UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA „MATEMATICĂ ŞI INFORMATICĂ”

DEPARTAMENTUL “INFORMATICĂ”

TĂBÎICĂ CĂTĂLIN

Proiectarea unui joc 3D RPG in Unity

GAME DESING
TEZĂ DE AN

Șef de departament : _____ Capcelea Titu

(semnătura)

Conducător științific : _____ Crudu Cristi

(semnătura)

Autor : _____

(semnătura)

Chişinău 2025

Cuprins:

Introducerea tezei de an

Pentru teza de an am decis să abordez o temă mai complexă, care să-mi pună în valoare cunoștințele dobândite până în prezent și, în același timp, să reprezinte o provocare personală. Alegerea unui joc de tip **RPG (Role-Playing Game)**, dezvoltat în mediu 3D, presupune o complexitate ridicată atât din punct de vedere al designului, cât și al logicii de programare, ceea ce face tema potrivită pentru o lucrare de final de an.

Comparativ cu un joc 2D, un proiect 3D presupune o gamă largă de mecanici avansate: gestionarea mișcării și rotației în spațiu tridimensional, interacțiuni cu obiecte și NPC-uri, sistem complex de combat, upgrade-uri, inventar, gestionarea resurselor jucătorului (viață, energie, experiență), precum și o poveste coerentă care să încadreze toate aceste elemente într-o lume fantastică.

Jocul pe care îl dezvolt este un **RPG de acțiune**, cu accent pe explorare, progresie a personajului și luptă. Universul jocului este vast și încărcat de elemente de fantezie: magie, monștri, demoni și lumi paralele, toate îmbinate într-o poveste unitară ce oferă jucătorului o experiență imersivă.

Pentru dezvoltarea proiectului am ales **motorul grafic Unity**, datorită familiarității pe care o am cu acesta și a suportului solid pentru dezvoltarea de jocuri 3D. Un alt avantaj major este utilizarea limbajului de programare **C#**, limbaj pe care îl stăpânesc foarte bine și care îmi permite implementarea logicii de joc într-un mod eficient și flexibil.

Alegerea acestei teme vine din dorința de auto-depășire și învățare continuă. Nu am mai realizat un proiect asemănător, însă consider că tocmai această necunoscută este ceea ce mă va ajuta să mă dezvolt atât pe plan profesional, cât și personal. Din experiență știu că evoluez cel mai mult atunci când mă confrunt cu provocări reale, care mă obligă să învăț, să greșesc și să găsesc soluții creative pentru problemele apărute.

I. Fundamentele RPG și Unity3D

1. Introducere în jocurile RPG:

Jocurile RPG sunt cunoscute de mult timp, încă din anii 1970, când erau redate mai ales pe interfață textuală. Mai exact, te jucai doar cu informații primite sub formă de text, precum HP, Power, MP etc.

- În prezent, acest tip de jocuri sunt extrem de dezvoltate, variind de la 2D la 3D, cu modele de personaje și, în special, inamici detaliați. Dacă sărim direct în prezent, jocuri precum *Elden Ring* dispun de modele extrem de bine realizate, cu animații și efecte care oferă o imersiune foarte puternică jucătorului.

- Lumea pe care o creează jocul este extrem de importantă. Cu cât terenul jocului este mai vast și mai variat – de la munți la câmpii și vulcani, de la zone sigure și pașnice la regiuni periculoase, populate de monștri –, cu atât stârnește un interes mai mare pentru fiecare jucător. Aceasta îi oferă dorința de a explora locuri misterioase și necunoscute.

- În general, jocurile RPG includ anumite statistici care reprezintă jucătorul, precum clasa acestuia. De obicei, aceste jocuri îți oferă posibilitatea de a-ți selecta clasa, însă nu este obligatoriu. În funcție de clasa aleasă, anumite statistici ale personajului pot fi mai ridicate sau mai scăzute. De exemplu, dacă alegi un personaj bazat pe magie, un mag, acesta va avea un număr mai mare de puncte pentru magie (MP).

- Acest tip de jocuri oferă, de obicei, o experiență foarte interesantă datorită conținutului extins. Jucătorii cresc în nivel, ceea ce le oferă satisfacție și dorința de a progresa tot mai mult. Printre elementele esențiale ale unui joc RPG se numără:

1. **Progresia personajului** – așa cum am menționat, jucătorul își poate îmbunătăți statisticile prin *level up*.
2. **Luarea deciziilor** – jucătorul poate interacționa și influența povestea jocului prin intermediul NPC-urilor.
3. **Echipamente și abilități** – armele și obiectele pe care le poate folosi jucătorul în luptă, la fel și abilitățile, inclusiv cele magice.
4. **Explorare și quest-uri** – jocul trebuie să ofere o lume vastă și o multitudine de misiuni pe care jucătorul să le descopere pe măsură ce avansează.
5. **Lumea/ Terenul jocului** – un element esențial, așa cum am menționat, deoarece contribuie semnificativ la crearea unui sentiment de mister, explorare și imersiune. Un mediu bine construit oferă jucătorului nu doar spațiu pentru aventură, ci și oportunități de descoperire și interacțiune cu elemente surprinzătoare.

2. Platforma Unity3D, prezentare generală:

Unity este un engine în care poți dezvolta jocuri 2D, cât și 3D. Acest engine este unul dintre cele mai dezvoltate, la fel cum este și Unreal Engine, competitorul său, care este la un nivel mai ridicat. Unity3D îți oferă o gamă largă de posibilități prin care poți crea jocuri 3D.
În ceea ce privește Unity Editor, avem cele mai importante componente, precum:

1. **Scene View** – permite vizualizarea și editarea scenei jocului.
2. **Game View** – afișarea jocului în momentul în care rulează.
3. **Hierarchy** – lista tuturor obiectelor din scenă.
4. **Inspector** – afișarea și editarea proprietăților unui GameObject selectat.
5. **Console** – afișarea erorilor, avertismentelor și mesajelor pentru debug.
   Acestea fiind unele dintre cele mai utilizate.

Scene View îți permite să te deplasezi în spațiul jocului ca editor și să aplici modificări unde ai nevoie.

Game View este perspectiva jucătorului și, implicit, însuși jocul, incluzând meniul (dacă există unul în scenă) și controalele principale, în funcție de tipul jocului. În cazul jocului nostru, o scenă va conține doar meniul principal, iar scena de joc principală va include personajul pe care îl controlăm prin intermediul scripturilor necesare, precum și acțiunile sale, cum ar fi atacul.

Hierarchy conține ierarhia GameObject-urilor din scenă. Un GameObject poate fi orice de la un simplu Transform, care definește doar o poziție tridimensională, până la un jucător principal, NPC, AI, bot, inventar sau meniu.

Inspector – fiecare GameObject are un inspector, iar în acesta se află componente precum

Transform (care definește poziția și rotația GameObject-ului), dar și alte componente precum

BoxCollider, **Rigidbody** sau orice script atașat unui GameObject. Acesta îți oferă posibilitatea de a modifica proprietățile unui GameObject în funcție de necesitățile tale.

Console – prin intermediul acesteia ai posibilitatea de a face debugging. Cu consola poți detecta erorile fatale (din cauza cărora jocul nu poate porni), erorile medii (care adesea sunt doar informații despre lipsa unor date, dar care nu influențează jocul în mod radical) și informațiile de debugging introduse de către dezvoltator pentru a detecta anumite erori ale scripturilor. Spre exemplu, dacă anumite funcții nu sunt apelate, poți folosi Debug.Log("Apelarea funcției") pentru a verifica acest lucru. Dacă funcția este apelată, dar

jocul nu funcționează corect, problema poate fi în logica funcției. De aceea, după fiecare calcul, poți folosi debugging pentru a verifica dacă informațiile calculate sunt corecte.

3. Tehnoligii șii unelte utilizate:

1. **Unity Editor** – Mediul principal de dezvoltare, unde editezi scene, adaugi obiecte și gestionezi jocul.
2. **C#** – Limbajul de programare principal pentru scripting în Unity.
3. **Physics Engine (PhysX)** – Sistemul de coliziuni și fizică folosit pentru mișcare realistă.
4. **Animator & Animation System** – Pentru gestionarea animațiilor personajelor și obiectelor.
5. **NavMesh** – Sistem AI pentru pathfinding (deplasare inteligentă).
6. **URP (Universal Render Pipeline)** – Randare optimizată pentru performanță bună pe diverse platforme.

4. Exemple de jocuri similare și tendințe actuale:

Jocurile RPG, foarte populare, precum seria *The Witcher*, *Dark Souls*, *Elden Ring* și *Skyrim*, au influențat intenția mea pentru acest proiect. Iar inteția mea este de a crea un joc asemănator lumei RPG.

