în funcție de aceasta și de input, apelează funcțiile aferente din sistemele subordonate
* MovementSystem – aplică și sincronizează mișcarea caracterului în funcție de input și starea curentă
* AnimationSystem – gestionează animațiile caracterului în funcție de starea curentă
* WeaponSystem – permite echiparea și dezechiparea armelor, gestionează și actualizează armele echipate pe baza inputului și a stării curente
* HealthSystem – sistem event-driven care ține evidența punctelor de viață a agenților și semnalizează celelalte sisteme
* UISystem – actualizează pe ecranul afișat în timpul jocului statusurile agentului controlat de utilizator: puncte de viață, muniție, efecte speciale
* AudioSystem – controlează efectele audio
* ProjectileSystem – actualizează și sincronizează toate proiectilele existente în scenă
* GameplaySystem – ține evidența jucătorilor conectați în sesiunea de joc curentă și gestionează evenimentele care iau loc pe parcursul acesteia

Diagrama următoare arată modul în care sistemele sunt instantiate în cadrul aplicației. Am ales să exclud unele sisteme pentru a păstra claritatea diagramei, însă menționez că sistemul audio este activ pe tot parcursul aplicației, iar restul sistemelor sunt instanțiate în urma instanțierii agentului.



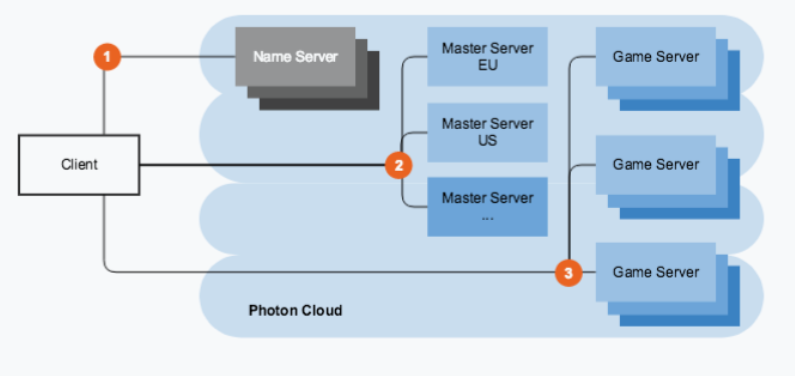
Sursa: reată folosind <https://sequencediagram.org/>

## Descrierea modulelor componente

### Photon Cloud

Este un serviciu cloud care oferă dezvoltatorilor o infrastructură performantă de servere pentru jocuri de tip real-time multiplayer. Serviciul oferă conectivitate la nivel global și facilitează experiența de joc cu o latență mică. Asigurarea acestor beneficii este realizată prin intermediul mai multor servere amplasate în diferite regiuni pe suprafața globului. Fiecare regiune este complet separată de celelalte și este alcătuită din două componente:

* Master Servers – se ocupă cu gestionarea matchmaking-ului
* Game Servers – se ocupă cu gestionarea camerelor în cadrul cărora clienții se conectează (10)



Sursa: https://doc.photonengine.com/fusion/current/connection-and-authentication/regions

În cadrul produsului software dezvoltat, am folosit doar servere de tip ’’Game’’ puse la dispoziție pentru crearera de camere care să permită realizarea sau manipualrea conexiunilor și care să acționeze ca un server relay între clienții conectați.

Camere sunt identificate pe baza unui id unic, care poate fi setat manual de cel care o creează. Clientul care a creat camera preia rolul de Host și are posibilitatea de a realiza setări asupra camerei precum setarea unui număr maxim de conexiuni acceptate sau declararea unor filtre pe baza cărora conexiunile să fie acceptate sau refuzate.

Pentru realizarea conexiunii se realizează următorii pași:

1. În momentul pornirii aplicației în modul multiplayer se realizeaza conexiunea la Photon Cloud, care acționeaza ca un server relay.
2. Odată conexiunea la cloud realizată, clientul poate creea o cameră devenind Host sau se poate alătura unei camere deja existente menționând id-ul acesteia, continuând ca și client.
3. Clienții aflați în aceeași cameră vor face parte din aceeași instanță a jocului și pot interschimba mesaje folosind API-ul photon fusion.

### Aplicatia Host și Client

Cele două aplicații sunt identice din punct de vedere al executabilelor. Diferența dintre ele apare în timpul rulării, întrucât aplicația Host va îndeplini atât rolul unui client cât și pe cel al serverului.

În momentul pornirii, aplicația rulează în modul offline. În funcție de alegerile utilizatorului până în momentul începerii sesiunii de joc, aplicația va prelua rolul de host sau rolul de client. Aceste alegeri sunt expuse prin intermediul unei interfețe controlate de sistemul de meniuri.

#### Creearea interfeței

Un joc multiplayer are în centrul atenției utilizatorul și experiența acestuia. Așadar, pentru crearea interfeței am început cu identificarea utilizatorilor țintă și a contextului desfășurării aplicației, abordând un design centrat pe utilizator.

În ceea ce privește contextul general al aplicației, avansul tehnologic din prezent facilitează dezvoltarea simulatoarelor și a aplicaților în timp real folosite într-o varietate de domenii precum medicină pentru pregătirea corpului medical, afaceri pentru interacțiuni la distanță într-o manieră mai personală, educație și altele. Un joc multiplayer de acest gen antreanază interacțiunea între utilizatori, lucrul în echipă, reflexele și modul strategic de gândire.

Utilizatorii potențiali ai aplicației sunt tinerii cu vârstele cuprinse între 12 și 25 de ani care caută o experiență online alături de prieteni sau alături de străini răspândiți pretutindeni în lume.

