

UNIVERSITATEA “TITU MAIORESCU” DIN BUCUREȘTI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

COORDONATOR ȘTIINȚIFIC:
Conf. univ. dr. ing. Mironela Pîrnău

ABSOLVENTĂ:
Anda-Iulia Brătășanu

SESIUNEA IUNIE
2023

UNIVERSITATEA “TITU MAIORESCU” DIN BUCUREȘTI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Implementarea unui joc video 3D dedicat în Unity

COORDONATOR ȘTIINȚIFIC:
Conf. univ. dr. ing. Mironela Pîrnău

ABSOLVENTĂ:
Anda-Iulia Brătășanu

SESIUNEA IUNIE
2023

REFERAT
DE AprecieRE A LUCRĂRII DE LICENȚĂ

TITLU: Implementarea unui joc video 3D dedicat în Unity

ABSOLVENTA: Anda-Iulia Brătășanu

PROFESOR COORDONATOR: Conf. univ. dr. ing. Mironela Pîrnău

Referitor la conținutul lucrării, fac următoarele aprecieri:

A.	Conținutul științific al lucrării	1 2 3 4 5 6 7 8 9 10
B.	Documentarea din literatura de specialitate	1 2 3 4 5 6 7 8 9 10
C.	Contribuția proprie	1 2 3 4 5 6 7 8 9 10
D.	Calitatea exprimării scrise și a redactării lucrării	1 2 3 4 5 6 7 8 9 10
E.	Realizarea aplicației practice	1 2 3 4 5 6 7 8 9 10
	Punctaj total = (A+B+C+D+E)/5	

În concluzie, consider că lucrarea de licență întrunește condițiile pentru a fi susținută în fața comisiei pentru examenul de licență din sesiunea iunie 2023 și o apreciez cu nota_____.

CONDUCĂTOR ȘTIINȚIFIC,

Cuprins:

INTRODUCERE	5
CAPITOLUL 1 – TEHNOLOGII UTILIZATE	7
1.1 – Unity	7
1.2 – Visual Studio	12
CAPITOLUL 2 – DESIGNUL JOCULUI	15
2.1 – Tendințe curente în designul jocurilor.....	15
CAPITOLUL 3 – ETAPELE PROCESULUI DE LUCRU	21
3.1 – Mecanici și implementare	21
3.1.1 – Obiecte și legile fizice.....	21
3.1.2 – Mișcarea caracterului prin interacțiunea cu utilizatorul	24
3.1.3 – Prefabs și rezolvarea bug-urilor cu materialele fizice.....	27
3.1.4 – Inteligența artificială	30
3.1.5 – Sistemul de viață și moarte al caracterului.....	34
3.1.6 – Colectarea obiectelor.....	37
3.2 – Sunete și muzică	41
3.3 – Elementele vizuale	46
3.4 – Tranziția și meniul.....	48
CAPITOLUL 4 – VERIFICAREA APLICAȚIEI.....	55
CAPITOLUL 5 – CONCLUZII	59
BIBLIOGRAFIE.....	60

INTRODUCERE

La fiecare întrebare adresată de-a lungul timpului despre aspirațiile mele profesionale, răspunsul meu a fost întotdeauna același: „Vreau să creez jocuri video”. Fiecare curs la facultate, fiecare proiect și fiecare desen ilustrat au fost pași mici pe drumul spre acest vis. Această lucrare de licență reprezintă punctul culminant al acestei călătorii, în care am reușit să îmbin pasiunea mea pentru jocurile video competitive, interesul pentru artă și aptitudinile de programare pe care le-am dobândit în timpul studiilor.

Lucrarea pe care o prezint se axează pe dezvoltarea unui joc video tridimensional. Inspirat de clasicul joc "Mario", proiectul meu de licență și-a propus să aducă nostalgia acestor jocuri într-un spațiu modern și tridimensional, oferind în același timp un mediu de relaxare pentru jucători. Am explorat diverse tehnici de dezvoltare a jocurilor și am implementat aceste cunoștințe în propria mea creație pentru a aduce noutate în tema aleasă.

Un rol esențial în dezvoltarea jocului l-au avut tehnologiile pe care le-am utilizat. Programul pentru dezvoltarea jocurilor - Unity - și mediul de dezvoltare integrat Visual Studio au fost aliații mei în acest proiect, ajutându-mă să pun în aplicare ideile creative.

Lucrarea detaliază pe larg jocul dezvoltat, evidențiază caracteristicile sale unice și modul în care ele contribuie la experiența totală a jucătorului. Sunt descrise aici premisele jocului și modul în care am modelat universul 3D pentru a le susține. Am descris și etapele de lucru implicate în dezvoltarea jocului, cuprinzând elemente precum mecanica jocului, crearea și aplicarea legilor fizice, implementarea mișcării personajului, inteligența artificială, sunetul și muzica, elementele grafice și tranzițiile de meniu.

Un alt aspect crucial a fost testarea lucrării pentru depistarea erorilor. Consider acest proces esențial pentru a asigura o experiență de joc confortabilă pentru utilizatori și pentru a garanta calitatea lucrării mele.

Lucrarea se încheie cu o secțiune dedicată concluziilor și perspectivelor viitoare. Sper că aceasta nu doar demonstrează competențele tehnice pe care le-am dobândit în timpul studiilor, dar oferă și o imagine a modului în care pasiunea pentru jocuri și artă poate alimenta inovația și creativitatea în domeniul tehnologic.

Mai mult decât atât, prin această lucrare îmi propun să împărtășesc lecțiile pe care le-am învățat în procesul de dezvoltare a jocului. Nu am avut doar provocări tehnice, ci și creative. Încorporarea elementelor de design, dezvoltarea mecanicilor și nivelelor au fost sarcini la fel de importante ca și codarea în sine. Am învățat că dezvoltarea unui joc video nu este doar despre programare și grafică, ci despre crearea unei experiențe imersive pentru jucători.

Prin lucrarea mea, am încercat să păstrez echilibrul dintre elementele clasice ale jocurilor video și inovațiile moderne. Am studiat tendințele curente în designul jocurilor și am încercat să le îmbin cu propriul meu stil artistic. În acest fel, sper să fi creat un joc care să fie nu doar distractiv, dar și plăcut estetic.

O altă provocare a fost să mă asigur că jocul este accesibil și intuitiv pentru toți jucătorii, indiferent de nivelul lor de experiență. Aceasta a necesitat o atenție deosebită la detalii precum interfața cu utilizatorul și feedback-ul. În tot acest proces, am fost inspirată de dorința de a crea un joc care să aducă bucurie și distracție celor care îl joacă.

Proiectul a fost, de asemenea, o ocazie de a aplica tehnici de lucru și de management al timpului, dobândite în timpul studiilor. În timp ce dezvoltarea unui joc poate fi o activitate solitară, am învățat să apelez la alte surse externe pentru a îmbunătăți calitatea muncii mele. Mai mult decât atât, respectarea termenelor limită și prioritizarea sarcinilor sunt abilități pe care le-am aprofundat în acest proces.

Această lucrare este rezultatul unei călătorii pline de provocări și satisfacții. Cu toate acestea, sunt conștientă că este doar un punct de plecare în ceea ce privește cariera mea în dezvoltarea jocurilor video. Am încredere că experiența pe care am acumulat-o până acum și pasiunea mea vor fi un fundament solid pentru proiectele viitoare.



Figura 1 – de la schiță la joc video

CAPITOLUL 1 – TEHNOLOGII UTILIZATE

1.1 – Unity

Descrierea scurtă a programului

Această platformă este folosită pentru crearea jocurilor video, dar și a aplicațiilor, în 2D și 3D. Cu ajutorul ei, creatorii pot să beneficieze de diferite tipuri de sarcini legate de procesul de producție a jocurilor, precum:

- sistem de redare avansat, suport pentru simularea fizicii, animație, gestionarea resurselor și a scenelor;
- un editor intuitiv și ușor de utilizat, care permite construirea și ajustarea lumii jocului, fără a fi nevoie de scrierea de cod în mod constant (cu ajutorul editorului, se pot crea și organiza scene, se pot importa modele și materiale, se pot ajusta setări ale jocului);
- suport pentru programare în limbaje populare precum C# și UnityScript (o variantă modificată de JavaScript) - acest lucru permite creatorilor să scrie cod pentru a controla comportamentul obiectelor și al personajelor în joc, pentru a crea interacțiuni și simulări complexe;
- magazin online numit Asset Store, unde creatorii pot accesa o gamă largă de pachete de resurse, modele 3D, texturi, efecte vizuale, scripturi și alte elemente care pot fi utilizate pentru a-și îmbunătăți jocurile sau aplicațiile;
- comunitate mare și activă de utilizatori și dezvoltatori care oferă suport și împărtășesc resurse și tutoriale utile. Există numeroase forumuri, grupuri de discuții și site-uri online dedicate în care creatorii pot găsi răspunsuri la întrebări și își pot împărtăși cunoștințele;
- funcționalități extinse pentru dezvoltarea de aplicații de realitate virtuală (VR) și augmentată (AR). Aceasta include suport pentru dispozitive VR/AR populare, precum Oculus Rift, HTC Vive, Microsoft HoloLens, dar și altele.

Istoria lui Unity

Unity a fost fondat la Copenhaga de Nicholas Francis, Joachim Ante și David Helgason. Povestea sa a început pe un forum OpenGL în mai 2002, unde Francis a postat un apel pentru colaboratori pe un shader-compiler open source (instrument grafic) pentru populația de nișă a dezvoltatorilor de jocuri bazate pe Mac, ca el. Cel care a răspuns a fost Ante care era pe atunci elev de liceu la Berlin.