- Ceea ce doresc să implementez este, bineînțeles, un control al jucătorului cât mai complex pentru mișcare, dar și pentru atac. De asemenea, vreau să implementez un sistem de inventar pentru acesta. Jucătorul va avea posibilitatea de a găsi și de a stoca în inventar săbii, poțiuni și incantații/abilități magice. Jucătorul va putea ține două arme și le va putea schimba între ele fără a deschide inventarul. În funcție de arma echipată, aceasta va influența daunele pe care le vor primi inamicii.

- Poțiunile vor avea diverse efecte, precum creșterea vitezei, a punctelor de viață (HP) sau a staminei, iar unele pot oferi mai multe beneficii simultan. Incantațiile vor funcționa într-un mod asemănător cu poțiunile, dar vor necesita o anumită cantitate de mană. Iar în comparație cu poțiunele, incantațiile vor avea și efecte vizuale.

- Totodată, doresc să implementez inamici diferiți, iar lupta dintre jucător și inamic să

fie cât mai unică pentru fiecare tip de adversar. În plus, plănuiesc proiectarea unui teren cât mai variat și a unui sat de orci, conform poveștii stabilite. Pentru a oferi mai multă viață jocului, vreau să adaug și NPC-uri care să reprezinte orcii din sat, însă aceștia se vor deplasa doar între anumite puncte predefinite.

- Un alt aspect important pe care vreau să-l implementez este un UI potrivit pentru un joc RPG. Acesta va oferi jucătorului informațiile necesare despre itemele și obiectele pe care le deține și le poate folosi. De asemenea, UI-ul îi va permite să-și vadă statisticile principale, precum viața (HP), energia (stamina), hrana (food), evidențiind faptul că jucătorul va trebui să mănânce pentru a supraviețui. Toate aceste informații vor fi afișate în partea stângă sus a ecranului, iar în partea stângă jos vor apărea obiectele echipate, cum ar fi sabia, poțiunea și incantația. Fiecare în parte o să dețină câte o iconiță diferită.

- Bineînțeles, va exista și un meniu care face parte din UI-ul jocului. Jucătorul îl va putea deschide apăsând tasta „Escape”, iar acesta va conține trei butoane principale:

1. **Meniul de statistici** – permite jucătorului să-și îmbunătățească statisticile atunci când obține puncte după creșterea unui nivel.
2. **Meniul de echipament** – unde vor fi afișate toate săbiile disponibile. Jucătorul poate echipa maximum două săbii și le poate schimba în timpul luptei fără a accesa inventarul.
3. **Meniul de aventurier** – afișează datele personajului, inclusiv numele, clasa, misiunile active și finalizate, nivelul său ca aventurier și rangul actual.

- Un alt obiectiv este crearea unui teren vast și diversificat pentru lumea jocului. Fiecare zonă va fi distinctă și complexă, iar harta va fi înconjurată de un ocean, oferind senzația unei insule pentru un plus de realism. În plus, intenționez să adaug peșteri în care jucătorul se poate aventura pentru provocări și experiențe unice precum și unele puzzle pentru fiecare peșteră din lume.

- Nu în ultimul rând, intenționez să implementez și sistemul de licență pentru aventurier. Jucătorul își va crea o licență de aventurier conform poveștii, iar pe baza acesteia va putea accepta misiuni în funcție de rangul pe care îl deține. Rangul va crește în funcție de numărul de misiuni completate și de dificultatea acestora.

5. Motivul alegerei temei date

- Motivul pentru care am ales tema **„Joc 3D RPG în Unity”** a fost dorința de dezvoltare personală. Un joc RPG este foarte complex, iar gândirea mea a fost că, dacă reușesc să creez un astfel de joc, mă voi dezvolta în mod corespunzător, îmbunătățindu-mi abilitățile de scriptare, proiectare, implementare și gândire logică.

- La începutul proiectului, nu știam cum să implementez mecanicile de atac ale jucătorului ca un exemplu. Eram însă conștient că, datorită experienței acumulate, voi reuși să găsesc soluții. Cel mai dificil aspect mi s-a părut crearea unui inventar pentru jucător, unde să poată stoca obiecte precum săbii, poțiuni și vrăji. Acest lucru mă îngrijora, deoarece nu mai realizasem astfel de mecanici până atunci și știam că implementarea lor nu este tocmai ușoară. În plus, nu mai scrisesem astfel de scripturi anterior.

- De asemenea, acest gen de jocuri se numără printre preferatele mele, ceea ce a reprezentat un motiv suplimentar pentru alegerea acestei teme.

II. Concept și mecanici de joc

1. Conceptul jocului, poveste, obiective si personaje:

1. Conceptul jocului este că te trezești dintr-o dată într-un loc necunoscut, aflându-te într-o căruță a unui orc dintr-un sat din apropiere. Trecutul nu ți-l poți reaminti decât printr-o singură imagine: ai fugit de un monstru chiar înainte de a ajunge în căruța orcului. Scopul tău este să-ți recapeți memoria, iar pentru a putea face expediții, afli că în satul orcilor te poți înscrie ca aventurier. Astfel, ai posibilitatea de a lua misiuni de la cavernă și de a o explora, cu scopul de a găsi orice obiect magic care te-ar putea ajuta să-ți redescoperi trecutul și identitatea. Între timp, începi să te adaptezi la noul trai și să te atașezi de satul orcilor, devenind eroul lor și apărându-i de monștri.

2. Povestea din spatele jocului este că, în trecut, ai fost un erou antic care s-a luptat cu cel mai mare demon din lumea ta anterioară. Ai avut o echipă cu care, pe parcursul călătoriilor, ați devenit din ce în ce mai puternici. Însă, în momentul în care ați ajuns în castelul demonului, un portal s-a activat pe neașteptate, aruncându-vă în lumi diferite. În noile lumi, ați fost secătuiți de putere, deoarece particulele magice din aer nu sunt la fel ca cele din lumea voastră, astfel că nu mai puteți extrage energie din mediu. Totuși, pe măsură ce te adaptezi, îți dai seama că poți învăța să extragi energie și din această lume. Devii aventurierul de top și ajungi să salvezi satul orcilor. Pe măsură ce învingi fiecare boss, fragmente din memoria ta revin, ca și cum ai primi mici fotografii cu cine ai fost odinioară. Când te confrunți cu ultimul boss al lumii orcilor, o explozie te transportă înapoi în lumea ta veche, readucându-ți toate amintirile. Însă, te întorci exact de unde ai plecat… din castelul demonului, unde urma să aibă loc confruntarea finală. Tot ce a mai rămas din castel sunt ruinele sale și un vulcan gigantic, aflat în erupție exact în momentul revenirii tale. Jocul… continuă în partea a doua.

3. Obiectivele tale în primul joc sunt, să-ți recapeți memoria, să evoluezi în lumea nouă și să aperi orcii. Fragmente din memoria anterioară îți vor fi redate după ce vei învinge fiecare boss din povestea principală a jocului. Odată ce vei doborî și ultimul boss, vei fi readus în lumea anterioară, unde vei avea în continuare misiunea de a-ți găsi vechea echipă.

4. Personajul principal ești tu, un erou antic necunoscut, nici măcar de tine, din cauza memoriei pierdute. Urmează orcul care te-a găsit prăbușit la pământ și te duce în satul său. Acolo îl vei întâlni și pe orcul, liderul aventurierilor, la care vei merge pentru a te înregistra, devenind și tu, la rândul tău, aventurier. Următorii sunt inamicii, fiecare având o poveste în spatele său, dar și cei care te vor ajuta să îți recapeți amintirile din lumea veche.

2. Structura scenelor si mecanicile principale:

2.1 Structura scenelor

Jocul deține 4 scene. **Scena 1**, reprezintă meniul principal, de unde jucătorul poate apăsa butonul de *Play* pentru a începe jocul. De asemenea, există opțiunea *Quit.* Următoarea scenă,

Scena 2, este folosită ca o mică tranziție pentru intrarea în joc. Aici există un scurt cinematic de aproximativ un minut, în care tu, ca jucător, fugi de un dragon care încearcă să te prindă. Pe ultima sută de metri, are loc o explozie în fața ta, după care ecranul se întunecă complet.

Scena 3 începe după ce te-ai trezit în urma exploziei, găsindu-te dintr-o dată în căruța unui orc care te duce în satul său. Pe drum, reușiți să purtați o conversație care oferă detalii importante despre poveste.

Scena 4 este scena principală a jocului, reprezentând însăși lumea în care va avea loc toată acțiunea.