Astfel, pentru realizarea unei interfețe care să aducă un grad cât mai ridicat al calității interacțiunii utilizatorilor cu produsul software, design-ul interfeței a fost realizat având în minte nevoile utilizatorilor și următoarele principii de design:

* Principul simplității: se evită supraîncărcarea interfeței, preferându-se o abordare simplă pentru afișarea setărilor si statusurilor
* Principiul consistenței: sunt folosite aceleașo fonturi și elemente de desing în întreaga aplicație
* Principiul feedback-ului vizual: sunt folosite animații, sunete, schimbări de dimensiune sau culori pentru a răspunde la fiecare acțiune a utilizatorului
* Principiul ușurinței de utilizare: caracterul intuitiv este asigurat prin folosirea metodelor clasice, folosite în majoritatea jocurilor de acest gen, pentru interacțiunea cu interfața
* Principiul testării: interfața a fost folosită, sub observare, de utilizatori cu grade diferite de experiență pentru a fi adaptată la nevoile acestora

Două sisteme stau la baza implementării interfeței cu utilizatorului: sistemul de meniuri și sistemul de UI din timpul desfășurării unui joc.

#### Sistemul de meniuri

Aplicația prezintă mai multe ecrane de meniu pentru configurarea jocului sau vizualizarea anumitor statistici relevante utilizatorului. Fiecare ecran este reprezentat de un GameObject cu funcționalități specifice, precum:

* Ecranul de logare: primul prezentat utilizatorului, permite logarea sau înregistrarea pe baza unui cont de utilizator
* Ecranul principal: un punct de acces pentru ecranul de inițiere a jocului sau pentru cel de setări
* Ecranul de setări: permite adjustarea volumui sau schimbarea graficelor
* Ecranul de alegere al modului de joc: host sau client
* Ecranul de configurare al camerei: accesibil doar în modul host, folosit pentru setarea camerei ce urmează a fi creată
* Ecranul camerei create: prezintă o listă cu jucătorii conectați și setările configurate de host în momentul creării
* Ecranul de scor: accesibil în timpul desfășurării unui joc sau la finalul acestuia, oferind statistici pentru scorul fiecărui jucător
* Ecranul gol: un meniu transparent, invizibil, activ pe durata desfășurării uni joc

Managerierea acestor ecrane este prin cadrul unui script central care asigură navigarea dintr-un meniu în altul, activarea unui singur ecran la un moment de timp și gestionarea datelor introduse de utilizator în cadrul diferitelor câmpuri prezentate.

#### Sistemul de UI din timpul jocului

Spre deosebire de sistemul de meniuri, acest sistem are durata de viață limitată, egală cu cea a unei sesiuni de joc și nu permite interacționarea activă a utilizatorului deoarece este folosit strict pentru afișarea statusurilor relevante pentru acesta în timpul jocului precum: puncte de viață, muniție, crosshair, anunțuri pentru ucideri, efecte vizuale ca răspuns la atacuri, timp rămas din rundă.



Jocurile video au un caracter dinamic din punct de vedere al elementelor vizuale ce necesită a fi afișate pe ecran la un moment de timp. Numărul și tipul acestora poate varia în cadrul unui singur stil de joc, dar mai ales în momentul schimbării sau chiar adăugării unor noi stiluri.

Aplicația suportă, la momentul prezent, 2 tipuri de joc. În vederea posibilității lărgirii acestui orizont, am implementat un sistem flexibil care să permită personalizarea ecranului din timpul jocului într-un mod cât mai accesibil.