Ante a completat concentrarea lui Francis pe grafică și joc cu o abilitate intuitivă pentru arhitectura din spate. Deoarece jocul la care lucra cu o altă echipă nu avea succes, au decis să colaboreze la shader într-un ritm mai redus, în timp ce fiecare își urmărea propriile proiecte de dezvoltare a motorului de joc. Totuși, au hotărât să-și unească forțele în timpul unei întâlniri personale. Într-un sprint pentru a combina bazele de cod ale motoarelor lor, au locuit în apartamentul lui Helgason timp de câteva zile, în timp ce acesta era plecat din oraș. Planul lor era să înființeze un studio de jocuri bazat pe o infrastructură tehnologică puternică, care ar putea fi, de asemenea, licențiată.



Figura 2 – Nicholas Francis, Joachim Ante și David Helgason <https://venturebeat.com/games>

Helgason și Francis au lucrat împreună încă de la liceu, lucrând la diverse proiecte de dezvoltare web și chiar încercări de scurtă durată de a produce filme. Helgason a intrat și a ieșit de la Universitatea din Copenhaga în timp ce lucra ca dezvoltator web independent. A oferit ajutor acolo unde a putut și s-a alăturat cu normă întreagă după câteva luni, vânzând partea sa de acțiuni asociațiilor săi din firma sa de dezvoltare web.

Potrivit lui Ante, Helgason era "bun cu oamenii" și era mai orientat spre afaceri, așa că a luat titlul de CEO după ce trio-ul nu a reușit să găsească o persoană mai experimentată pentru acest rol. (Ar trebui să treacă doi ani înainte ca Ante și Francis să extindă titlul de co-fondator și o sumă corespunzătoare de capitaluri proprii către Helgason.)

Ei au recrutat o distribuție rotativă pentru a-i ajuta gratuit în timp ce încercau o gamă largă de idei. Diversitatea ideilor pe care le-au urmărit a dus la un motor care putea face față unei game largi de cazuri de utilizare. Comercializarea motorului a devenit un scop, ca și lansarea unui joc de succes care să evidențieze motorul ca un mare avantaj; pentru dezvoltatorii indie, nevoia de a reconstrui un motor cu fiecare idee nouă de joc a fost un punct dureros care, dacă ar fi fost rezolvat, ar fi permis rezultate mai creative.

Sușinuți de economiile lor, de o investiție de 25.000 de euro de la tatăl lui Ante și de jobul cu fracțiune de normă a lui Helgason la o cafenea, au continuat timp de trei ani, înregistrând compania în al doilea an (2004) cu numele Over The Edge Entertainment.

Jocul pe care s-au angajat să-l lanseze în primăvara anului 2005, GooBall, a fost "mult prea greu de jucat", spune Ante și nu a câștigat prea multă tracțiune. Recunoscând că erau mai buni la construirea de instrumente și prototipuri de dezvoltare decât jocurile viabile comercial, și-au pariat compania pe obiectivul de a lansa un motor de joc pentru mică comunitate de dezvoltatori bazată pe Mac. Combinând conotațiile de colaborare și compatibilitate încrucișată, ei au numit motorul Unity.

Un aspect mai puțin cunoscut despre Unity este că motorul său grafic folosește o tehnică numită "occlusion culling" pentru a optimiza randarea scenelor 3D. Occlusion cullingul este procesul prin care se determină ce obiecte sau porțiuni ale scenei sunt ascunse de către alte obiecte și nu ar trebui să fie vizibile de la punctul de vedere al camerei.

Unity utilizează algoritmi avansați pentru a calcula și determina vizibilitatea obiectelor și pentru a exclude obiectele ascunse în timpul randării, reducând astfel sarcina pe procesor și accelerând performanța jocului. Aceasta este o tehnică esențială pentru a asigura randarea eficientă a scenelor complexe și pentru a permite rularea jocurilor pe o varietate de dispozitive cu resurse limitate.

În plus, Unity oferă dezvoltatorilor posibilitatea de a configura manual cullingul și de a optimiza procesul prin utilizarea zonelor de culling personalizate, astfel încât să se concentreze doar pe porțiunile relevante ale scenei.



Figura 3 – Logo-ul Unity

Descarcarea si instalarea programului

Pentru a instala Unity, am accesat site-ul oficial de la adresa <https://unity.com> și am apăsător pe butonul „See plans and pricing” / „Vezi planuri și prețuri” - din colțul din dreapta sus al meniului principal (Figura 4).

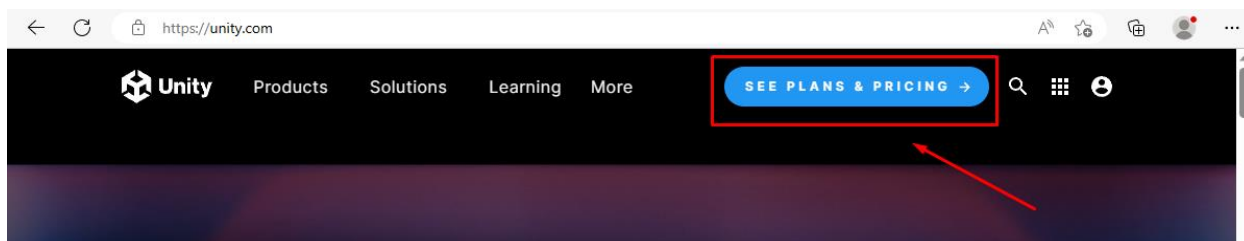


Figura 4 – Meniul Principal

Apoi, după ce am fost redirecționată către o nouă pagină, am ales ales planul personal gratuit (Figura 5).

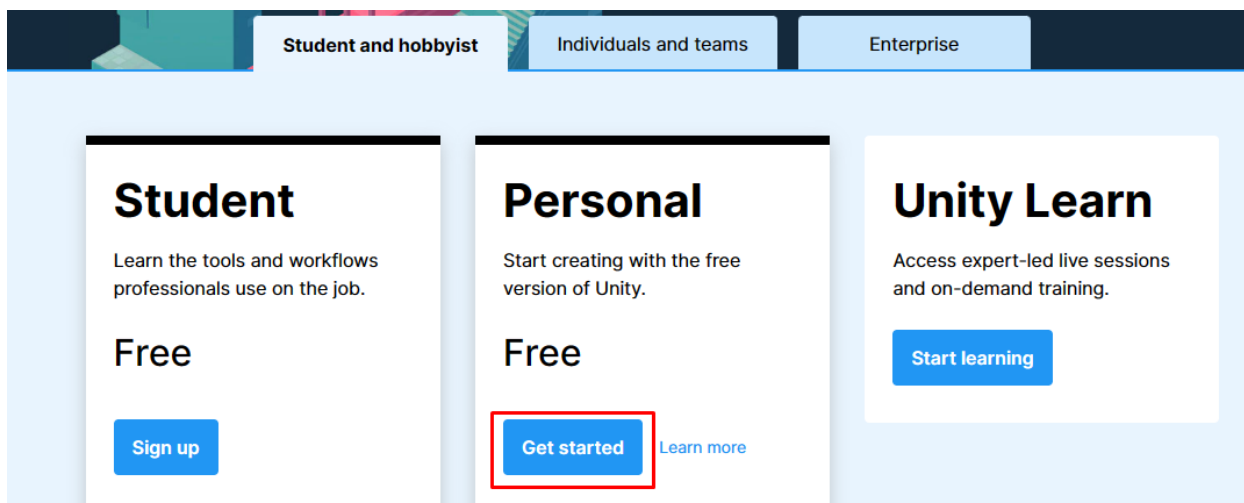


Figura 5 – Opțiunile de selectare

Ca ultimă etapă, am făcut un cont nou și am descărcat programul, iar după instalare, UnityHub s-a deschis. Din interfața aceasta avem posibilitatea să creăm un program nou (apăsând butonul „New project” / „Proiect nou”, evidențiat prin chenarul roșu) sau să deschidem un proiect deja creat anterior, prin selecția acestuia (chenarul verde) așa cum se observă în Figura 6.

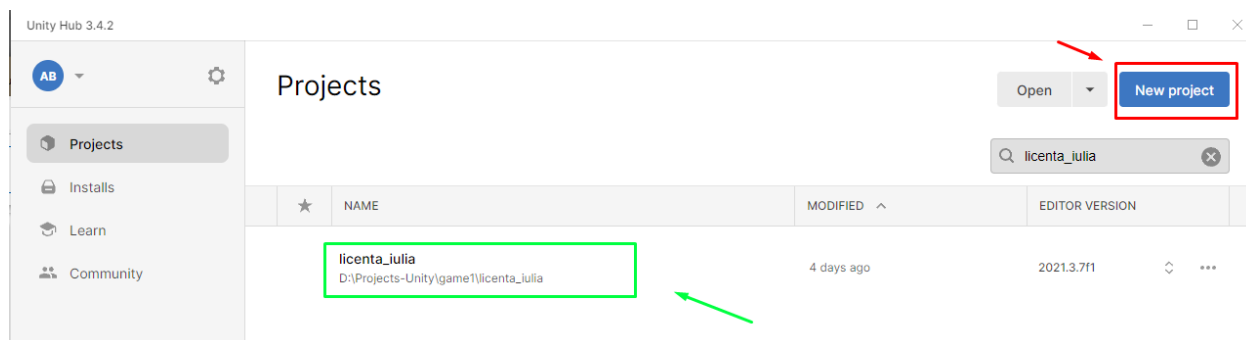


Figura 6 – Captura de ecran cu Unity Hub

Prin selectarea unui proiect nou, Unity ne cere să alegem tipul de aplicație, numele și locația unde vrem să fie salvată (Figura 7).