2.2 Mecanicile principale ale jocului

- Controlul jucătorului, atacul cu săbii, inventarul pentru obiecte și statisticile jucătorului.
- Cresterea în nivel inclusiv și a statisticiilor be baza punctelor din level.
- Mecanici pentru cufere care vor detine aiteme/ obiecte precum sabii.
- Incantatii/ Magie pe care jucătorul le va putea lansa.
- NPC-uri care vor fi în mișcare pentru atmosfera satului și a jocului.
- Inamicii și boși cu care te vei lupta ca jucător.

3. Dezvoltarea elementelor de GamePLay

Am dezvoltat următoarele elemente pentru gameplay:

1. GameEngine, 2. PlayerController, 3. StatsPlayer, 4. PlayerInventory, 5. PlayerAttack, 6. MainCanvas, 7. Treasure_Chest, 8. Sword_Manager, 9. Potion_Manager, 10. MagicIncantation, 11. Stone_for_Incantation, 12. EnemyController, 13. NPC_Manager.

1. GameEngine doar setează mouse-ul pe **lock** în centru și îl face invizibil. De asemenea, pornește **PostProcessing-ul**, acesta fiind setat inițial pe **false** pentru ca scena să fie curată în momentul editării și să nu consume resurse inutil.

2. playerController Se ocupă de mișcările jucătorului, sprint, săritură, interacțiunea cu obiectele folosind RaycastHit și animațiile camerei. Jucătorul se poate deplasa înainte-înapoi, stânga-dreapta, iar camera este controlată în spațiu, asemănător unui joc RPG.

1. Mișcarea jucătorului pentru mișcare, se utilizează Input.GetKey(KeyCode.W/A/S/D). Se definesc variabile move_forward și move_right pentru direcție, iar acestea sunt folosite într-un Vector3 move_player:

```
move_player = (transform.forward * move_forward + transform.right * move_right) * speed_player;
```

Mișcarea este aplicată prin CharacterController cu

```
cc.Move(move_player * Time.deltaTime);
```

asigurând o viteză constantă indiferent de FPS.

2. Sprint sprintul este activat prin

```
if (Input.GetKey(KeyCode.LeftShift));
```

crescând speed_player. Se salvează viteza inițială într-o variabilă separată pentru a reveni la normal după sprint.

3. Gravitație și Săritură CharacterController.Move() nu include gravitația, așa că aceasta trebuie implementată manual. Se folosesc:

- float velocityY pentru gravitație,
- float gravity pentru intensitatea gravitației,
- float forceJump pentru forța săriturii.

Gravitația este aplicată astfel:

```
velocityY -= gravity * Time.deltaTime;
move_player.y = velocityY;
```

Săritura se declanșează cu

```
if (Input.GetKeyDown(KeyCode.Space));
velocityY = forceJump;
```

4. Rotirea camerei și a jucătorului se utilizează Input.GetAxis("Mouse X/Y") pentru a modifica rotationX și rotationY.

Jucătorul se rotește astfel:

```
transform.rotation = Quaternion.Euler(0, rotationX, 0);
```

Camera este controlată separat, limitând rotația pe axa Y între -70 și 70 grade:

```
rotationY = Mathf.Clamp(rotationY, -70, 70);
mainCamera.transform.rotation = Quaternion.Euler(rotationY, rotationX, 0);
```

5. Interacțiunea cu obiecte se folosește un Raycast pornind din poziția camerei pentru a detecta obiectele interactive:

```
if (Physics.Raycast(mainCamera.transform.position, mainCamera.transform.rotation *
Vector3.forward, out hit, rangeInteract, layerMask_for_interact))
```

Dacă raza atinge un obiect de tip chestInteract sau objectInteract, se activează un element UI pentru interacțiune.

3. statsPlayer acest script gestionează statisticile jucătorului și datele sale, precum HP, stamina, MP, hrană, XP curent, XP necesar, nivel, puncte, dexteritate și armură.

1. HP reprezintă viața jucătorului. Dacă HP scade la 0 sau mai jos (if(hp_player <= 0)), se va apela o funcție, de exemplu player_die(), care va opri jucătorul și va porni animația morții.

2. Stamina, acțiunile jucătorului cum ar fi sprintul, atacul și săritura, consumă stamina. Dacă jucătorul nu are suficientă stamina, aceasta se regenerează treptat:

```
curent_stamina += Time.deltaTime * add_stamina;
```

Variabila add_stamina determină cât de repede se regenerează stamina.

3. Level Up, Funcția level_up(float xp) este apelată când jucătorul înfrânge un inamic și primește XP. Apoi XP-ul curent al jucătorului (curent_xp) este actualizat cu valoarea primită:

```
curent_xp += xp;
```

Dacă curent_xp depășește necesar_xp, jucătorul va urca la nivelul următor, primește 5 puncte și nivelul crește:

```
points += 5;
level++;
```

XP-ul necesar pentru următorul nivel va fi recalculat, iar procesul continuă într-un (do while) până când XP-ul nu mai este suficient pentru un nou nivel.

4. Primirea de Daune (Take Damage) Funcția takeDMG(float dmg) este apelată atunci când un inamic lovește jucătorul. Daunele primite sunt scăzute din HP-ul jucătorului. Dacă jucătorul blochează lovitura, daunele sunt reduse. Reducerea depinde de mainArrmor și se calculează astfel:

```
damageReduction = mainArrmor / (mainArrmor + 100);
finalDamage = dmg * (1 - damageReduction);
```

Astfel, daunele sunt reduse în funcție de armură.

5. Generarea Sunetelor Funcția

```
generate_voidSound(Transform theParent, AudioClip theClip, float volume)
```

este folosită pentru a reda efecte sonore, cum ar fi atacurile și daunele. Se instanțiază un gameObject care conține un AudioSource, se setează clipul sonor și volumul, iar după 2 secunde, obiectul este distrus pentru a elibera memoria:

```
Destroy(gameObject, 2f);
```

4. playerInvetory acest script gestionează itemele jucătorului, incluzând săbiile obținute, poțiunile și incantațiile.

1. Gestionarea Săbiilor și Poțiunilor, când jucătorul obține o sabie sau o poțiune, acestea sunt adăugate într-o listă specifică: swordManager, potionManager sau incantationManager. Obiectele sunt adăugate în liste prin funția

```
addItem(GameObject theItem);
```

Funcția dată verifică dacă obiectul primit este un item de tip poțiune sau sabie. În dependență de itemul primit, îl va adăuga la lista corespunzătoare de săbii sau poțiuni.

2. Array pentru Săbii, scriptul include un array de tip swordManager cu două elemente, care reprezintă săbiile pe care jucătorul le poate echipa rapid fără a accesa inventarul. Dacă un slot este liber (de exemplu, index 0 sau index 1), sabia corespunzătoare va fi echipată. În cazul în care ambele sloturi sunt ocupate, obiectele anterioare vor fi înlocuite. Astfel, pe index 0 se va rescrie sabie nouă și, dacă este necesar, și pe index 1.

5. playerAttack acest script se ocupă de efectuarea atacurilor și blocurilor jucătorului. Pe lângă atacurile principale, scriptul gestionează și blocurile efectuate de jucător pentru a reduce daunele primite de la inamic.

1. Blocul jucătorului. Jucătorul poate bloca atacurile inamicilor apăsând butonul drept al mouse-ului

```
(if (Input.GetMouseButton(1)));
```

Când jucătorul ține blocul timp de câteva milisecunde, variabila publică mainArrmor din statPlayer va fi setată în funcție de swordArrmor din swordManager, înmulțită cu 10. După acest timp, mainArrmor va reveni la valoarea swordArrmor fără amplificare. Astfel, dacă blocul este efectuat corect, daunele primite de la inamic vor fi semnificativ reduse.

2. Atacurile principale. Atacurile principale sunt realizate prin apăsarea butonului stâng al mouse-ului:

```
if(Input.GetMouseButton(0));
```

Acestea se efectuează într-un combo de 3 lovituri. Pentru a sincroniza atacurile, se folosește variabila cooldown_attack de tip float, care controlează timpul dintre fiecare atac. Dacă timpul de cooldown este complet, variabila se resetează și combo-ul începe din nou cu primul atac.

3. Consum de Stamina. La fiecare atac, scriptul accesează stats_player și, din lista all_curent_stats, extrage valoarea staminei, care va fi redusă în funcție de atacul efectuat:

- Primul atac scade 10 din stamina.
- Al doilea atac scade 15 din stamina.
- Al treilea atac scade 20 din stamina.

Înainte de fiecare atac, se verifică dacă jucătorul are suficientă stamina pentru a-l executa.