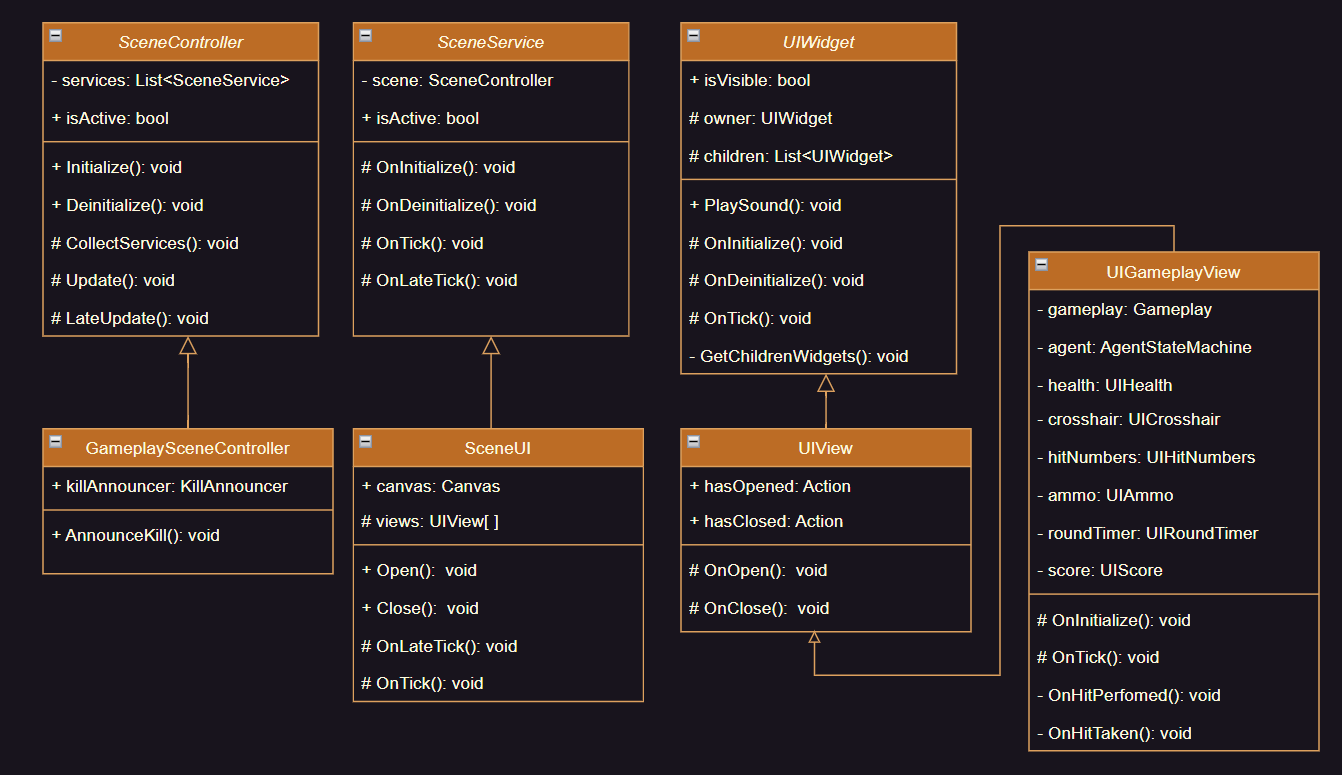


Diagrama de clase a sistemului de UI din timpul jocului

Fiecare nivel al jocului conține un singur GameplaySceneController și un singur UIGameplayView care colectează și actualizează restul elementelor prezente în scenă. Fiecare panou diferit care poate fi activ la un moment de timp în cadrul unui nivel, conține o componentă SceneUI. Iar fiecare element ce se dorește a fi afișat pe acel panou trebuie să conțină o componentă de tipul UIWidget, care permite actualizarea, afișarea sau ascunderea elementului. Astfel, avem un șablon pentru interfața fiecărui nivel, iar în vederea modificării acestuia adăugăm elemente care implementeză clasa UIWidget pentru a fi incluse în sistem.

#### Caracterul

Obiectul de tip caracter este cel mai important obiect din cadrul unui joc deoarece reprezintă puntea prin care utilizatorul pătrunde în lumea jocului și mediul prin care acesta interacționează cu celelalte entități.

Caracterul reprezintă un obiect complex alcătuit dintr-un număr mare de componente:

* NetworkObject – clasă din cadrul API-ul Fusion pentru înregistrarea obiectului în rețea
* Transform – clasă de bază, componentă obligatorie oricărui obiect, folosită pentru manipularea poziției, rotației și a dimensiunii
* AgentStateMachine – componenta care gestionează starea curentă a obiectului, tranziția condiționată în alte stări și acțiunile disponibile în acestea
* BeforeHitboxManagerUpdater – componentă folosită pentru înregistrarea unor metode cu scopul de a fi apelate într-o ordine fixă în cadrul buclei de simulare a aplicației
* AfterHitboxManagerUpdater – componentă cu același scop ca cea menționată anterior
* Animator – componentă specifică tehnologie Unity ce prezintă o interfață pentru controlul sistemului de animații Mecanim
* AnimationController – componentă ce abstractizează lucrul cu componenta Animator, definind stările acestuia si oferind metode de redare a diferitelor animații
* Rig Builder – componentă specifică tehnologie Unity care permite crearea unor animații personalizate
* AgentVisual – componentă ce gestionează încărcarea modelului 3D dorit
* Health – componentă ce gestionează punctele de viață
* Weapons – componentă ce manageriază armele echipate
* PickupItem – componentă ce asigură interacțiunea cu orice obiect consumabil sau echipabil aflat în lumea de joc
* HitboxRoot – componentă specifică API-ului Fusion folosită pentru detectarea coliziunilor
* Rigidbody – componentă specifică tehnologie Unity pentru aplicarea legilor fizicii asupra obiectului

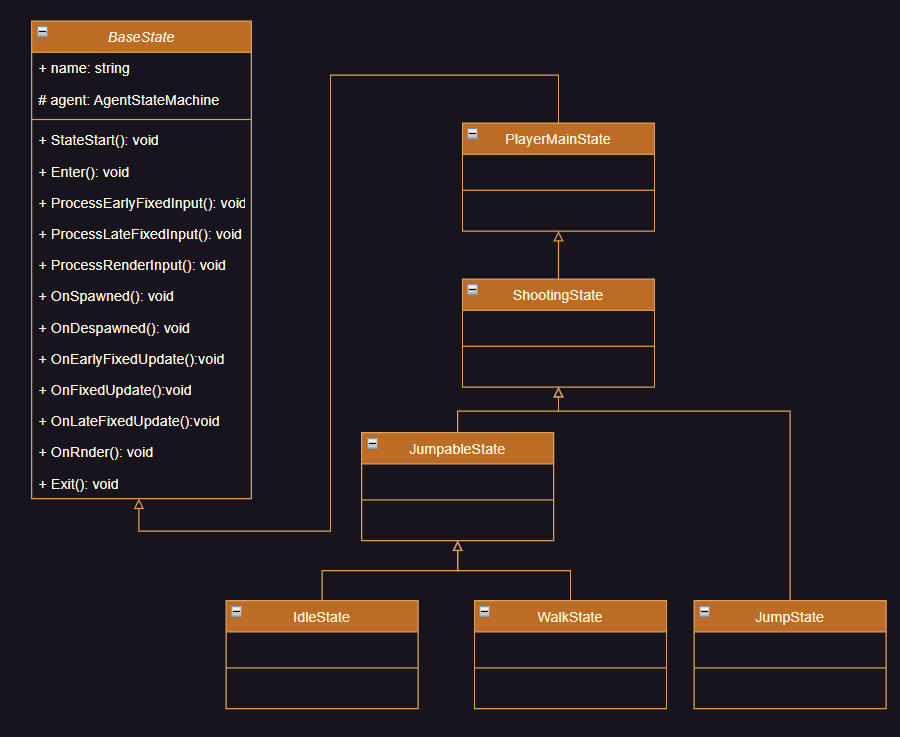
În continuare voi grupa aceste componente după sistemul de apartenență și voi detalia modul în care au fost implementate și în care interacționează între ele.

##### Sistemul de stări

Logica implementării unui caracter poate deveni foarte complexă, iar codul aferent acesteia poate să ajungă repede greu de citit și deloc scalabil din cauza multitudinilor de condiții diferite pe care se bazează. Pentru rezolvarea acestei probleme am ales să folosesc o mașină de stări.

Modelul arhitectural bazat pe o mașină de stări este des întâlnit în cadrul aplicațiilor software și este alcătuit din două componente principale: o mulțime finită de stări, dintre care doar una poate fi activă la un moment de timp, și un set de condiții care reprezintă modalitățile de tranziție între aceste stări.