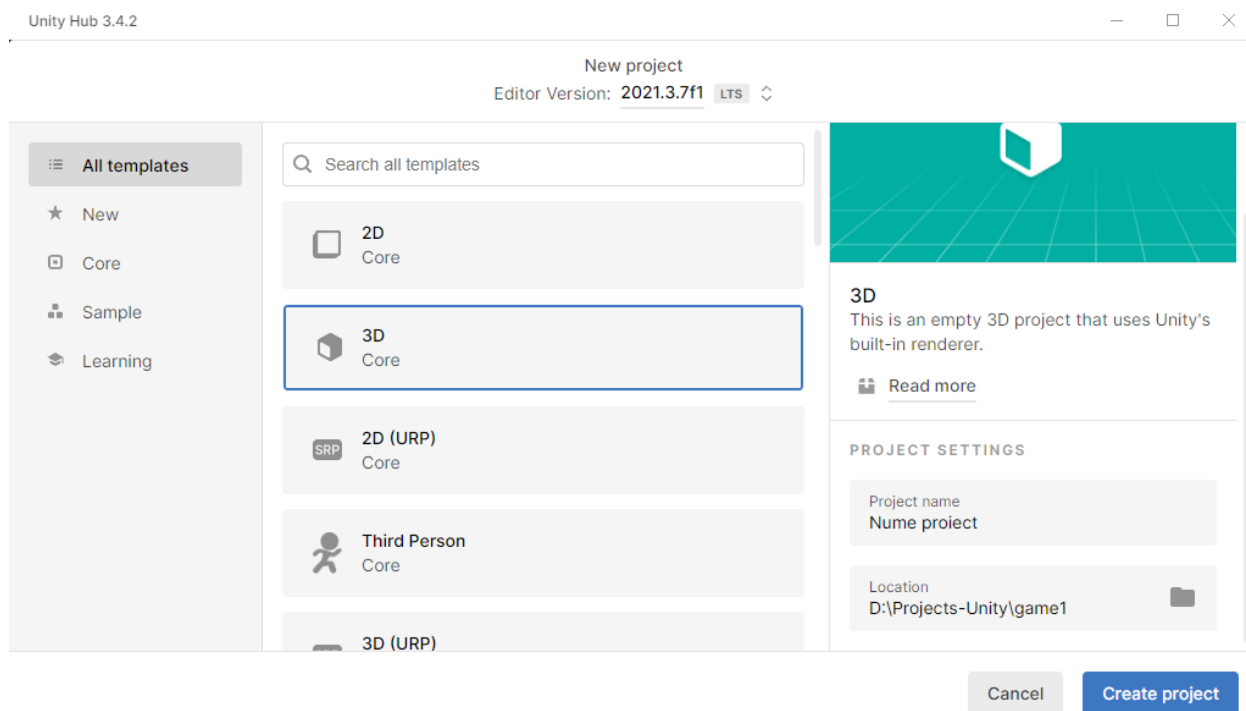


Figura 7 – Captura de ecran cu Unity Hub

Un alt lucru important de menționat este sincronizarea prin cloud a proiectului, oferită de Unity, care m-a ajutat să lucrez de la mai multe stații de lucru (laptop și PC), permițându-mi să-mi continui lucrarea de licență, indiferent de locație. Pentru a beneficia de sincronizare, utilizatorul trebuie să aibă un cont, să fie autentificat cu ele pe ambele platforme și să exporte proiectul online.



Figura 8 – Logo-ul Visual Studio

Descrierea aplicației

Visual Studio Code este o aplicație utilizată în scrierea codului în limbajele HTML, CSS, JavaScript și JSON. De asemenea, include și LESS, Sass, PHP, Python sau C# cu ASP.NET.

La deschiderea Visual Studio Code, utilizatorii sunt întâmpinați de o interfață elegantă și modernă, care oferă un sentiment de claritate și organizare. Culorile vibrante și designul intuitiv creează senzația că te afli într-un univers digital plin de posibilități.

Editorul de cod este una dintre componentele centrale ale Visual Studio Code și este conceput pentru a permite utilizatorilor să scrie și să editeze codul sursă într-un mod eficient. Funcționalitățile avansate de evidențiere a sintaxei și completare automată facilitează recunoașterea și scrierea corectă a elementelor de cod, sporind astfel precizia și viteza dezvoltării.

Un aspect distinctiv al Visual Studio Code este abordarea sa: platforma oferă o gamă largă de extensii și plugin-uri dezvoltate de comunitatea utilizatorilor, permitând personalizarea și extinderea funcționalității editorului în funcție de necesitățile specifice ale proiectului sau limbajului de programare utilizat.

Printre funcționalitățile cheie ale Visual Studio Code se numără și cele legate de rezolvarea problemelor și profilare a codului. Aceste instrumente permit programatorilor să identifice și să rezolve erori și probleme de performanță într-un mod eficient, contribuind astfel la îmbunătățirea calității și performanței aplicațiilor.

Visual Studio Code oferă, de asemenea, suport pentru gestionarea proiectelor și fișierelor prin intermediul panourilor și ferestrelor laterale, permitând o navigare ușoară și o gestionare eficientă a resurselor.

La instalarea lui Unity, utilizatorilor li se recomandă să instaleze și Visual Studio Code sau Community dacă nu o au deja, deoarece acestea împreună reprezintă o modalitate completă de a crea jocurile video.

O caracteristică interesantă a Visual Studio Code este integrarea puternică cu sistemele de control al versiunilor, cum ar fi Git. Utilizatorii pot gestiona și monitoriza eficient modificările în codul lor folosind funcționalitățile încorporate pentru controlul versiunilor.

Istoria alianței lui Visual Studio cu Unity

Conexiunea dintre platforma de dezvoltare Unity și mediul de dezvoltare integrat Visual Studio reprezintă un rezultat al colaborării strânse între Unity Technologies și Microsoft. Această alianță strategică a avut ca scop principal îmbunătățirea experienței de dezvoltare a jocurilor și a aplicațiilor interactive, utilizând puterea și flexibilitatea motorului de joc Unity împreună cu mediul de dezvoltare sofisticat oferit de Visual Studio.

Istoria conectării lui Unity cu Visual Studio își are rădăcinile în etapele incipiente ale dezvoltării jocurilor bazate pe platforma Unity. La început, dezvoltatorii se bazau în mare parte pe un editor de cod simplu și limitat încorporat în cadrul platformei Unity. Cu toate acestea, pe măsură ce proiectele de dezvoltare a jocurilor deveneau din ce în ce mai complexe și impuneau necesități avansate de programare, era evident că o integrare puternică cu un mediu de dezvoltare extern ar reprezenta un avantaj considerabil.

În această optică, parteneriatul între Unity Technologies și Microsoft a condus la integrarea platformei Unity cu Visual Studio, un mediu de dezvoltare integrat bine-cunoscut și larg utilizat în industria software. Integrarea a generat posibilitatea pentru dezvoltatori de a lucra în mod transparent în mediul familiar al Visual Studio, beneficiind de facilitățile avansate pe care acesta le oferă, cum ar fi evidențierea sintaxei, completarea automată și depanarea eficientă a codului.

Prima versiune a integrării Unity cu Visual Studio a fost lansată în 2010 și a furnizat dezvoltatorilor opțiunea de a utiliza Visual Studio ca principal mediu de dezvoltare pentru proiectele Unity. Aceasta a permis programatorilor să beneficieze de toate avantajele pe care Visual Studio le oferă, inclusiv instrumente puternice de depanare și profilare, integrare cu sisteme de control al versiunilor și posibilitatea de a utiliza limbi de programare suplimentare, precum C#.

De-a lungul timpului, integrarea Unity cu Visual Studio a fost continuu îmbunătățită și actualizată pentru a oferi o experiență de dezvoltare mai fluidă și eficientă. Caracteristici precum navigarea între script-uri Unity, evidențierea corectă a sintaxei specifice Unity, completarea automată inteligentă și instrumentele avansate de depanare și profilare au fost adăugate pentru a sprijini dezvoltarea jocurilor în Unity utilizând Visual Studio.

Astăzi, alianța dintre Unity și Visual Studio este recunoscută ca o soluție de încredere, preferată de către dezvoltatori pentru crearea de jocuri în mediul Unity. Prin intermediul acestei integrări strânse, dezvoltatorii beneficiază de un flux de lucru și instrumente.

Integrarea Unity cu Visual Studio a avut un impact semnificativ asupra eficienței și productivității dezvoltatorilor de jocuri. Prin utilizarea Visual Studio ca mediu de dezvoltare principal pentru proiectele Unity, dezvoltatorii au acces la caracteristici avansate, cum ar fi refactoring-ul automat, analiza statică a codului și deblocarea performanței. Aceste funcționalități îmbunătățesc calitatea codului și facilitează procesul de dezvoltare, reducând erorile și timpul necesar pentru dezvoltare și depanare.

O altă inovație adusă de integrarea Unity cu Visual Studio este posibilitatea de a dezvolta jocuri în limbajul de programare C#. Acesta este un limbaj puternic și versatil, care oferă dezvoltatorilor un control mai mare asupra codului și permite o dezvoltare mai rapidă și mai eficientă. Integrarea cu Visual Studio facilitează dezvoltarea în C#, oferind funcționalități precum completarea automată a codului, navigarea rapidă prin proiect și instrumente avansate de depanare.

Un alt beneficiu al integrării Unity cu Visual Studio este posibilitatea de a utiliza extensii și plugin-uri disponibile în ecosistemul Visual Studio. Dezvoltatorii pot accesa și utiliza o gamă largă de extensii dezvoltate de comunitatea Visual Studio, care îmbunătățesc fluxul de lucru și adaugă funcționalități suplimentare. De la instrumente de testare și analiză a codului la integrarea cu servicii de gestionare a proiectului și publicare, extensiile extind capacitatea de dezvoltare a jocurilor în Unity.

În plus, integrarea Unity cu Visual Studio facilitează colaborarea în echipă și partajarea codului. Dezvoltatorii pot utiliza instrumentele de control al versiunilor din Visual Studio pentru a gestiona și sincroniza eficient modificările în codul sursă al jocului. Aceasta facilitează lucru în echipă și permite dezvoltatorilor să lucreze în paralel asupra aceluiași proiect, contribuind la creșterea eficienței și coordonării în procesul de dezvoltare.