6. mainCanvast Acest script permite afișarea a peste 15 date ale caracterului, inclusiv stamina, daune, mana, armura principală (mainArrmor), armura multiplayer (multiplayerArrmor) și alte informații despre jucător. Aceste date sunt utile pentru a verifica, în timpul jocului, dacă funcțiile și metodele implementate funcționează corect. Meniul de statistici poate fi activat sau dezactivat apăsând tasta F2.

- De asemenea, scriptul include un meniu de pauză al jocului, care se deschide cu tasta „escape”. Meniul de pauză conține două butoane: „Stats” și „Swords”.

Figura nr.2.1

- La deschiderea panoului de statistici, jucătorul poate, dacă deține puncte suficiente, să își îmbunătățească statistici precum HP sau Stamina, prin apăsarea unui buton asociat fiecărei statistici. Iar panoul „Swords” afișează toate sabii pe care jucătorul le deține în inventar.

Figura nr.2.2

- În panoul săbiilor, fiecare sabie are două butoane: unul pentru echipare („Equip”) și unul pentru aruncare („Drop”). Butonul de aruncare este doar decorativ, deoarece săbiile nu pot fi aruncate. În schimb, butonul de echipare funcționează corect.

- Când jucătorul apasă butonul de echipare al unei săbii, aceasta este adăugată în „inventoryPlayer”. Array-ul cu două elemente verifică care slot este liber sau, dacă ambele sunt ocupate, înlocuiește sabia echipată anterior cu cea nou selectată.

7. Treasure_Chest Acest script gestionează interacțiunea jucătorului cu cuferele din joc, care pot conține obiecte precum săbii sau poțiuni.

Figura nr.2.3

Interacțiunea este declanșată prin calculul distanței dintre jucător și cufăr folosind Vector3.Distance. Dacă jucătorul se află suficient de aproape, funcția:

chestInteract(theChest) este apelată, utilizând Raycast pentru a determina obiectul lovit. Interacțiunea se realizează prin apăsarea tastei „E”, ceea ce declanșează setInteract() din scriptul cufărului. Această funcție verifică dacă cufărul a fost deja deschis. Dacă nu, activează animația de deschidere și impune un timp de așteptare înainte de o nouă interacțiune. Odată deschis, meniul cu obiectele din interior apare pe ecran. Dacă jucătorul interacționează din nou, funcția giveItem_void() transferă obiectul către inventarul jucătorului și realizează următoarele acțiuni:

1. Elimină obiectul din cufăr.
2. Șterge iconița obiectului din meniul cufărului.
3. Adaugă obiectul în inventarul jucătorului (playerInventory.addItem(theItem)).
4. Creează un efect vizual temporar care indică primirea obiectului.
5. Iconița dispare după două secunde.
6. Se generează un efect sonor (generate_VoidSound).

Dacă jucătorul încearcă să interacționeze din nou după ce obiectul a fost luat, observă că “theItem” nu mai există și interacțiunea este anulată.

8. Sword_Manager Scriptul este atașat pe prefaburile care reprezintă o sabie. Este utilizat pentru a prelua datele sabiei, care sunt:

1. string name_item – folosit în momentul în care jucătorul preia un item din cufăr. Efectul generat în timpul acțiunii, reprezentat de o iconiță/imagine în dreapta ecranului, va conține un text care va fi înlocuit cu denumirea sabiei.

2. float dmg – gestionează puterea atacului jucătorului, determinând forța acestuia atunci când echipează o sabie.

3. float protection_for_block – reprezintă protecția jucătorului atunci când acesta blochează un atac prin apăsarea butonului dreapta al mouse-ului.

4. BoxCollider – activat în momentul efectuării unui atac și dezactivat imediat după acesta.

5. Transform position_for_expertBlock – definește poziția pe sabie unde se generează un efect de particule atunci când jucătorul reușește să efectueze un blocaj perfect.

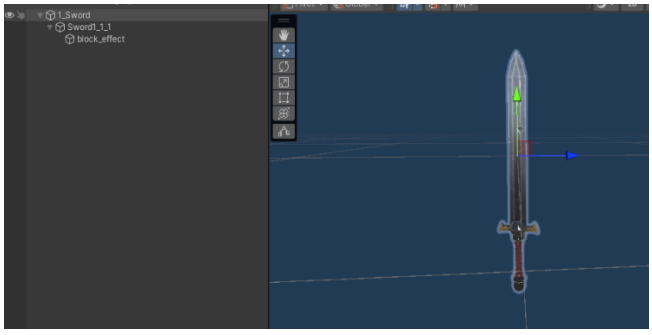

Figura nr.2.4

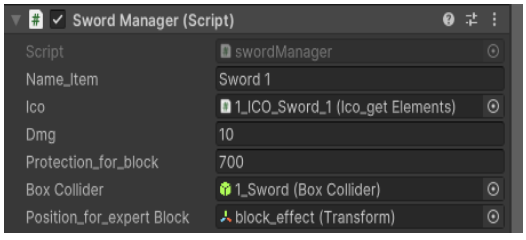

Figura nr.2.5

9. Potion_Manager Acest script este atașat pe prefaburile care reprezintă poțiuni și este utilizat pentru preluarea datelor oferite de acestea:

1. name_tem – numele poțiunii.
2. float timer_for_effect – durata de timp pentru care efectul poțiunii va fi activ.
3. float dmg, hp, stamina, speed – influențează creșterea puterii jucătorului, a HP-ului, staminei sau vitezei.

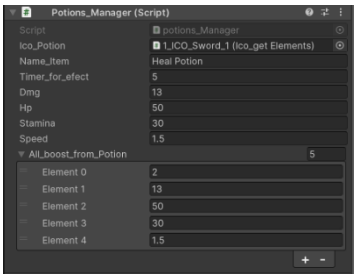

Figura nr.2.6

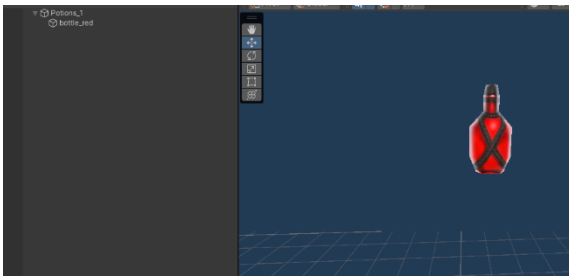

Figura nr.2.7

- Scriptul conține o funcție numită get_all_bonusPotion(), care preia efectele poțiunii consumate de jucător.

- Funcția este apelată prin intermediul scriptului statsPlayer atunci când jucătorul apasă butonul „Q” de la tastatură. De asemenea, cât timp efectul unei poțiuni este încă activ, jucătorul nu poate consuma o altă poțiune simultan.

- Toate datele preluate de la poțiune prin funcția get_all_bonusPotion() vor fi adăugate la statisticile jucătorului.

10. MagicIncantation Acest script este foarte asemănător cu cel din *Potion_Manager*, cu excepția faptului că pentru a efectua o incantație este necesară mana:

```
Public float MP_necesary;
```

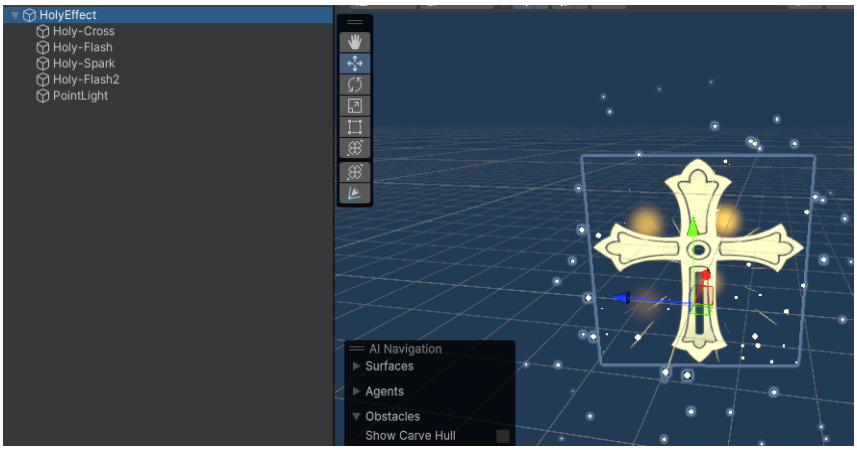

Figura nr.2.8

Generarea unei incantații se face prin apăsarea butonului „LeftControler”.

```
if (Input.GetKeyDown(KeyCode.LeftControl))
```

O dată apăsând acest buton, se verifică incantația care este echipată în acel moment, iar prin accesarea scriptului *MagicIncantation* se extrage variabila publică *float MP_necesary* și se verifică dacă jucătorul deține suficientă mana pentru a efectua incantația respectivă:

```
if (statsPlayer.current_MP >= magicIncantation_script.MP_necesary)
```

Dacă jucătorul are suficientă mana, atunci va fi instantiată incantația echipată, care va reprezenta particulele pentru efectele vizuale.