Astfel, caracterul prezent în joc se poate afla la un moment de timp în una din stările reprezentate de nodurile frunză din diagrama de mai jos, tranziția realizându-se la apăsarea tastelor specifice.



Stările specifice caracterului

Implementarea se bazează pe conceptele fundamentale ale programării orientate pe obiecte: abstractizare și polimorfism. Clasa BaseState este o clasă abstractă și reprezintă baza oricărei alte stări diferite a caracterului. Fiecare clasă copil implementează metodele acesteia și le suprascrie pentru a realiza funcționalitățile în moduri diferite. Variabila de tipul AgentStateMachine reprezintă metoda de interacționare cu restul sistemelor din cadrul aplicației.

##### Punctul central

Componenta AgentStateMachine atașată obiectului de tip caracter reprezintă punctul central al caracterului, întrucât scriptul acesteia conține referințe spre toate celelalte componente și gestionează cum și când scripturile acestora sunt executate.

##### Sistemul de mișcare

Întrucât simularea aplicației este realizată de mai multe ori, în paralel, atât pe fiecare client cât și pe server, este necesară o implementare cu caracter deterministic. Sistemul de simulare a fizicii specific tehnologiei Unity are un caracter dinamic și nu este potrivit pentru sincronizarea obiectelor în rețea. Astfel, pentru realizarea mișcării caracterului, am folosit și extins sistemul oferit de API-ul Fusion care folosește o implementare proprie a mișcării propriu-zise și a verificării coliziunilor.

Sistemul este reprezentat de componenta KCC atașată caracterului și oferă o serie de funcționalități precum:

* Control deplin asupra poziției și rotației
* Interpolare pentru procesul de randare
* Suport pentru forțe externe precum accelerația sau impulsul
* Detecție continuă a coliziunilor
* Suport pentru maparea pe diferite tipuri de teren (15)

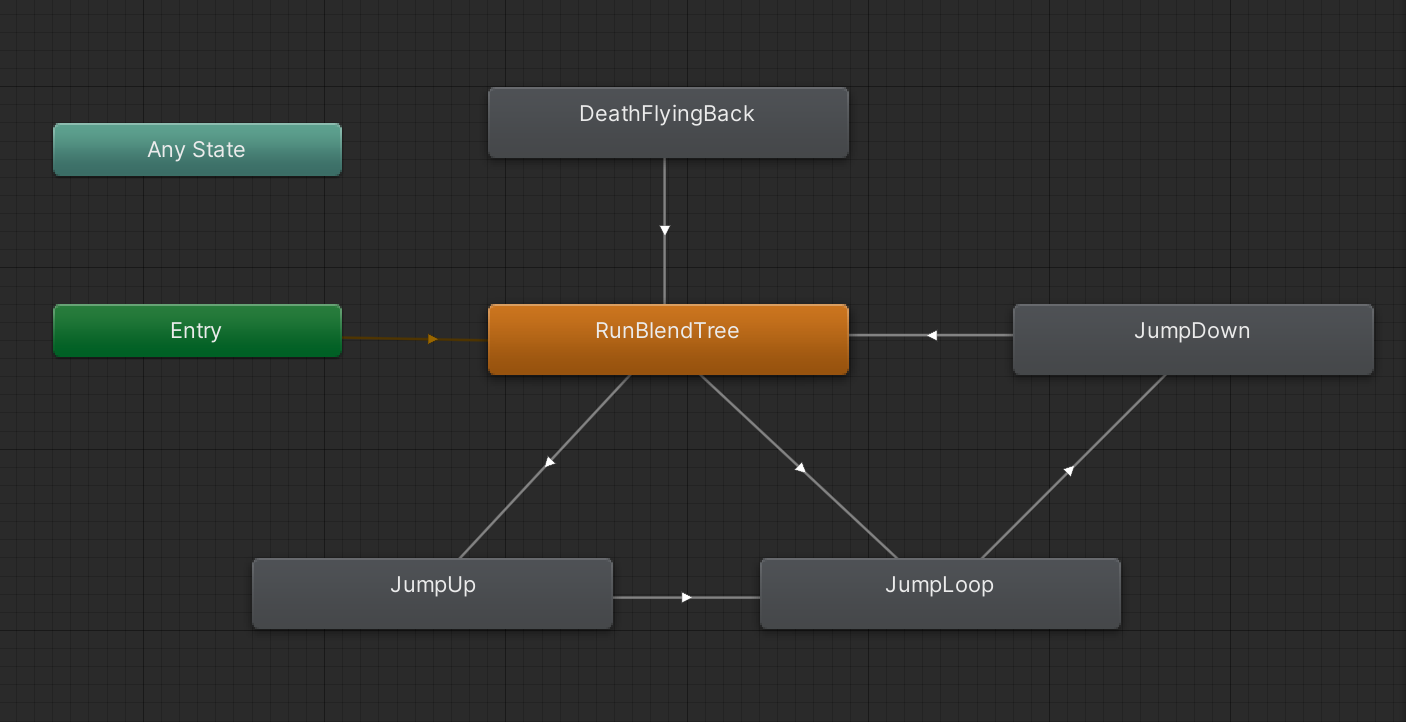
Pe lângă acestea, sistemul oferă structuri de date extensibile și un pipeline personalizabil care permite mai departe dezvoltarea acestuia și care mi-au permis adăugarea modificatorilor de mișcare pentru un caracter aflat în aer, în modul sprint sau în modul furișat.

##### Sistemul de animații

Animațiile sunt secvențe de imagini sau de reprezentări vizuale a unor obiecte care creează iluzia mișcării. Ele contribuie la calitatea experienței vizuale întrucât reprezintă o modalitate de a aduce caracterele și lumea de joc la viață și de o oferi o varietate mai mare în spectrul vizual al aplicației.