	Visual Studio	Unity
ANUL ÎNFIINȚĂRII	1997	2004
SCOP PRINCIPAL	mediu de dezvoltare integrat (IDE)	platformă de dezvoltare
LIMBAJE DE PROGRAMARE	C, C++, C++/CLI, Visual Basic	C# și UnityScript
	TypeScript, .NET, C#, F#, JavaScript	
	XML, XSLT, HTML, și CSS	
SUPORT PENTRU DEZVOLTAREA DE SOFTWARE	da	da

CAPITOLUL 2 – DESIGNUL JOCULUI

2.1 – Tendințe curente în designul jocurilor

Având în vedere natura dinamică a lumii jocurilor video, pentru a rămâne relevant și a te distinge în această industrie, înțelegerea și adaptarea la tendințele curente prezintă un avantaj. În acest subcapitol, îmi doresc să subliniez unele dintre cele mai proeminente și influente tendințe care modelează designul de jocuri în prezent.

O tendință de bază în designul de jocuri este implementarea realității virtuale (VR) și a realității augmentate (AR). Aceste tehnologii oferă dezvoltatorilor o oportunitate de a crea experiențe interactive. Jocuri precum “Beat Saber” sau “Pokemon Go” ilustrează puterea acestor tehnologii de a propulsa jocurile către un grad superior de realism și interactivitate. Mai jos se poate observa impactul la lansare al jocului “Pokemon Go”.

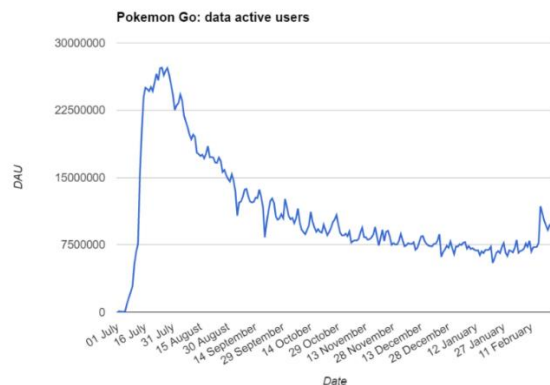


Figura 9 – informații privind numărul de utilizatori activi ai aplicației Pokemon Go

Un alt curent care a câștigat popularitate este acela al jocurilor axate pe poveste și experiențe narrative. Jocuri precum “The Last of Us” sau “Life is Strange” combină aspectele tradiționale ale gameplay-ului cu o poveste îmbietoare și personaje complexe. Aceste jocuri nu pun neapărat accentul pe reflexe rapide sau rezolvarea de puzzle-uri, ci se concentrează mai mult pe furnizarea unei experiențe emoționale și narrative jucătorului. Un alt lucru important de menționat, este că popularitatea jocului în “The Last of Us II” lansat în 2020 s-a datorat diversificării: o protagonistă feminină.

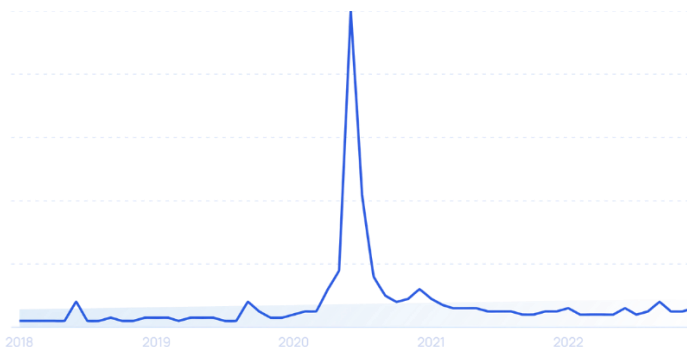


Figura 10 – informații privind numărul de utilizatori activi ai aplicației The Last Of Us II

În plus, se remarcă o tendință de a combina stiluri diferite de jocuri. Multe jocuri recente încorporează elemente din diferite genuri pentru a crea o experiență unică. Un exemplu notabil este “Fortnite”, care combină elemente din jocurile de supraviețuire, jocurile de construcție și jocurile shooter.

O altă practică frecventă este actualizarea regulată a conținutului jocurilor prin intermediul patch-urilor și a extinderilor. Acest lucru ajută la menținerea angajamentului jucătorilor și la extinderea duratei de viață a jocului.

Un element care câștigă teren în jocurile contemporane este componenta socială. Jocurile moderne se bazează nu doar pe o experiență individuală, ci se străduiesc să creeze un mediu în care jucătorii pot interacționa, concura și colabora. Jocuri precum “Among Us” sau “Fall Guys” au capitalizat pe acest aspect, oferind o experiență socială prin natura gameplay-ului lor.

În același timp, se poate observa o tendință de diversificare și incluziune în jocuri. Dezvoltatorii se străduiesc să reprezinte un spectru larg de personaje și contexte, reflectând identitățile și experiențele diverse ale jucătorilor lor. Această abordare este benefică, deoarece face jocurile mai accesibile și mai relevante pentru un public diversificat.

De asemenea, o altă tendință notabilă în designul de jocuri este creșterea importanței jocurilor indie. Cu accesibilitatea crescută a instrumentelor de dezvoltare a jocurilor, dezvoltatorii independenți au acum capacitatea de a crea jocuri care pot concura cu cele produse de studiourile mari. Aceasta duce la o mai mare diversitate și inovație în industrie, deoarece dezvoltatorii indie au libertatea de a experimenta cu concepte noi și neortodoxe.

Este fascinant să observ cum aceste tendințe se reflectă în propriul meu proiect. Am încercat să combin elemente din diferite genuri, să creez o experiență accesibilă și captivantă și să recunosc importanța actualizărilor periodice și a conținutului nou. În acest fel, lucrarea mea de licență nu este doar un produs al timpului său, ci și o reflectare a acestuia.

2.2 – Descrierea jocului dezvoltat

În acest capitol, voi detalia elementele centrale ale jocului creat, pornind de la caracterul principal, mecanicile de joc și atmosfera generală, la elementele de design și muzica ce însoțesc experiența jucătorului.

Personajul principal al jocului (*Figura 11*) este un erou agil ce se deplasează prin lume printr-o serie de sărituri de la o platformă la alta. Scopul este simplu, dar provocator: supraviețuirea pentru cât mai mult timp posibil. Jocul recompensează atenția și reflexele rapide, oferind un echilibru delicat între dificultate și satisfacția jocului.

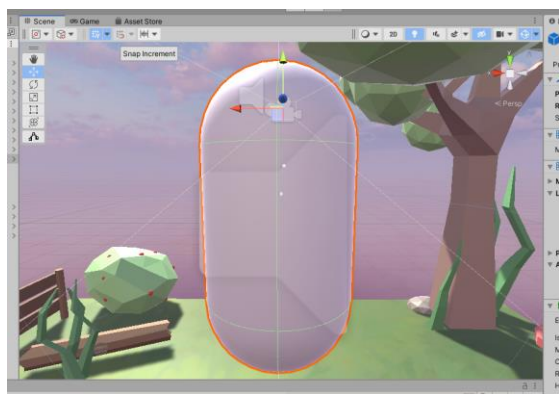


Figura 11 – Captură de ecran (caracterul controlat de jucator)



Figura 12 – Captură de ecran (inceputul jocului)

După cum se poate observa din Figura 12, doar prima platformă are elemente de natură în ea. Scopul a fost să creez un mediu confortabil și liniștitor, astfel încât, jucătorul să se simtă în siguranță într-un mediu plăcut. Nu este niciun pericol pe prima platformă, iar jucătorul poate alege să își înceapă călătoria spre necunoscut când este el pregătit. Între prima și a doua platformă, utilizatorul se poate întoarce dacă nu se simte încă pregătit să avanseze – acest lucru ajută la familiarizarea acestuia cu lumea din joc.

Un element central al jocului este repetabilitatea; dacă personajul cade de pe o platformă sau este atins de inamici, nivelul se reinițializează. Acesta este un mecanism clasic de "trial and error", ce aduce un plus de dinamică și competitivitate în joc. Nivelele au fost concepute să intrige jucătorii, primul nivel având o dificultate medie, iar cel de-al doilea una ușoară.

Atmosfera generală a jocului a fost creată pentru a oferi o experiență plăcută și captivantă. Am folosit o paletă de culori ușor vibrantă și o coloană sonoră calmă care se potrivește cu ritmul jocului și amplifică sentimentul de relaxare și destresare. Am dorit să încurajez jucătorul să se simtă de parcă se află într-o lume fără griji, creând astfel o atmosferă confortabilă.

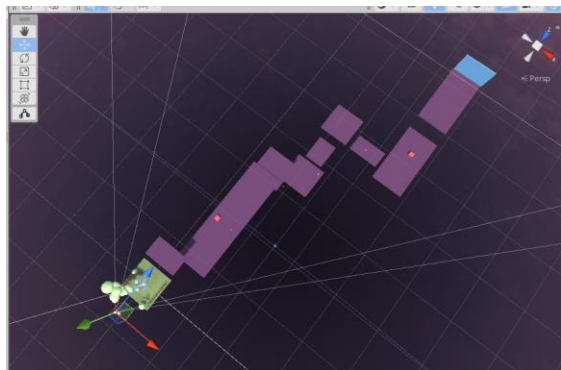


Figura 13 – Captură de ecran cu primul nivel

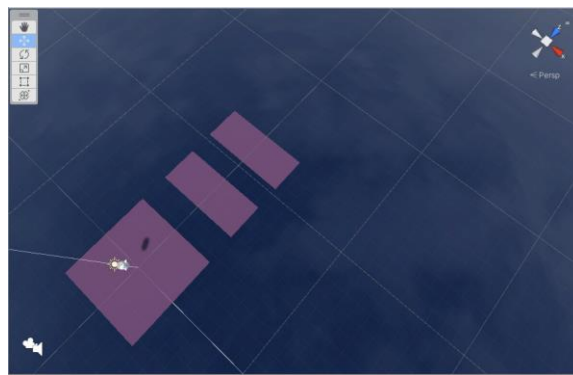


Figura 14 – Captură de ecran cu al doilea nivel

Pe parcursul aventurii, personajul întâlnește diferiți inamici care pot fi înfrânți prin sărituri. Aceștia se mișcă și prezintă un pericol pentru jucători, de care trebuie să se ferească sau să se riște și să îi înfrângă, pentru niște puncte în plus.