- Apoi, din *statsPlayer.current_MP* se scade *magicIncantation_script.MP_necesary* pentru a reflecta consumul de mana în urma utilizării incantației.

11. Stone_for_Incantation Scriptul dat este foarte asemănător cu scriptul Treasure_Chest, doar că stone_for_incantation este folosit pentru preluarea incantațiilor pe care le vei găsi pe diverse statui la care este atașat acest script.

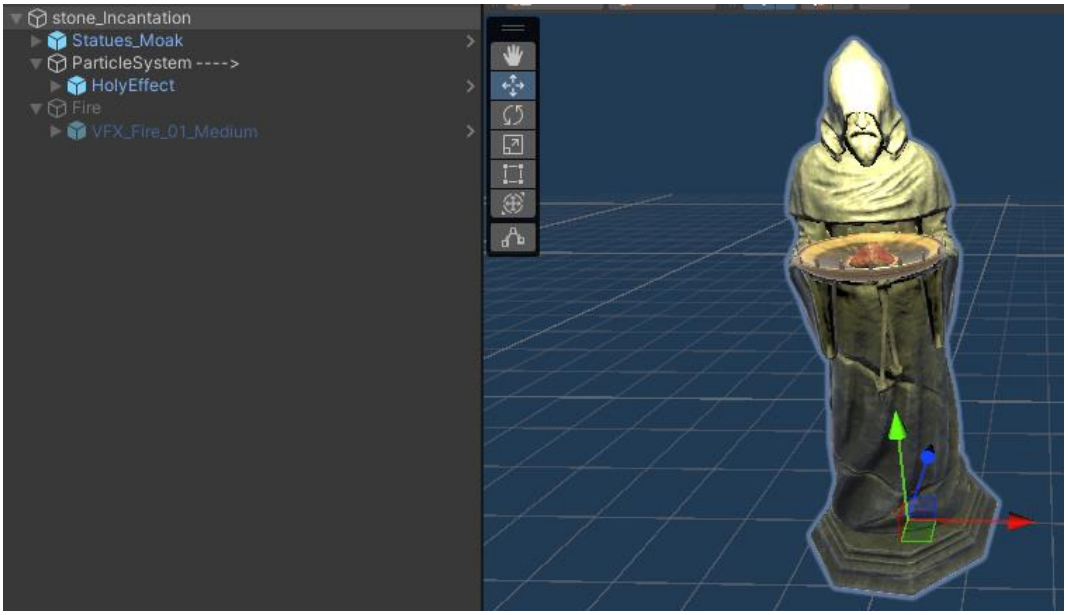

Figura nr.2.9

La scriptul dat există o funcție la fel set_interact(), doar că aceasta nu mai include animațiile de „open”, pentru că este vorba doar despre o statuie. De asemenea, nu are un meniu ca în cazul cufărilor, ci doar în momentul apelării funcției set_interact(), aceasta va accesa și va apela funcția

```
playerInventory_script.addItem(theItem).
```

- După apelarea acestei funcții, se va verifica din nou itemul primit, iar dacă este de tipul magicIncantation, va fi adăugat în lista de incantații a jucătorului.

12. EnemyController Prin intermediul acestui script sunt controlați toți inamicii. Aici a fost folosită una dintre uneltele cele mai importante și utile, și anume **NavMeshAgent**. Această unealtă poate fi descărcată din **Package Manager** în Unity.

- Așa cum am menționat mai sus, **NavMesh** este folosit pentru deplasare. Acesta creează zona sau terenul pe care este posibil să te deplasezi, iar AI-urile care utilizează această componentă pot fi folosite foarte ușor.

De ce este foarte simplu să folosești această unealtă? Pentru că:

- Creează automat terenul pentru deplasare printr-un singur click.
- Și pentru a muta un GameObject pe acest teren, se folosește o linie de cod foarte simplă: ai_nav_mesh_agent.SetDestination(thePlayer.transform.position);

Aceasta va calcula cea mai scurtă cale și include o mulțime de mecanisme utile, cum ar fi viteza de deplasare, accelerația, rotația și multe altele.

1. *Începem cu calcularea distanței dintre jucător și inamic*, și respectiv inamic și zona sa de spawn/home. Scriptul utilizează două variabile de tip float, ambele folosite în metoda Vector3.Distance(). Prima variabilă, denumită distanceByPlayer, calculează distanța dintre inamic și jucător, iar a doua variabilă, distanceByHome, calculează distanța dintre inamic și zona sa de spawn.

2. *Verificăm dacă jucătorul este suficient de aproape de inamic* folosind o condiție

```
if (distanceByPlayer < can_see_thePlayer)
```

și o variabilă denumită can_see_thePlayer. Aceasta va verifica constant dacă inamicul este suficient de aproape, în funcție de raza introdusă în variabila can_see_thePlayer.
În cazul în care jucătorul se află în raza de acțiune a inamicului, acesta se va deplasa către el: ai.SetDestination(thePlayer.transform.position);

3. *Întoarcerea inamicului către zona sa de spawn* se face în cazul în care prima condiție este falsă

```
(distanceByPlayer < can_see_thePlayer).
```

Atunci se va verifica dacă:

```
(distanceByHome > 5);
```

dacă nu este suficient de aproape, inamicul se va deplasa către zona sa:

```
ai.SetDestination(home.transform.position);
```

Variabila „home” reprezintă un GameObject care va fi poziționat fix acolo unde dorim să marcăm casa inamicului.

Animațiile de deplasare sunt controlate cu ajutorul Animatorului atașat la modelul ce reprezintă inamicul. Dacă inamicul este în mișcare, se va activa un bool pentru animația de mers sau alergat

```
animator.SetBool("walk", true).
```

4. *Atacurile inamicului sunt gestionate astfel:* dacă se intră în condiția

```
if (distanceByPlayer < can_see_thePlayer)
```

atunci sunt executate următoarele operațiuni. Inamicul are în jurul său un BoxCollider setat pe isTrigger, care va verifica cu ajutorul condiției:

```
private void OnTriggerStay(Collider other)
{
   if (other.gameObject.CompareTag("Player"))
   {
      the_player_is_in_attack = true;
   }
}
```

- Dacă jucătorul, care are tag-ul "Player", se află în această zonă, se va activa un bool pe true, indicând inamicului că este suficient de aproape pentru a ataca. Dacă inamicul este suficient de aproape, va opri toate animatiile de deplasare iar **NavMeshAgent** va seta viteza acestuia la 0, deoarece, în momentul atacului, inamicul nu se poate deplasa, având deja animația de atac activată:

ai.speed = 0;

- Iar viteza sa va reveni la valoarea inițială în momentul în care:

```
animator.GetBool("walk");
```

este pe true.

- Va activa un trigger din Animator care reprezintă atacul său:

```
animator.SetTrigger("Attack");
```

În momentul în care efectuează un atac, este apelată o metodă care va activa un BoxCollider atașat la arma inamicului. Acest BoxCollider, în momentul în care detectează un jucător, îi va cauza daune în funcție de puterea inamicului, care este transmisă prin acel BoxCollider. Fiecare atac are o metodă pentru cooldown-ul său de activare și dezactivare cu ajutorul IEnumerator:

```
IEnumerator enable_Collider_for_attack(float cooldown)
{
   yield return new WaitForSeconds(cooldown);
   capsule_for_attack.enabled = true;
}
```

- Prin apelarea metodei date este pornit un cronometru, iar când acesta expiră, se trece la următoarea linie de cod. Aceeași logică este creată și pentru dezactivarea BoxCollider-ului.

5. *Primirea de daune din partea jucătorului* se face printr-o funcție denumită getDMG(float dmg). Această funcție este apelată când sabia jucătorului lovește inamicul, și astfel se transmit daunele pe care le va primi inamicul.

6. *Funcția de moarte* este apelată atunci când inamicul primește daune și se verifică dacă viața sa este mai mare decât 0. Dacă inamicul are mai puțin sau egal cu 0 viață, se va apela funcția death(), care va dezactiva toate animațiile, cu excepția celei de moarte/înfrângere.

7. *Experiența jucătorului este gestionată astfel:* jucătorul, având posibilitatea de a face level up pe baza experienței câștigate în urma înfrângerii inamicilor, are la dispoziție o funcție aflată în scriptul inamicului, denumită xp_for_player(). În momentul apelării funcției death(), se va apela și această funcție, care va accesa:

```
statsPlayer_script.updateXP((dmg * multiply) + (save_start_hp_Enemy / multiply)) * 2.5f);
```

Cu ajutorul unor calcule, jucătorul va primi o anumită cantitate de xp, în funcție de puterea inamicului și cantitatea de viață pe care o are.