Însă animațiile nu sunt importante doar datorită beneficiilor vizuale. Ele sunt folosite ca feedback, comunicând informații despre acțiunile și evenimentele din lumea de joc. Mai mult, în cazul unui joc în rețea, ele sunt folosite pentru a ascunde latența anumitor eveniemente. De exemplu, în cazul aruncării unei grenade, animația de pregătire a acesteia oferă timp suplimentar acțiunii de instanțiere a obiectului. Fără aceasta, ar apărea un timp mort între apăsarea tastei de către utilizator și instanțierea efectivă.

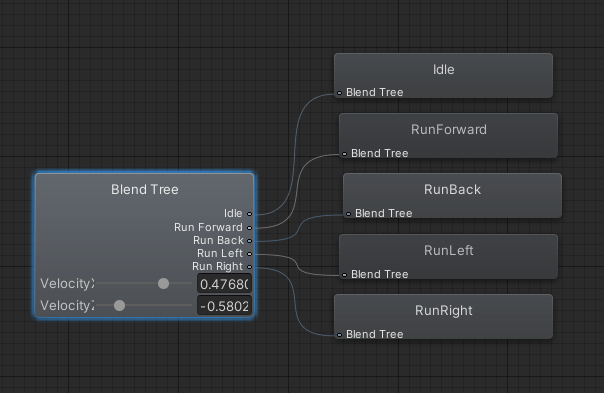
Am folosit soluția integrată în Unity, numită Mecanim, pentru realizarea animațiilor caracterului. Întrucât aceasta funcționează ca o mașină de stări, am definit o stare pentru fiecare animație diferită, iar tranziția este controlată prin intermediul componentei AnimationController atașată caracterului.



Animațiile caracterului

Se observă că animarea unei sărituri este împărțită în trei stări diferite: JumpUp, JumpLoop și JumpDown. Acest lucru se datorează timpului variabil pe care caracterul îl petrece în aer la fiecare săritură.

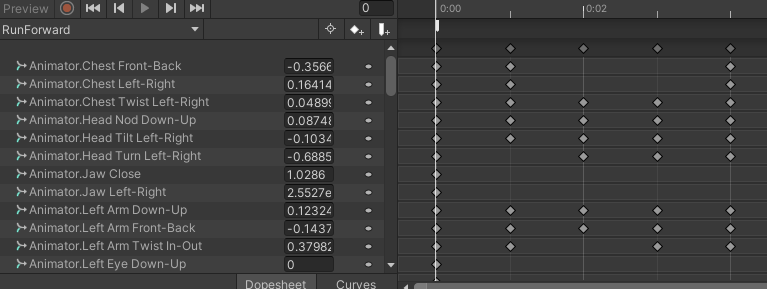
Mai mult, starea RunBlendTree este o stare complexă alcătuită din mai multe stări cu funcționalități similare: mișcarea caracterului în diferite direcții. Animațiile acestor stări sunt combinate și îmbinate împreună pentru a crea impresia unor noi animații care conțin un anumit procent din variantele lor inițiale. De exemplu, prin combinarea animației de mișcare laterală și înainte, se formează animația mișcării pe diagonală.



Run Blend Tree

##### Personalizarea animațiilor

Fiecare animație este reprezentată de un clip ce conține informatii cheie, precum poziția și rotația, despre părțile componente ale obiectului în diferite momente de timp. Aceste informații cheie pot fi modificate prin intermediul unei unelte vizuale integrate în Unity, numite Aniamtion Window. În momentul rulării animației, valorile înregistrate vor fi interpolate și vor creea impresia unei mișcări continue.



Animation Window

Această unealtă este suficientă pentru modificarea unor animații simple. Însă, în cazul unui caracter, este nevoie să realizăm un proces numit ’’animation rigging’’, care presupune generarea unui schelet digital, numit rig, care servește drept bază pentru a anima diferite părți ale corpului.

Scheletul este format dintr-o structură ierarhică de oase sau articulații interconectate care imită structura obiectului. Aceste oase sunt conectate într-un sistem de constrângeri, care permit manipularea și deformarea scheletului în diferite poziții. Mai mult, părțile componente pot fi configurate să se influențeze între ele pentru a oferi un caracter natural mișcării.

Pentru realizarea structurii ierarhice am folosit un pachet suplimentar oferit de Unity numit Animation Rigging. Prin adăugarea unor componente suplimentare caracterului, această unealtă generează scheletul acestuia și oferă scripturi pentru a defini constrângeri și legături între diferite oase.



Scheletul caracterului

Acest proces este necesar pentru adaptarea animațiilor la fiecare armă diferită din joc, întrucât fiecare armă are dimensiuni diferite, locații diferite pentru mâinile caracterului și mișcări diferite pentru încărcarea muniției.

Astfel, odată scheletul realizat, am adăugat constrângeri pentru ca fiecare mână să urmărească continuu locul său stabilit pe arma curentă. Pentru fiecare armă am creat componente care să indice locurile mâinilor și, folosind animation window, am creat animațiile de încărcare a muniției definind o succesiune de poziții clare a părților componente pe durata animației.

Animarea unui caracter de la zero este un proces complex care necesită și îndemânare artistică. Prin procesul descris anterior, doar am modificat animații deja existente preluate de pe site-ul https://www.mixamo.com/#/.

##### Sistemul de viață

Într-un joc multiplayer de tip shooter, metoda principală prin care poți ajunge la victorie este uciderea inamicului. Astfel, un sistem care să gestioneze punctele de viață este crucial, oferind un scop jucătorilor.

În primul rând, sistemul ține evidența punctelor de viață ale fiecărui jucător și asigură sincronizarea acestor valori în rețea. Clienții nu pot modifica în mod direct acest valori, fiecare modificare fiind validată și modificată de server.

În al doilea rând, sistemul este structurat sub forma unui model architectural orientat spre evenimente. Acesta expune două evenimente cheie la care orice alt sistem se poate abona pentru a fi înștiințat în momentul în care un jucător este lovit sau este ucis. Folosind această metodă de comunicare se actualizează statusurile afișate pe ecran sau se initiază diferite efecte vizuale.