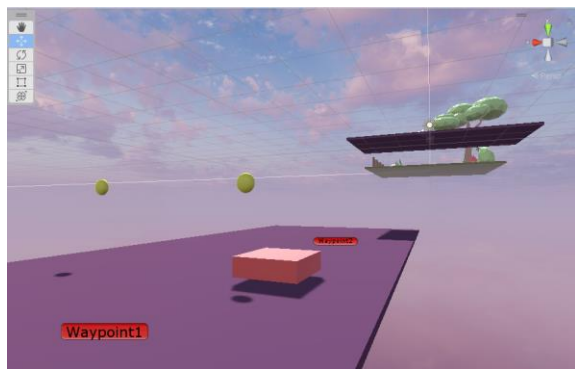


Figura 15 – Captură de ecran cu inamicul

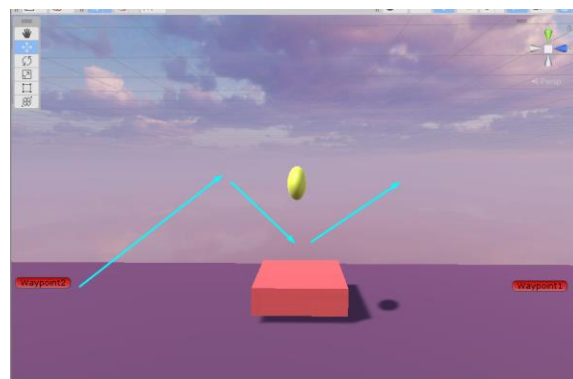


Figura 16 – Captură de ecran peste care am desenat mecanica de înfrângere al inamicului

Pentru înfrângerea inamicilor am avut ca inspirație jocul Mario 2D cu scopul de a da lumii tridimensionale elemente nostalgice. Astfel, personajul principal poate sări pe capul inamicilor pentru a-i învinge. Această mecanică de joc aduce o notă de familiaritate și permite jucătorului să-și folosească strategiile și abilitățile pentru a depăși obstacolele și a înfrunta inamicii într-un mod distractiv și satisfăcător.

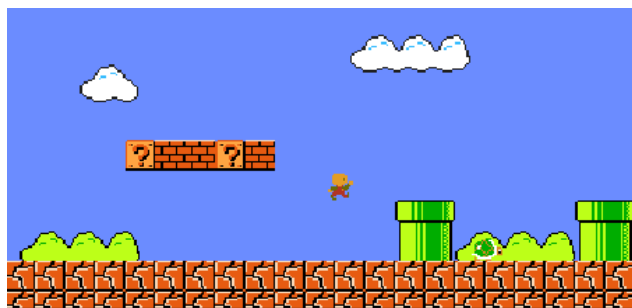


Figura 17 – jocul Mario

Sistemul de colectare oferă jucătorilor posibilitatea de a fi recompensați pentru mecanicile lor. Punctele au culoare galbenă, se învârt și sunt regăsite la înălțime de cele mai multe ori. La capturarea unui punct (adică la atingere), obiectul dispare din spațiu, iar scorul din colțul nord-vestic al ecranului crește.

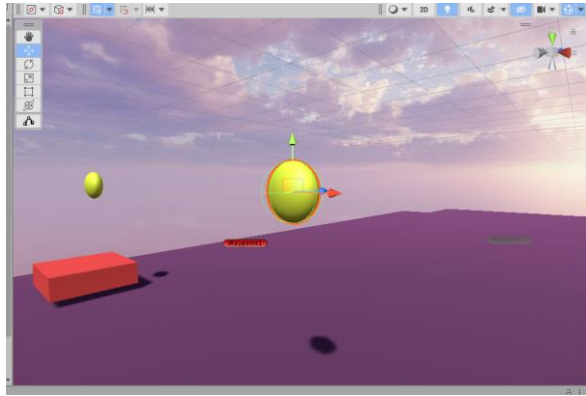


Figura 18 – Captură de ecran cu obiectele ce pot fi colectate

Trecerea de la un nivel la altul se face printr-o tranziție la atingerea unui obiect de culoare albastră, care simbolizează portalul către următoarea etapă. Imediat ce personajul principal intră în contact cu acest obiect, al doilea nivel se încarcă.

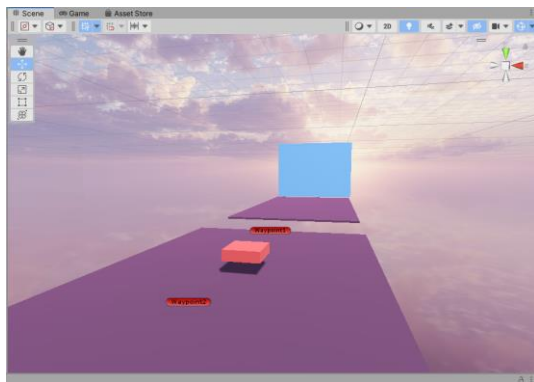


Figura 19 – Captură de ecran cu obiectul albastru ce activează tranziția

Designul obiectelor și al caracterului este simplu, favorizând în schimb mecanicile de joc complexe și bine definite. Cerul roz și elementele naturale de pe prima platformă oferă o atmosferă confortabilă și relaxantă, menită să-i ajute pe jucători să se deconecteze de la grijile cotidiene și să se îmbrățișeze în lumea jocului.

Muzica, inspirată din “Diana – Sessions” de la Riot Games, aduce un plus de atmosferă și autenticitate. Ritmurile sale liniștitoare se potrivesc cu tonul general al jocului.



Figura 20 – poza de tip coperta de la videoclipul Diana – Sessions, de pe canalul de YouTube Riot Games

În timp ce jucătorul se află în meniul principal, este prezentat un captură de ecran din scena de pornire a jocului (Figura 21). Această imagine oferă o mică privire asupra lumii și atmosferei jocului, creând anticipare și stimulând curiozitatea jucătorului. De asemenea, îi permite acestuia să se obișnuiască cu estetica și stilul vizual al jocului încă de la început, pregătindu-l pentru experiența captivantă care urmează.

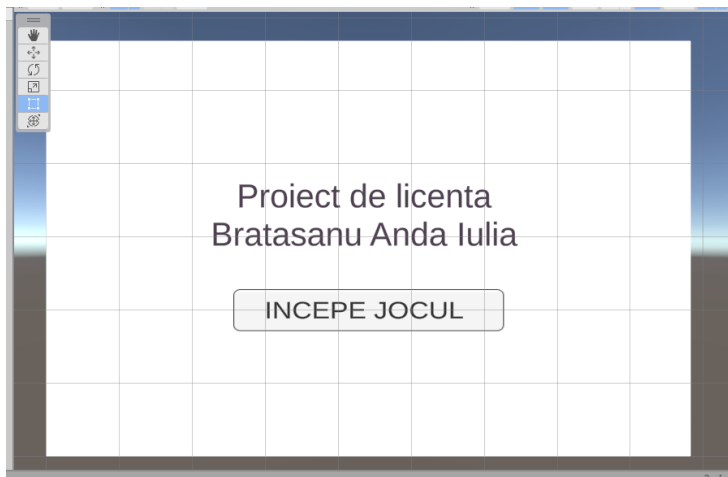


Figura 21 – captură de ecran cu scena de pornire

Astfel, jocul meu de licență oferă o experiență completă, combinând mecanici solide cu un design vizual plăcut și muzică ambientală. Prin integrarea meniului principal intuitiv și captura de ecran din scena de pornire, se creează o atmosferă invitantă și intrigantă pentru jucător. Scopul final este acela de a oferi jucătorilor un mediu în care să se relaxeze, să se distreze și să uite, chiar și pentru o clipă, de grijile cotidiene, intrând astfel într-o experiență de joc captivantă și plină de satisfacție.

CAPITOLUL 3 – ETAPELE PROCESULUI DE LUCRU

3.1 – Mecanici și implementare

3.1.1 – Obiecte și legile fizice

Mi-am început aplicația cu adăugarea unui obiect de tip cub, iar cu ajutorul variabilelor din Scale (Transform), acesta a prins forma de platformă pe care mi-am dorit-o (Figura 22). Am urmărit să creez un joc cu mai multe astfel de platforme, pentru a da jucătorului posibilitatea să-și antreneze tehnicile de supraviețuire în încercarea de a rămâne pe platforme pentru cât mai mult timp.