De asemenea, bara de viață a inamicului este afișată folosind un **Canvas** separat, atașat direct pe inamic, care conține un **Slider**. La începutul jocului, în metoda Start(), sliderul este accesat, iar **maxValue** este setat la valoarea maximă de HP a inamicului (enemy_current_hp). Valoarea actuală (value) este actualizată constant în **Update()**, reflectând astfel în timp real starea de sănătate a inamicului.

Figura nr.2.10

Figura nr.2.11

Figura nr.2.12

13. NPC_Manager Acest script controlează mișcările NPC-urilor din satul principal. De asemenea, folosește **NavMeshAgent** pentru a controla mișcările acestora pe suprafața pământului. NPC-urile au fost introduse pentru a reda mai multă viață jocului, ceea ce este foarte important pentru a crea o experiență mai plăcută pentru orice jucător.

- Fiecare NPC are un număr de poziții prestabilite pe hartă, la care se deplasează pe rând. Când ajunge la ultima poziție, revine la prima și repetă traseul. În momentul în care ajunge la o poziție, are un **cooldown** de așteptare:

```
cooldown_for_stopping = UnityEngine.Random.Range(50, 100);
```

Figura nr.2.13 **Figura nr.2.14**

- Pentru a detecta momentul în care NPC-ul ajunge la o poziție, am folosit **Vector3.Distance**. Atunci când se află suficient de aproape, animația de **walk** este dezactivată, redirecționându-se în **Animator** spre **idle**, iar viteza din **NavMeshAgent** este setată la **0**, astfel încât NPC-ul să stea pe loc. Odată ce **cooldown-ul** ajunge la **0** sau mai jos, NPC-ul va trece la următoarea poziție.

- Pentru gestionarea pozițiilor, care sunt stocate într-o **listă de GameObject**, am folosit o variabilă de tip **int** pentru selectarea indexului. În momentul în care NPC-ul ajunge suficient de aproape de o poziție, indexul crește cu 1:

```
curentIndex++;
```

- Dacă **curentIndex** ajunge la numărul total de obiecte din listă, înseamnă că a depășit intervalul indexului și va fi resetat la **0**.

III. Implimentare și provocari

1. Ce a fost reușit și implimentat

Din planul inițial propus pentru acest proiect, am reușit să implementez în totalitate controlul jucătorului. Așa cum am menționat anterior, scripturile pentru jucător includ **PlayerController**, care gestionează mișcarea. Acesta este destul de complex și bine realizat, utilizând componenta **CharacterController** din Unity. Comparativ cu un **BoxCollider** și **Rigidbody**, această componentă nu doar că le poate înlocui, dar este și mult mai sigură, prevenind probleme precum căderea prin hartă sau traversarea obiectelor solide. Mișcările jucătorului au fost implementate cu succes.

- **StatsPlayer** gestionează statisticile jucătorului, precum mana, mâncarea și stamina. Acesta a fost implementat cu succes și funcționează în joc fără erori.

- **PlayerInventory** a fost de asemenea implementat cu succes, fără blocaje semnificative. Acest script este responsabil pentru gestionarea obiectelor deținute de jucător, cum ar fi armele, săbiile, pațiunile și incantațiile.

- Un alt script important, **PlayerAttack**, a fost realizat cu succes și implementat fără probleme. Funcționalitatea sa este stabilă, permițând jucătorului să execute combo-uri de 2 sau 3 lovituri consecutive.

- În concluzie, controlul jucătorului nu se rezumă doar la mișcare și deplasare, ci cuprinde toate mecanicile esențiale. Am reușit să creez un sistem de control specific jocurilor RPG, datorită mișcărilor fluide, atacurilor și mecanicilor precum inventarul, echiparea săbiilor, consumul de pațiuni și utilizarea incantațiilor. De asemenea, am implementat logica utilizării acestora: atacurile consumă stamina, iar incantațiile consumă mana.

- Am reușit să creez și scripturile necesare pentru săbii, pațiuni și incantații/magie, făcându-le cât mai universale, astfel încât să poată fi multiplicate cu ușurință. Jucătorul poate colecta săbii și pațiuni din cufere, care au fost implementate la fel cu succes. Incantațiile pot fi preluate din anumite statui, similar cufarelor.

- Astfel, avem un jucător complet, capabil de mișcare, atac cu săbii, utilizare de poțiuni și incantații.

- De asemenea, au fost implementați inamicii. Datorită modelelor diverse utilizate (diferite animații, dimensiuni și modele), am reușit să creez un script universal pentru toți inamicii, păstrând totodată unicitatea fiecărei experiențe de luptă.

- Terenul a fost proiectat cu succes, inclusiv satul orcilor și NPC-urile, utilizând **NavMeshAgent**. NPC-urile se deplasează liber în sat, oferind un plus de dinamism și viață jocului.

De asemenea am adăugat și un ocean care înconjoară toată scena din jur in prejur.

2. Ce nu sa reușit

Din păcate, nu am reușit să implementez complet mecanica pentru licența de aventurier. În acest moment, ea reprezintă doar un model al ideii pe care doresc să o dezvolt, însă funcționalitatea sa este inexistentă.

- Am menționat că am reușit să proiectez un teren, însă acesta nu este unul complex. Este un mediu simplu, fără zone distincte, reprezentând un întreg cu aceeași natură. Am intenționat să creez și peșteri, dar m-am limitat, așa cum am menționat, la un teren simplu.

- În ceea ce privește inamicii, am implementat doar doi. Am creat un script universal pentru aceștia, însă, în funcție de fiecare inamic, este necesară proiectarea și configurarea individuală. Cea mai mare provocare este găsirea unor modele adecvate pentru tematica jocului și integrarea lor astfel încât să funcționeze corect ca inamici în joc.

- De asemenea, intenția mea inițială pentru NPC-uri, pe lângă deplasările lor prin sat, a fost să creez o mecanică de interacțiune cu acestea. Totuși, am reușit doar să le programez pentru deplasare.

3. Greutațile întâmpinate și cum le-am rezolvat

Dificultățile întâmpinate au fost mai ales în ceea ce privește grafica și animațiile, mai puțin partea de scripting. Am avut un mic blocaj în crearea terenului folosind tool-urile disponibile din Unity, în special pentru generarea vegetației, cum ar fi copacii, pietrele și iarba. Cel mai mare obstacol a fost modelarea terenului astfel încât să creez zone cât mai diverse. Fiind prima experiență cu un proiect de amploare în acest tool, nu am reușit să creez ceva spectaculos, ci mai degrabă un teren simplu, dar stabil. Cu toate acestea, am învățat să folosesc

mult mai bine aceste instrumente pe parcursul procesului. De asemenea, am întâmpinat dificultăți în plasarea caselor pentru sat, încercând să le poziționez corect pe un teren drept.

- O altă problemă a fost programarea inamicilor, în special animațiile lor. Cei doi inamici importați aveau deja animații predefinite, inclusiv mai multe tipuri de atac, fiecare cu o durată diferită. Am găsit însă o soluție prin care inamicul își activează collider-ul pentru atac exact în momentul în care lovitura din animație se produce și îl dezactivează imediat după. Pentru asta, am creat două liste de tip float, unde fiecare element corespunde unui atac. De exemplu, dacă fiecare listă are două elemente, atunci inamicul are două animații de atac. Atunci când alege un atac la întâmplare, accesează prima listă pentru a determina momentul activării colliderului și a doua listă pentru dezactivare. Astfel, tipurile de atac pot fi modificate în funcție de animațiile fiecărui inamic.

4. Posibilele îmbunătățiri a proiectului

Proiectul are numeroase posibilități de îmbunătățire. De exemplu, adăugarea unor brațe pentru jucător. În prezent, sabia apare pur și simplu în aer, fără a fi ținută în mână. Din punct de vedere vizual, acest aspect este foarte important pentru imersiunea jocului, deoarece un plus de realism oferă o experiență mai plăcută jucătorului.

O altă îmbunătățire majoră ar fi terenul. Așa cum am menționat, cel creat până acum este foarte simplu. Refacerea acestuia sau adăugarea unor elemente care să-l facă mai dinamic – zone variate, peșteri, munți, regiuni periculoase etc.
– ar îmbunătăți considerabil experiența. Cu cât lumea jocului este mai diversificată, cu atât devine mai interesantă și mai tentantă de explorat.

La fel de importantă este și provocarea adusă de inamici. Consider că scriptul actual este destul de bun, însă adăugarea unor modele variate și integrarea lor în acțiuni mai complexe ar îmbogăți luptele și ar oferi o experiență mai captivantă jucătorului.