##### Sistemul de inventariere

Jocul prezintă o varietate de arme și grenade pe care caracterul le poate echipa sau folosi. Nevoia creării unui stil de joc echilibrat, care să mențină avantajul ce poate fi obținut de jucători la un nivel plăcut pentru adversari, impune implementarea unui set de reguli care să limiteze folosirea acestor echipamente.

Sistemul de inventariere permite jucătorilor să dețină un număr de maxim două arme diferite și maxim câte o grenadă de fiecare tip la un moment de timp. Mai mult, în funcție de impactul acestora, armele sunt grupate pe categorii, iar caraterul nu poate deține decât o singură armă din fiecare categorie. La un moment de timp, jucătorul poate folosi doar una din armele deținute.

Sistemul reprezintă un punct de acces în cadrul sistemului de gestionare al armelor, care va fi descries în capitolul următor.

#### Armele și proiectilele

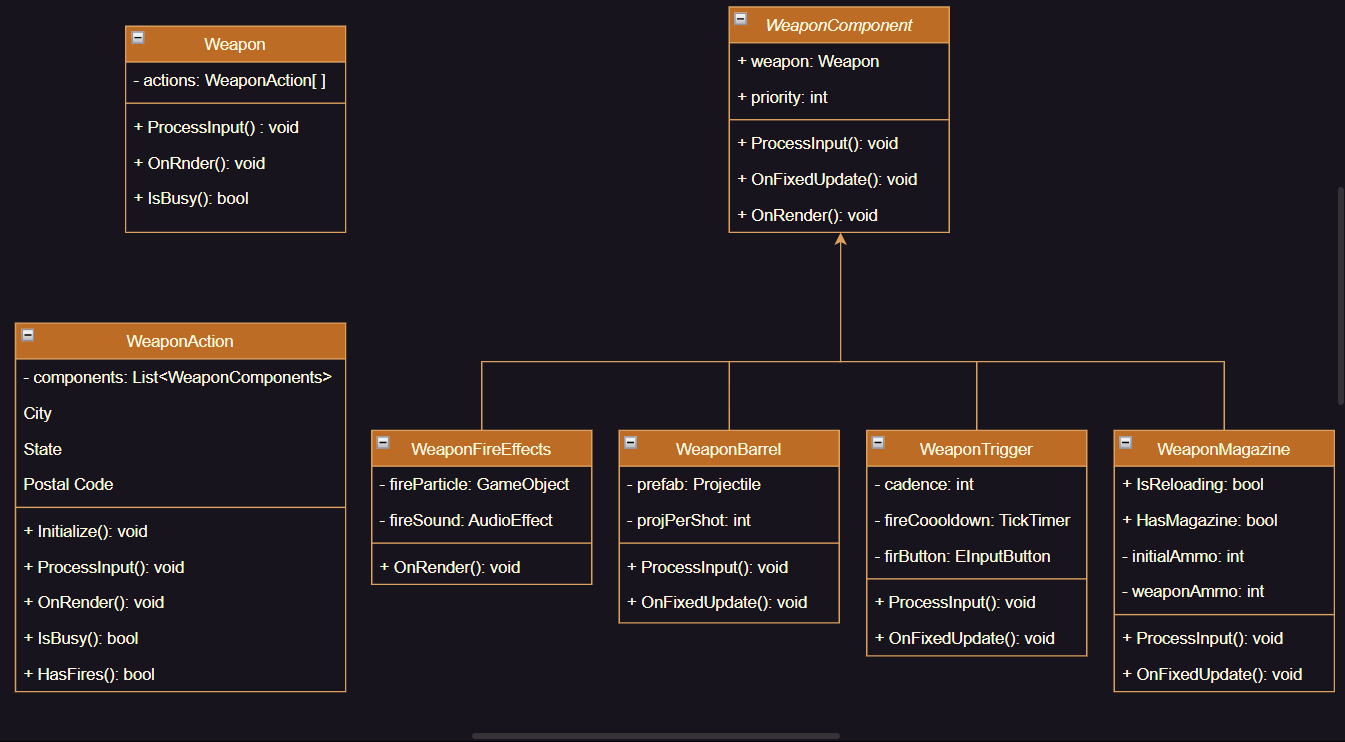
##### Sistemul de arme

Acest sistem a fost implementat având în centrul atenției flexibilitatea, întrucât trebuie să permită implementarea unei plaje variate de arme diferite și personalizarea acestora.

Astfel, fiecare obiect de tip armă trebuie să conțină o singură componentă de tip Weapon, care reprezintă interfața acestui sistem cu restul aplicației, și oricâte sub-componente de tip WeaponAction, care reprezintă acțiunile pe care obiectul le poate realiza. Majortiatea armelor implementate au asociate trei acțiuni: exucutarea focului, lansarea unei grenade vătămătoare și lansarea unei grenade fumigene.

Mai apoi, fiecare acțiune conține sub-componente care implementează efectiv funcționalitățile și permit personalizarea acestora:

* WeaponTrigger – preia input-ul jucătorului; permite configurarea butonului de acționare și cadența armei
* WeaponMagazine – reprezintă încărcătorul armei; permite configurarea numărului de gloanțe și al timpului de încărcare
* WeaponBarrel – realizează execuția focului; permite configurarea numărului de proiectile lansate și tipul proiectilului
* WeaponFireEffects – gestionează efectele vizuale și sonore; permite configurarea efectului de recul al armei, al efectului vizual de la capătul țevii și al sunetului din momentul execuției focului



##### Sistemul de proiectile

Una dintre provocările majore ale implementării produsului a fost realizarea unui sistem care să gestioneze instanțierea si sincronizrea proiectilelor într-un mod eficient. Dificultatea a apărut din cauza numărului mare de obiecte care sunt introduse în rețea într-un interval scurt de timp și care trebuie actualizate continuu în fiecare tick al simulării.