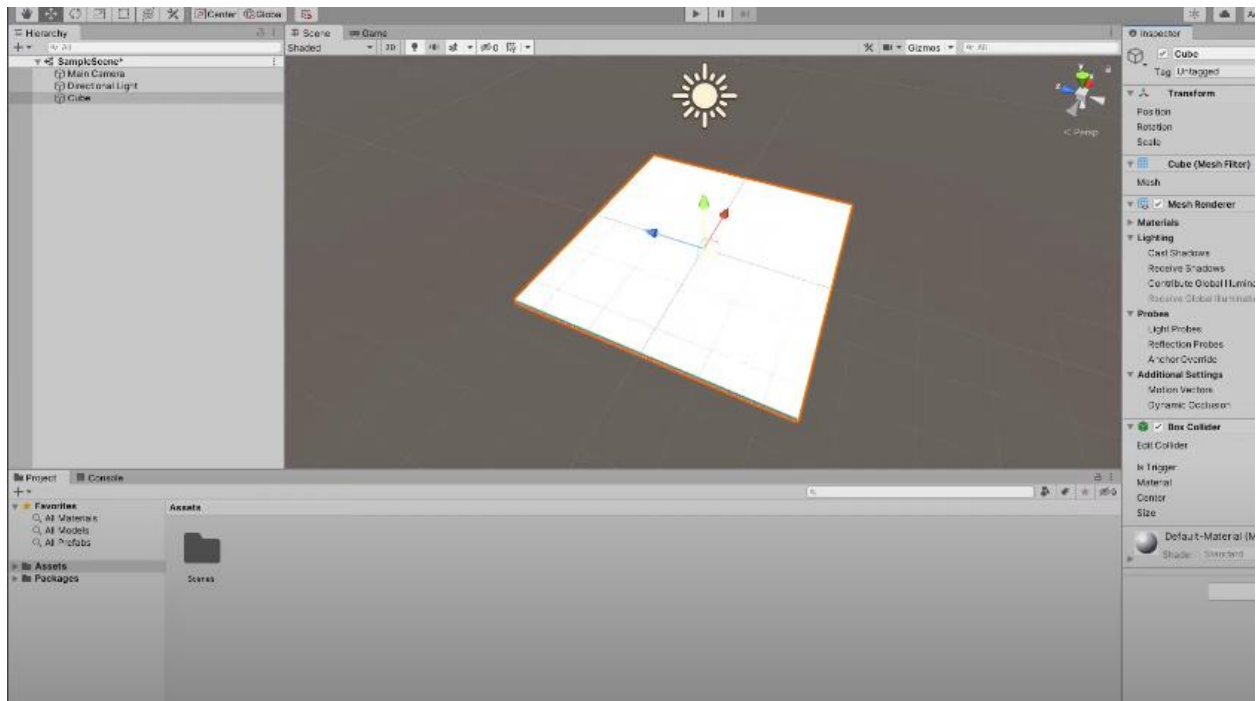


Figura 22

Pentru modelul caracterului principal, am folosit un obiect de tip "Capsule" / "Capsulă" ca element provizoriu pentru a seta dimensiunile pe care mi le doresc ale personajului. Acest model se poate înlocui mai târziu, fapt pentru care am putut să mă concentrez pe partea de programare înainte de cea artistică. La fel cum am procedat înainte, am editat dimensiunile obiectului și i-am înălțat poziția pentru a putea mai târziu să verific legile fizicii ale lumii create (Figura 23).

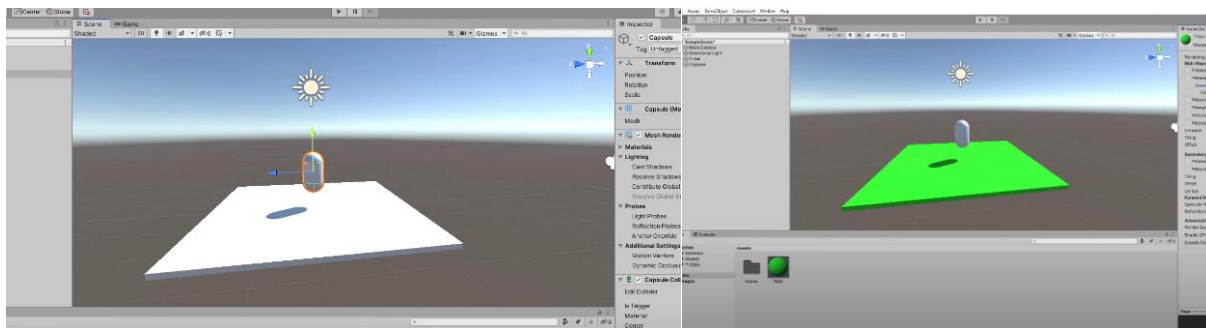


Figura 23

Figura 24

Pentru a diferenția terenul de personaj, am creat un material pe care l-am numit "Floor" / "Podea", și i-am setat proprietățile, selectând culoarea verde. Apoi, am tras materialul peste platformă, iar aceasta și-a schimbat culoarea în cea dorită (Figura 24). De asemenea, am redenumit numele obiectelor folosite în unele mai sugestive, lucru pe care l-am realizat și pentru viitoarele elemente ale jocului.

Am observat că obiectele pe care le-am creat au atașat deja un component numit "Capsule Collider", însă, atunci când am vrut să verific gravitația mutându-mă pe perspectiva jucătorului, am sesizat că nu se întâmplă nimic (Figura 25). M-am documentat și am înțeles că pentru a seta gravitația trebuie să adaug un component nou la obiectul jucătorului pe nume "Rigidbody" / "Obiect rigid" (Figura 26).

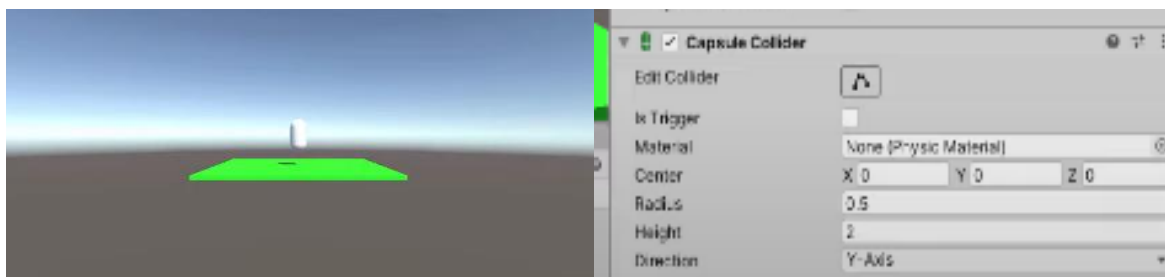


Figura 25 – gameview din care se observă ca gravitația nu este încă implementată / componenetul Capsule Collider

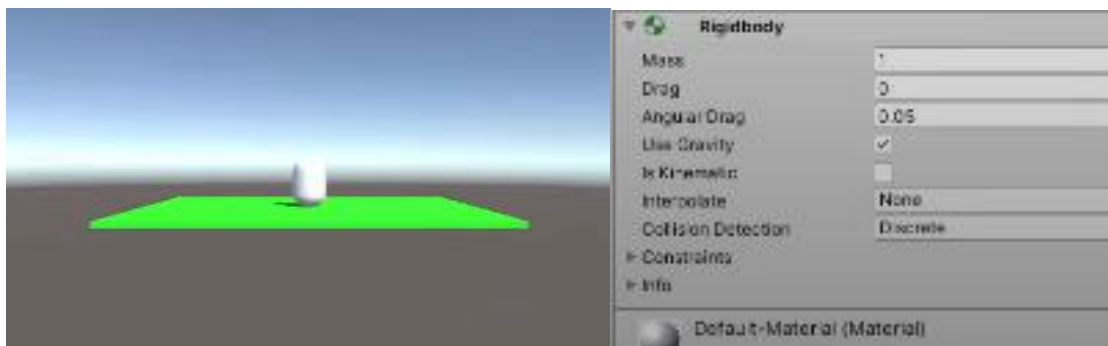


Figura 26 – gameview din care se observă ca gravitația este implementată / componenetul Rigidbody

Acest lucru se datorează Mesh Rendererului, care oferă celor două obiecte formă în spațiu. Prin teste am observat că dacă dezactivez Mesh Rendererul de la teren, obiectul caracterului o să cadă prin el. Este important de menționat și componenta "Rigidbody", care atașat și la teren ar permite platformei să cadă împreună cu obiectul jucătorului în spațiul predefinit de Unity.

Fiind mulțumită de legile fizicii fundamentale predefinite de Unity prin componentele menționate anterior, am ales să continui progresul aplicației prin schimbarea poziției camerei pentru a oferi jucătorului o perspectivă mai clară a scenei: am înălțat-o, apropiat-o de scenă și rotit-o pe axa "X" într-un unghi de 60 de grade (figurile 27 și 28).

Schimbarea de unghi se poate realiza atât prin rotirea propriu zisă a camerei cu ajutorul mouse-ului, cât și prin schimbarea valorii rotației din interfața Unity prin panoul "Transform" pentru o precizie mai mare.

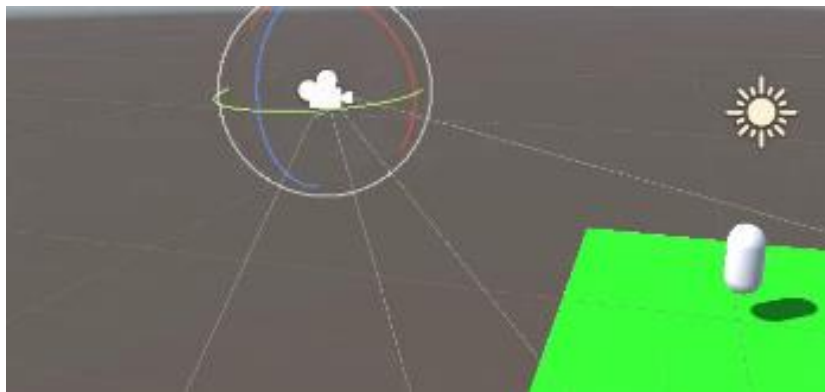


Figura 27 – Rotatia camerei

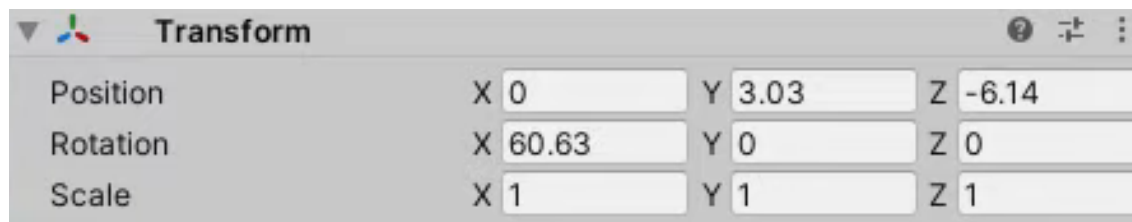


Figura 28 – interfața Unity

Aceste ajustări ale camerei au avut un impact semnificativ asupra experienței de joc, permițând jucătorului o perspectivă mai dinamică și mai captivantă. De la această nouă poziție, jucătorul poate observa mai bine platformele și obstacolele din calea sa, făcând planificarea și navigarea mai eficiente.