Codurile Realizate

Controlul principal al jucătorului:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class mainPlayerController : MonoBehaviour
{
    public bool test;

    [SerializeField] LayerMask layerMask_treasure;
    [SerializeField] float speedPlayer, range_Interact;
    private float saveSpeed, sprint, bonusSpeed = 1, dodge_speed = 1;
    [SerializeField] float mouseSensitive;
    [SerializeField] float jumpForce;
    [SerializeField] float gravity;
    public Transform mainCamera;
    public bool isGround = false;
    public bool inCombat = false, inConversation = false;
    private CharacterController cc;
    private statsPlayer statsPlayer_;
    private playerAttack playerAttack_;
    private Vector3 move, dodge_move;
    private float velocityY;
    private int directionY;
    private int directionX;
    private float rotationY;
    private float rotationX;

    public Animator bounsAnimator;

    [SerializeField] GameObject E_Interact;

    private AudioSource walk_sound_effect;

    void Start()
```

```
    {
        walk_sound_effect = GetComponent<AudioSource>();
        walk_sound_effect.Stop();
        saveSpeed = speedPlayer;
        sprint = speedPlayer * 1.5f;
        cc = GetComponent<CharacterController>();
        statsPlayer_ = GetComponent<statsPlayer>();
        playerAttack_ = GetComponent<playerAttack>();
    }

    public void updateSpeed_drinkPotion_for_Speed(float addSpeed)
    {
        bonusSpeed = addSpeed;
    }
    private bool set_ico_for_interact = false;
    void Update()
    {
        RaycastHit hit;
        if (Physics.Raycast(mainCamera.position, mainCamera.rotation * Vector3.forward, out
hit, range_Interact, layerMask_treasure))
        {
            if (Input.GetKeyDown(KeyCode.E))
            {
                GameObject gameObject_interact = hit.collider.gameObject;
                if (gameObject_interact.CompareTag("chest"))
                {
                    chestInteract(gameObject_interact);
                }
            }
            if (!set_ico_for_interact)
            {
                E_Interact.SetActive(true);
                set_ico_for_interact = true;
            }
        }
        else
        {
            if (set_ico_for_interact)
            {
                E_Interact.SetActive(false);
                set_ico_for_interact = false;
            }
        }

        if ((directionX != 0 || directionY != 0))
        {
            if (speedPlayer == sprint)
            {
                if (inCombat && !test)
                {
                    statsPlayer_.all_curentStats[1] -= sprint * Time.deltaTime;
                    statsPlayer_.staminaRegen = false;
```

```
            statsPlayer_.cooldwonStamina = statsPlayer_.save_cooldwonStamina;
        }
        bounsAnimator.SetBool("walk", true);
        bounsAnimator.speed = 3f;
        if (isGround)
        {
            if (!walk_sound_effect.isPlaying)
            {
                walk_sound_effect.Play();
            }
            walk_sound_effect.pitch = 1.5f;
        } else walk_sound_effect.Stop();
    }
    else if (speedPlayer != sprint)
    {
        bounsAnimator.SetBool("walk", true);
        bounsAnimator.speed = 1.5f;
        if (isGround)
        {
            if (!walk_sound_effect.isPlaying)
            {
                walk_sound_effect.Play();
            }
            walk_sound_effect.pitch = 1.05f;
        }
        else walk_sound_effect.Stop();
    }
}
else
{
    bounsAnimator.SetBool("walk", false);
    bounsAnimator.speed = 1;
    walk_sound_effect.Stop();
}

if (Input.GetKey(KeyCode.LeftShift))
{
    statsPlayer_.cooldown_to_regen_the_stamina = 2;
    speedPlayer = sprint;
    if (statsPlayer_.numbers_of_enemy_is_in_battle_with_player > 0)
    {
        statsPlayer_.all_curentStats[1] -= Time.deltaTime * 7f;
    }
}
else
{
    speedPlayer = saveSpeed;
}

if (Input.GetKey(KeyCode.W))
{
    directionY = 1;
```

```
        }
        else if (Input.GetKey(KeyCode.S))
        {
            directionY = -1;
        }
        else
        {
            directionY = 0;
        }
        if (Input.GetKey(KeyCode.D))
        {
            directionX = 1;
        }
        else if (Input.GetKey(KeyCode.A))
        {
            directionX = -1;
        }
        else
        {
            directionX = 0;
        }

        if (isGround && velocityY < 0)
        {
            velocityY = -2f;
        }

        if (Input.GetKeyDown(KeyCode.Space) && isGround && can_do_dodge && directionX
!= 0 && directionY == 0)
        {
            StartCoroutine(dodgeCooldown(directionX));
        }
        else if (Input.GetKeyDown(KeyCode.Space) && isGround)
        {
            velocityY = jumpForce;
        }

        velocityY -= gravity * Time.deltaTime;

        move = (transform.forward * directionY + transform.right * directionX) * speedPlayer;
        move.y = velocityY;

        if (!inConversation)
        {
            if (can_do_dodge)
            {
                cc.Move((move * bonusSpeed) * Time.deltaTime);
            }
            else
            {
                dodge_move = (transform.right * direction_for_dodge) * speedPlayer;
                dodge_move.y = velocityY;
```

```
            cc.Move((dodge_move * dodge_speed) * Time.deltaTime);
        }
    }

    if (!playerAttack_.get_mainInventory_is_Open())
    {
        rotationX += Input.GetAxis("Mouse X") * (mouseSensitive * Time.deltaTime);
        transform.rotation = Quaternion.Euler(0, rotationX, 0);

        rotationY += Input.GetAxis("Mouse Y") * (mouseSensitive * Time.deltaTime);
        rotationY = Mathf.Clamp(rotationY, -70, 70);
        mainCamera.rotation = Quaternion.Euler(rotationY * -1, rotationX, 0);
    }
}
void chestInteract(GameObject chest)
{
    treasure_chest chest_script = chest.GetComponent<treasure_chest>();
    Stone_for_Incantation stone_incantation_script =
chest.GetComponent<Stone_for_Incantation>();
    if (chest_script != null)
    {
        chest_script.setInteract();
    }
    else if (stone_incantation_script != null)
    {
        stone_incantation_script.setInteract();
    }
}

public bool block_with_dodge = false;
private bool can_do_dodge = true;
private int direction_for_dodge;
private float save_mouseSensitive;
IEnumerator dodgeCooldown(int direction)
{
    save_mouseSensitive = mouseSensitive;
    statsPlayer_.generate_voidSound(gameObject.transform, statsPlayer_.dodge_clip, 0.05f);
    can_do_dodge = false;
    mouseSensitive = save_mouseSensitive * 0.2f;
    direction_for_dodge = direction;
    dodge_speed = 4;
    block_with_dodge = true;
    yield return new WaitForSeconds(0.3f);
    block_with_dodge = false;
    dodge_speed = 1;
    mouseSensitive = save_mouseSensitive;
    yield return new WaitForSeconds(0.3f);
    can_do_dodge = true;
}
private void OnDrawGizmosSelected()
{
    Gizmos.color = Color.green;
```

```
        Gizmos.DrawRay(mainCamera.position, mainCamera.rotation * Vector3.forward *
range_Interact);
    }
    public float get_Speed()
    {
        return speedPlayer;
    }