Pentru alegerea celei mai bune metode de implementare am analizat câteva variante:

1. Instanțierea și actualizarea folosind componentele NetworkObject și NetworkTransform oferite de API-ul Fusion

* Pro: este o soluție simplă de implementat care permite și folosirea simulării fizicii integrate în Unity
* Contra: introduce supraâncărcarea rețelei atât cu instanțierea obiectelor cât și cu actualizarea lor la fiecare tick

1. Instanțierea folosind componenta NetworkObject și actualizarea folosind date actualizate manual în rețea

* Pro: actualizarea consumă puțin din lățimea de bandă
* Contra: instanțierea obiectelor introduce supraîncărcarea rețelei

1. Sincronizarea doar a numărului de proiectile instantiate

* Pro: soluție simplă care permite cea mai eficientă folosire a lățimii de bandă și care nu necesită instanțierea obiectelor și la nivel de server
* Contra: traiectorile pentru ceilalți jucătorii nu sunt precise și permite doar proiectile cu effect instant

1. Instanțierea locală a efectelor vizuale e proiectilor și actualizarea manuală a acestora în rețea

* Pro: soluție care permite majoritatea tipurilor de proiectile, folosește eficient banda întrucât instanțierile se fac doar local și nu necesită instanțierea lor și la nivel de sever
* Contra: proiectile nu pot exista maim ult decât obiectul părinte

Astfel, ținând cont că aplicația prezintă un număr variat de proiectile și că am dorit minimizarea traficului în rețea și a procesării la nivel de server, am ales ultima variantă pentru a implementa sistemul de proiectile.

Pe lângă metoda de gestionare, o altă dificultate a fost detectarea coliziunilor. În capitolul în care am explicat cum funcționează procesul de predicție și reconciliere am văzut că niciun client nu se află la exact același tick al simulării ca ceilalți clienți.

Pentru a rezolva această problemă, API-ul Fusion ține evidența pozițiilor trecute a caracterelor. Astfel, în momentul verificării unei coliziuni, serverul realizează verificările folosind locațiile din momentul trecut pentru a oferi clientului ceea ce se numește ’’What you see is what you get’’.

#### Stilurile de joc

Pentru diversificarea și ridicarea calității experienței jucătorilor, aplicația prezintă două moduri diferite de joc:

* Deathmatch – fiecare jucător pe cont propriu, câștigătorul find cel cu scorul cel mai mare la finalul sesiunii de joc
* TeamDeathmatch – echipă contra echipă, câștigînd echipa cu scorul cumulat cel mai mare la finalul sesiuni de joc

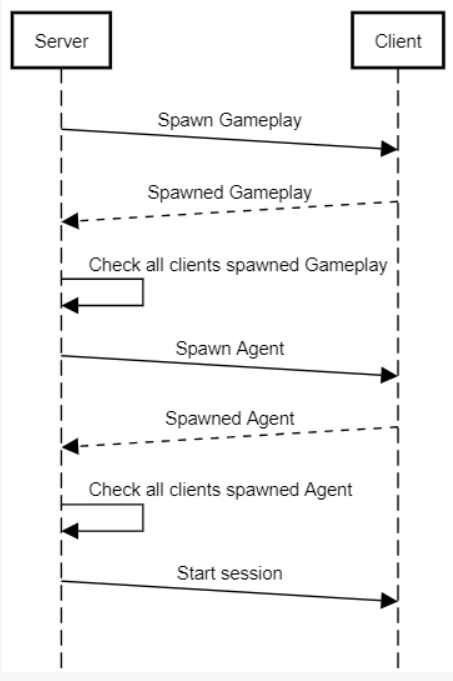
În cazul ambelor moduri, sesiunea de joc are o durată de timp fixă și fiecare caracter este reînviat la fiecare moarte. Scorul este calculat ca un tuplu alcătuit din uciderile și decesurile fiecărui jucător.

Implementarea sistemului se bazează pe o ierarhie simplistă de clase: o interfață care prezintă toate funcționalitățile și câte o clasă copil pentru fiecare stil de joc diferit, care personalizează experiența.

#### Începerea sesiunii de joc

În momentul începerii unei sesiuni de joc sunt necesare inițializarea și configurarea stilului de joc și a caracterelor aferente fiecărui jucător. Ordinea este importantă în realizarea acestor operații și, calitatea conectivității la rețea fiind diferită pentru fiecare client, am implementat un sistem care se bazează pe confirmarea din partea clienților înainte de începerea următoarei operații din partea serverului.

Voi reprezenta aceste schimburi de mesaje printr-o diagramă de secvență.



Odată începută sesiunea de joc, responsabilitatea gestionării jucătorilor conectați este pasată sistemului descris în capitolul anterior.

#### Obiecte interactive

#### 

Pe parcursul desfășurării unei sesiuni de joc, utilizatorii pot găsi în locuri aleatoare pe hartă obiecte care permit interacționarea. Aceste obiecte pot fi consumabile, pentru regenerarea vieții, sau echipabile pentru schimbarea armelor și opținerea grenadelor. Ele sunt folosite pentru complementarea continuității jocului oferită de stilurile implementate care presupun reînvierea jucătorilor.

Detecția interacțiunii dintre obiectele aflate în scenă se realizează cu o tehnică numită raycasting. Aceasta presupune simularea unei linii drepte în direcția indicată de utilizator cu mouse-ul și verificarea coliziunilor cu obiectele dorite.

### Web API

Componenta este reprezentată de o aplicație implementată după modelul REST, având o arhitectură MVC, care oferă utilizatorilor acces la resurse printr-un canal securizat cu ajutorul protocolului HTTPS și un mecanism de autentificare bazat pe standardul JWT. Aplicația expune un set de endpoint-uri pentru realizarea comunicării, iar mesajele transmise sunt în format JSON.