3.1.2 – Mișcarea caracterului prin interacțiunea cu utilizatorul

Mi-am propus să dau viață personajului principal și să ofer jucătorului posibilitatea de a controla caracterul prin apăsarea tastelor de la tastatură. Pentru aceasta, am avut nevoie să scriu linii de cod. Am creat un script nou în biblioteca de elemente, l-am denumit sugestiv și apoi l-am tras cu mouse-ul peste obiectul jucătorului pentru a oferi celor două acces la informații între ele și a le conecta. Apoi, prin intermediul limbajului C# am transmis programului în funcția "Update()" (care verifică liniile din cod în fiecare frame) ca obiectul jucător se se poată mișca în toate direcțiile, astfel:

```
// Start is called before the first frame upd
@ Unity Message | 0 references
void Start()
{
    rb = GetComponent<Rigidbody>();
}

// Update is called once per frame
@ Unity Message | 0 references
void Update()
{
    if (Input.GetKeyDown("space"))
    {
        rb.velocity = new Vector3(0, 5, 0);
    }

    if (Input.GetKey("up"))
    {
        rb.velocity = new Vector3(0, 0, 5);
    }
}
```

Figura 29

```
if (Input.GetKey("right"))
{
    rb.velocity = new Vector3(5, 0, 0);
}

if (Input.GetKey("down"))
{
    rb.velocity = new Vector3(0, 0, -5);
}

if (Input.GetKey("left"))
{
    rb.velocity = new Vector3(-5, 0, 0);
}
}
```

Figura 30

După cum se poate observa în imaginea anterioară, pentru a face programul mai eficient și liniile de cod mai clare, am creat o variabilă pe nume "rb" în care am stocat și permis conectarea scriptului la componenta Rigidbody din Inspector-ul obiectului jucătorului, apoi l-am apelat de câte ori am avut nevoie, fără să întrebuițez aceeași acțiune a programului de fiecare dată.

Deși aceste linii de cod menționate sunt perfect funcționale, iar obiectul s-a putut controla din tastatură, am vrut să-l rescriu pentru a da jucătorului posibilitatea de a controla caracterul prin mai multe seturi de taste (de exemplu W, A, S, D, dar și prin săgeți). Pentru a face acest lucru

posibil, am intrat în Project Settings > Input Manager și am reținut numele metodelor de input pentru a face referire la ele în program (Horizontal, Vertical, Jump).

Tot aici avem posibilitatea să schimbăm tastele de control, sensibilitatea, axele și multe altele.

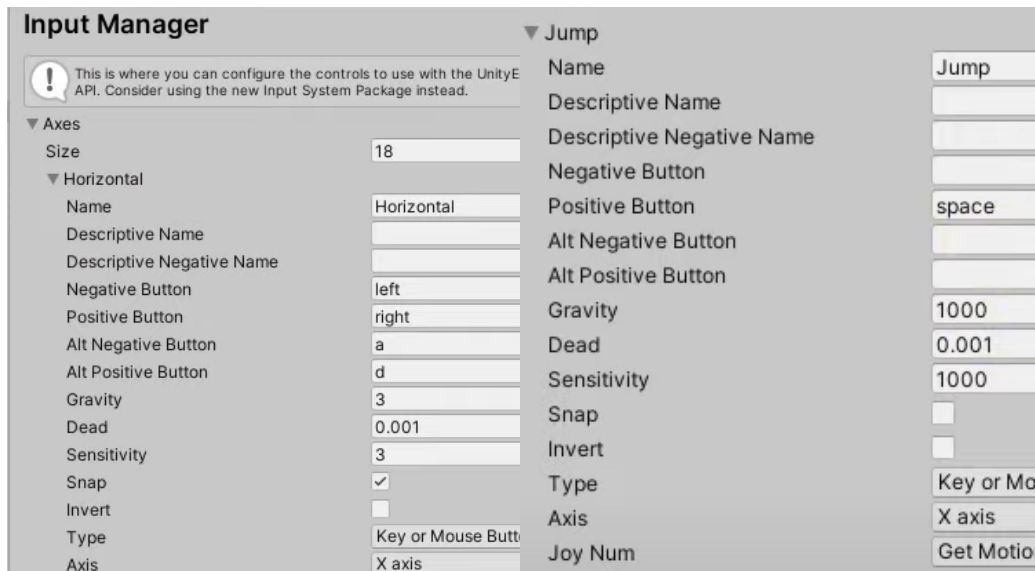


Figura 31



Figura 32

```
public class PlayerMovement : MonoBehaviour
{
    Rigidbody rb;
    [SerializeField] float movementSpeed = 5f;
    [SerializeField] float jumpForce = 5f;
    void Start()
    {
        rb = GetComponent<Rigidbody>(); }
}
```

```

void Update()
{
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");
    rb.velocity = new Vector3(horizontalInput * movementSpeed, rb.velocity.y, verticalInput *
movementSpeed);
    if (Input.GetButtonDown("Jump")) { Jump(); }

    void Jump()
    {
        rb.velocity = new Vector3(rb.velocity.x, jumpForce, rb.velocity.z);
        jumpSound.Play();
    }
}

```

Prin aceste linii de cod, caracterul nostru se poate mișca pe toate axele folosind seturile de taste menționate, și mai mult decât atât, cu ajutorul lui „[Serialized Field]” putem modifica oricând puterea de săritură și viteza de mișcare din interfața Unity.

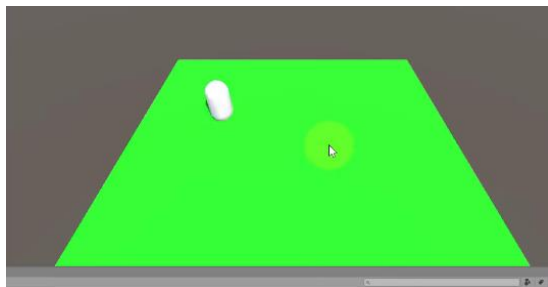


Figura 33



Figura 34

De menționat mai este și că putem modifica și testa aceste două variabile în timp ce suntem în "game view" / "perspectiva jucătorului", iar modificările nu vor fi salvate, ceea ce ne oferă libertatea să experimentăm cu diferite forțe și viteze într-un mod reversibil. Această flexibilitate ne permite să ajustăm parametrii și să reglăm caracteristicile jocului în funcție de preferințele noastre și de rezultatele obținute în timpul testării. Astfel, putem găsi combinația perfectă de forță, viteză și rezistență care să ofere jucătorilor o experiență captivantă și echilibrată.

3.1.3 – Prefabs și rezolvarea bug-urilor cu materialele fizice

Pentru a adăuga mai multe platforme, am utilizat tehnologia prefabs din Unity. Aceasta are rolul de a simplifica munca dezvoltatorilor de jocuri, prin încorporarea unor obiecte simple în niște obiecte predefinite de noi, iar astfel, am reușit să multiplicăm platformele împreună cu texturile adăugate împreună.

Acest lucru a fost posibil prin tragerea cu mouse-ul a obiectelor în folder-ul de Prefabs, de restul se ocupă Unity (Figura 35). Putem observa imediat că obiectul nostru și-a schimbat culoarea și logo-ul datorită faptului că este acum un prefab (Figura 36).

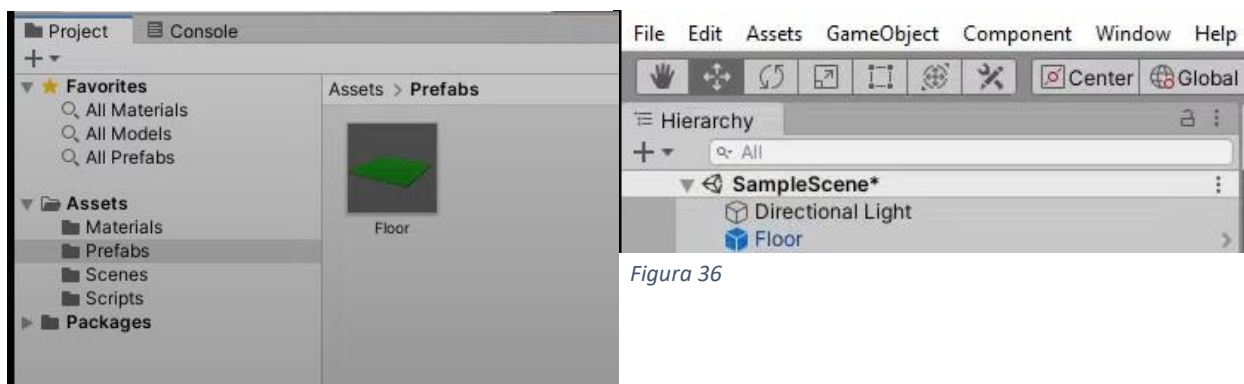


Figura 36

Figura 35

Apoi, am adăugat noul prefab creat, în scenă (Figura 37) și am poziționat-o corespunzător pentru a facilita deplasarea obiectului controlat de jucător cu o simplă săritură (Figura 38).