    public float get_speed_sprint_boostSpeed(int iandex)
    {
        switch (iandex)
        {
            case 1:
                return speedPlayer;
            case 2:
                return sprint;
            case 3:
                return bonusSpeed;
            default: return -1;
        }
    }
}
```

Scriptul principal pentru inamici:

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using Unity.Burst.CompilerServices;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;
using UnityEngine.UI;

public class enemyController : MonoBehaviour
{
    [SerializeField] AudioClip get_dmg_clip, sword_attack_clip;
    [SerializeField] Slider hp_Slider;
    public float hp_Enemy = 200;
    [SerializeField] float dmg, speed_for_run, speed_for_return_at_home;
    [SerializeField] float can_see_thePlayer = 10;
    [SerializeField] GameObject locationFX, blood_FX, theCollider_of_Enemy;

    private Animator animator;
    private GameObject thePlayer;
    private NavMeshAgent ai;
    private statsPlayer statsPlayer_script;
    [SerializeField] CapsuleCollider capsuleCollider;

    private Vector3 theHome;
```

```
    private bool the_player_is_in_attack = false;

    [SerializeField] List<float> time_for_startAttack = new List<float>();
    [SerializeField] List<float> time_for_finishAttack = new List<float>();

    [SerializeField] trigger_for_attack trigger_for_attack_script;
    CapsuleCollider capsule_for_attack;

    private bool enemy_stil_see_the_player = false, was_added_enemy_in_list_of_statPlayer =
false;

    private playerAttack playerAttack_;
    private void Start()
    {
        thePlayer = GameObject.FindGameObjectWithTag("Player");
        statsPlayer_script = thePlayer.GetComponent<statsPlayer>();
        animator = GetComponent<Animator>();
        capsule_for_attack = trigger_for_attack_script.GetComponent<CapsuleCollider>();
        capsule_for_attack.enabled = false;

        ai = GetComponent<NavMeshAgent>();
        theHome = transform.position;

        hp_Slider.maxValue = hp_Enemy;
    }

    private void Update()
    {
        hp_Slider.value = hp_Enemy;
        Vector3 direction = hp_Slider.transform.position - thePlayer.transform.position;
        hp_Slider.transform.rotation = Quaternion.LookRotation(direction);

        float distanceByPlayer = Vector3.Distance(transform.position,
thePlayer.transform.position);
        float distanceByHome = Vector3.Distance(transform.position, theHome);

        if (hp_Enemy > 0)
        {
            if (distanceByPlayer < can_see_thePlayer + ((enemy_stil_see_the_player) ? 2 : 0))
            {
                if (!was_added_enemy_in_list_of_statPlayer)
                {
                    statsPlayer_script.numbers_of_enemy_is_in_battle_with_player++;
                    was_added_enemy_in_list_of_statPlayer = true;
                }
                if (!the_player_is_in_attack && !enemy_do_attack_now)
                {
                    giveDMG = true;
                    animator.SetBool("returnHome", false);
                    animator.SetBool("run", true);
                    ai.SetDestination(thePlayer.transform.position);
```

```
            ai.speed = speed_for_run;
        }
        else if (!enemy_do_attack_now)
        {
            animator.SetBool("run", false);
            animator.SetBool("returnHome", false);
            select_attack = UnityEngine.Random.Range(1, time_for_startAttack.Count + 1);
            start_attack();
        }
        else rotation_the_enemy_to_player();
    }
    else if (distanceByHome > 5)
    {
        animator.SetBool("run", false);
        animator.SetBool("returnHome", true);
        ai.SetDestination(theHome);
        ai.speed = speed_for_return_at_home;
    }
    else
    {
        if (was_added_enemy_in_list_of_statPlayer && !animator.GetBool("death"))
        {
            StartCoroutine(substract_the_enemy_from_list_of_statsPlyer());
        }
        animator.SetBool("run", false);
        animator.SetBool("returnHome", false);
        ai.speed = 0;
    }
  }
  else
  {
    death();
    animator.SetBool("run", false);
    animator.SetBool("returnHome", false);
    if (was_added_enemy_in_list_of_statPlayer)
    {
        StartCoroutine(substract_the_enemy_from_list_of_statsPlyer());
    }
  }
}
IEnumerator substract_the_enemy_from_list_of_statsPlyer()
{
  was_added_enemy_in_list_of_statPlayer = false;
  yield return new WaitForSeconds(4);
  statsPlayer_script.numbers_of_enemy_is_in_battle_with_player--;
}
private float rotationSpeed = 10;
private void rotation_the_enemy_to_player()
{
  Vector3 direction = thePlayer.transform.position - transform.position;
  direction.y = 0;
```

```
    if (direction != Vector3.zero)
    {
        Quaternion targetRotation = Quaternion.LookRotation(direction);

        transform.rotation = Quaternion.Slerp(
            transform.rotation,
            targetRotation,
            rotationSpeed * Time.deltaTime
        );
    }
}
private void start_attack()
{
    animator.SetBool("run", false);
    ai.speed = 0;
    enemy_do_attack_now = true;
    animator.SetTrigger("attack" + select_attack);
    StartCoroutine(enable_Collider_for_attack(time_for_startAttack[select_attack - 1]));
    StartCoroutine(disable_Collider_for_attack(time_for_finishAttack[select_attack - 1]));
    StartCoroutine(finish_the_attack(time_for_finishAttack[select_attack - 1]));
}

private bool enemy_do_attack_now = false;
IEnumerator enable_Collider_for_attack(float cooldown)
{
    yield return new WaitForSeconds(cooldown);
    statsPlayer_script.generate_voidSound(transform, sword_attack_clip, 0.5f);
    capsule_for_attack.enabled = true;
}
IEnumerator disable_Collider_for_attack(float cooldown)
{
    yield return new WaitForSeconds(cooldown);
    capsule_for_attack.enabled = false;
}
IEnumerator finish_the_attack(float cooldown)
{
    yield return new WaitForSeconds(cooldown + 1);
    enemy_do_attack_now = false;
    setRandom = false;
}

private bool giveDMG = true;
private int select_attack = 1;
public bool setRandom = false;
public void attack()
{
    statsPlayer_script.takeDMG(dmg * 0.5f);
    giveDMG = true;
}
private void OnTriggerStay(Collider other)
{
    if (other.gameObject.CompareTag("Player"))
```

```
    {
        the_player_is_in_attack = true;
    }
}
private void OnTriggerExit(Collider other)
{
    if (other.gameObject.CompareTag("Player"))
    {
        the_player_is_in_attack = false;
    }
}
public void getDMG(float getDMG)
{
    hp_Enemy -= getDMG;

    statsPlayer_script.generate_voidSound(transform, get_dmg_clip, 0.5f);
    void_blood_effect();
    GameObject thePlayer = GameObject.FindGameObjectWithTag("Player");
    inventoryPlayer inventoryPlayer_ = thePlayer.GetComponent<inventoryPlayer>();
    inventoryPlayer_.sword_echipe_now.boxCollider.enabled = false;
    //Debug.Log("Enemy get dmg from player: " + getDMG);
}
private void void_blood_effect()
{
    GameObject blood = Instantiate(blood_FX);
    blood.transform.SetParent(locationFX.transform);
    blood.transform.localPosition = Vector3.zero;
    blood.transform.localRotation = Quaternion.identity;
    Destroy(blood, 3);
}
public bool theEnemyDefeat = false;
void death()
{
    if (!theEnemyDefeat)
    {
        Destroy(trigger_for_attack_script.gameObject);
        capsuleCollider.enabled = false;
        animator.SetBool("death", true);
        ai.acceleration = 0;
        ai.angularSpeed = 0;
        ai.speed = 0;
        hp_Slider.gameObject.SetActive(false);
        xp_for_player();
        theEnemyDefeat = true;
        ai.enabled = false;
        theCollider_of_Enemy.SetActive(false);
    }
}

private bool giveXP = false;
void xp_for_player()
{
```

```
        if (!giveXP)
        {
            float multimply = 5;
            statsPlayer_script.update_XP(((dmg * multimply) + (hp_Enemy / multimply)) * 2.5f);
            giveXP = true;
        }
    }
}
```

Concluzie

Realizarea acestui proiect m-a ajutat să îmi dezvolt abilitățile de proiectare, gândire și scriptare. Am aplicat cunoștințele acumulate pe parcursul primului și celui de-al doilea an. În final, am reușit să ating toate obiectivele principale ale proiectului.

Am creat o lume mică pentru un joc RPG, iar îmbunătățirile pot fi făcute cu ușurință datorită mecanicilor introduse. De exemplu, adăugarea unui inamic nou, a unei săbii, a unei poțiuni sau a unei incantații este un proces simplu. Terenul a fost partea la care am lucrat cel mai puțin, însă consider că s-a integrat destul de bine în ansamblu.

Ceea ce mi-aș fi dorit cel mai mult să finalizez este sistemul de licență pentru aventurieri, inclusiv mecanica prin care jucătorul poate accepta misiuni în funcție de rangul deținut. Acest sistem ar fi înregistrat fiecare misiune începută și finalizată, calculând nivelul și posibilitatea de avansare în rang.

În ansamblu, sunt foarte mulțumit de rezultatele obținute. Am reușit să creez un produs bine realizat, fără erori majore sau probleme. Am proiectat ceva ce nici nu eram sigur că pot face, ceea ce m-a forțat să mă dezvolt. Am dobândit cunoștințe noi despre diverse aspecte ale Unity-ului. În final, rezultatul acestui proiect mi-a adus satisfacție, chiar dacă există multe posibilități de îmbunătățire. Fiind un joc RPG, are un potențial vast de extindere și diversificare.

Bliliografie

1. https://docs.unity3d.com/Manual/index.html
2. https://docs.unrealengine.com/
3. https://www.gamedev.net/
4. https://www.gamasutra.com/
5. https://docs.blender.org/
6. http://www.cgsociety.org/
7. https://stackoverflow.com/questions/tagged/game-development