Specific arhitecturii *MVC*, în care cererile pentru resurse sunt supuse unui proces de rutare în urma căruia fiecare request este trimis mai departe către componenta de tip *Controller* potrivită care se ocupă cu procesarea sa și oferirea unui răspuns corespunzător, am implementat următoarele componente de acest tip, cu funcționalități specifice, astfel:

* LoginController – prezintă funcționalități de gestionare a conturilor și face parte din procesul de autentificare, generând token-uri pentru sesiunile utilizatorilor
* MatchesController – pune la dispoziție resurse legate de istoricul meciurilor jucătorilor și rezultatele aferente acestora

#### 

#### 3.3.1. Procesul de autentificare

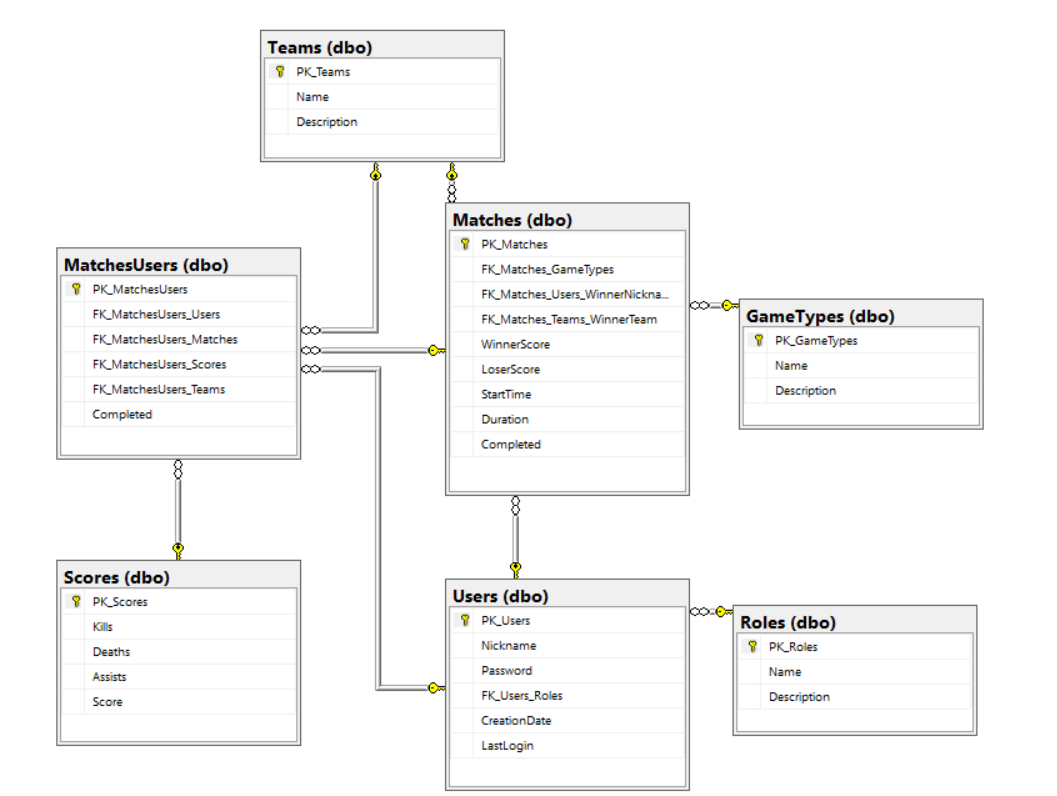
Resursele expuse de către API pot fi accesate doar în urma realizării cu succes a procesului de autentificare. Acesta presupune ca utilizatorul să genereze un request de logare, conținând credențialele, pe baza căruia serverul, în urma validării, va genera un secret unic și îl va trimite utilizatorului încapsulat sub forma unui răspuns HTTP în format JSON. Aplicația client memorează secretul pe toată durata sesiunii de joc pentru a-l adăuga fiecărei cereri HTTP viitoare.

În cazul în care un client nu deține un cont de utilizator, procesul descris mai sus trebuie precedat de procesul de înregistrare. Acesta presupune trimiterea unei cereri HTTP care să conțină credențialele dorite de utilizator. În urma validării de către server și a înserării acestora în baza de date, clientul primeste un răspuns care să confirme reușita procesului.

### Baza de date

Baza de date din cadrul produsului este implementată folosind o soluție oferită de Microsoft de tipul PaaS care oferă suport pentru managementul bazei de date și monitorizarea acesteia, numită Azure SQL Database. Aceasta asigură integritatea și persistența datelor stocate, oferind și posibilitatea introducerii scalabilității automate.

Aplicația Web API este componenta mediatoare între clienți și baza de date, oferind o serie de funcționalități care implementează operații CRUD pentru interogarea și modificarea tabelelor.



## Rezultatele testelor efectuate și interpretarea acestora

# Concluzii

## Sinteza principalelor idei din lucrare

## Direcții pentru continuarea cercetării

# Bibliografie

# Bibliography

1. **Wikipedia. *https://en.wikipedia.org/wiki/Game\_engine.* [Online]**

**2. *https://kevurugames.com/blog/unity-vs-unreal-engine-pros-and-cons/.* [Online]**

**3. *https://www.youtube.com/watch?v=V3UrSEnk5bo&ab\_channel=JasonWeimann.* [Online]**

**4. Unity documentation. *https://docs.unity3d.com/Manual/UsingComponents.html.* [Online]**

**5. Unity Documentation. *https://docs.unity3d.com/Manual/ExecutionOrder.html.* [Online]**

**6. Unity Documentation. *https://docs.unity3d.com/Manual/PhysicsSection.html.* [Online]**

**7. Unity Documentation. *https://docs.unity.cn/es/2021.1/Manual/GraphicsAPIs.html.* [Online]**

**8. Unity Documentation. *https://docs.unity3d.com/Manual/AnimationOverview.html.* [Online]**

**9. Unity Documentation. *https://unity.com/how-to/architect-game-code-scriptable-objects.* [Online]**

**10. Photon. *https://doc.photonengine.com/fusion/current/connection-and-authentication/regions.* [Online]**

**11. Photon Fusion. *https://doc.photonengine.com/fusion/current/getting-started/fusion-intro.* [Online]**

**12. Fusion Documentation. *https://doc.photonengine.com/fusion/current/manual/network-object/simulation-behaviour.* [Online]**

**13. Fusion Documentation. *https://doc.photonengine.com/fusion/current/getting-started/fusion-intro.* [Online]**