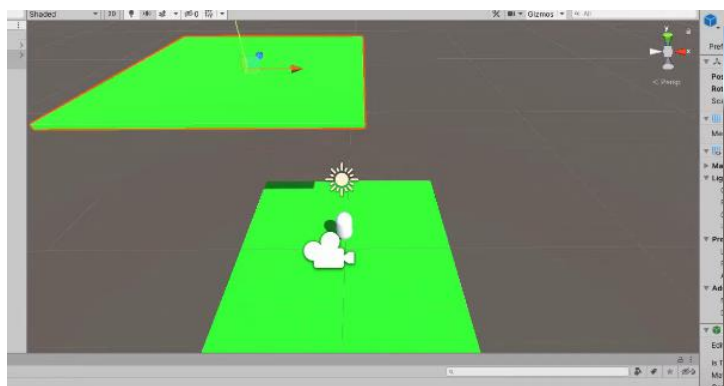


Figura 37

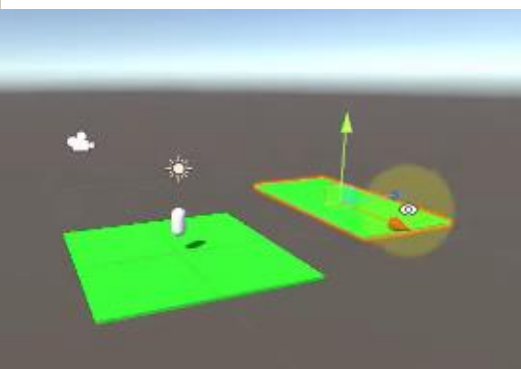


Figura 38

Testând jocul, am observat că atunci când sărim de pe o platformă pe alta, obiectul controlat de jucător nu se mișcă odată cu camera. Pentru a corecta acest detaliu, am atașat camera ca și "copil" al obiectului menționat, iar astfel, mișcarea a fost rezolvată simplu, printr-un click, datorită interfeței. De asemenea, am observat și următoarele "bug-uri" / "erori", și anume:

- a) săritura infinită, care se întâmplă atunci când jucătorul apasă în mod repetat pe tasta "space" asociată cu săritura caracterului (Figura 39);

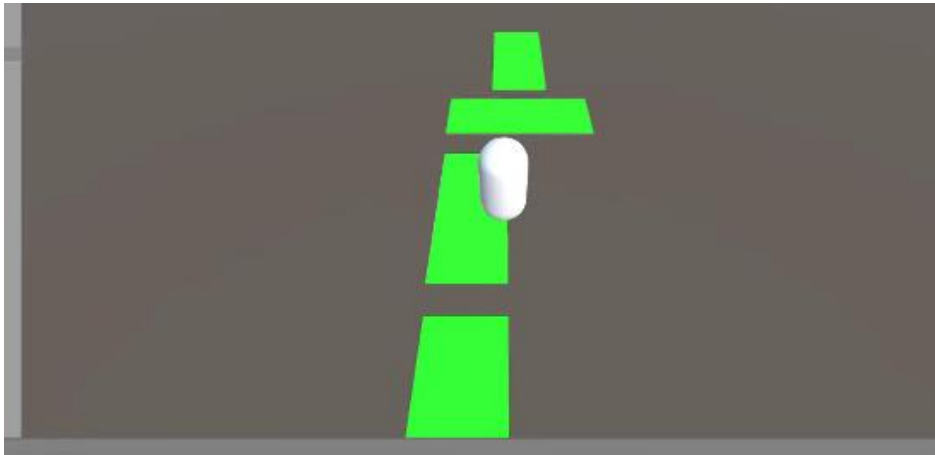


Figura 39

- b) blocarea caracterului în marginea platformei, atunci când obiectul controlat de către jucător sare și atinge una din marginile terenului (Figura 40);

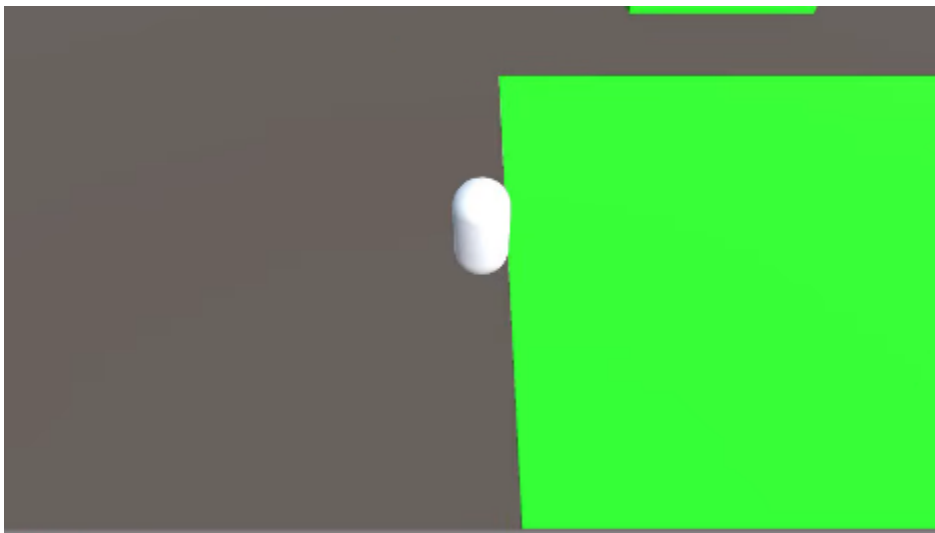


Figura 40

Pentru rezolvarea primului bug, am creat următoarea funcție în scriptul care se ocupă cu mișcarea jucătorului, care verifică dacă obiectul controlat de jucător este pe pământ:

```
bool IsGrounded ()  
{  
    return Physics.CheckSphere(groundCheck.position, .1f, ground); } }
```

Apoi, am adăugat în funcția deja creată pentru acțiunea de săritură, condiția ca funcția de mai sus să fie adevărată, astfel:

```
if (Input.GetButtonDown("Jump") && IsGrounded())  
{  
    Jump();  
}
```

Și bineînțeles, am definit variabilele folosite:

```
[SerializeField] Transform groundCheck;  
[SerializeField] LayerMask ground;
```

Pentru a rezolva eroarea b.) legată de blocarea caracterului în marginea platformei, am creat un material fizic cu textura "alunecoasă" (slippery) pentru a permite obiectului controlat de jucător să alunece în loc să rămână blocat în margine. Am definit proprietățile materialului, cum ar fi coeficientul de frecare și coeficientul de alunecare, pentru a obține efectul dorit. Apoi, am atașat acest material fizic atât terenului, cât și obiectului controlat de jucător, astfel încât să se aplice comportamentul de alunecare și să se prevină blocarea personajului (Figura 41). Această soluție a permis jucătorului să se deplaseze lin și fără probleme pe platforme, eliminând problema blocării în margine și asigurând o experiență mai fluidă în joc.

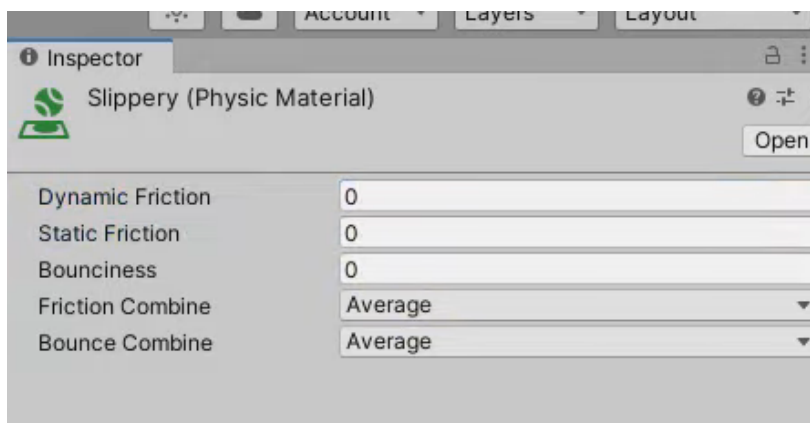


Figura 41

3.1.4 – Inteligența artificială

Inteligența Artificială (AI) în jocuri este utilizată pentru a crea personaje non-jucători (NPC) care reacționează inteligent și se adaptează în funcție de acțiunile jucătorului. AI-ul în jocuri poate controla strategii de atac, navigarea în mediu, luarea deciziilor și simularea inteligenței umane, oferind astfel o experiență interactivă mai imersivă și captivantă.

Pentru a adăuga puțină diversitate la nivel, am creat obiecte mișcătoare (platforme și inamici) care au ca scop să simuleze că sunt controlate de un alt jucător. Concentrându-mă pe fundamente, logica din spate nu este una atât de avansată precum în jocurile video moderne, însă reprezintă un model bun de A.I cu loc de îmbunătățiri. Astfel, mișcarea obiectelor a fost scopul meu țință, și am realizat aceasta cu ajutorul tehnologiei „Waypoints” din Unity. Practic, odată ce sunt setate, aceste repere de locație arată obiectului selectat unde trebuie să meargă, iar ordinea se repetă de la început odată ce s-a terminat.

Waypoint-urile pot să fie obiecte goale (fără textură mesh), care nu sunt vizibile pentru jucători, însă ajută programatorii să creeze astfel de obiecte mișcătoare. Astfel, am generat două obiecte goale în folder-ul nivelului curent și folosind ustensilele de copy-paste, am copiat și lipit pozițiile platformei, urmând să schimb locația celor două obiecte noi, în stânga, respectiv, în dreapta platformei create. Acestea devin locațiile între care platforma se va mișca, de la una la alta.

Pentru a face platforma să se miște în aceste direcții, am creat un script nou C#, pe care l-am denumit sugestiv "WaypointFollower" / "Căre urmărește punctele de referință" și am definit următoarele variabile:

```
[SerializeField] GameObject[] waypoints;  
int currentWaypointIndex = 0; [SerializeField] float speed = 1f;
```

În cadrul scriptului "WaypointFollower", am implementat o logica de deplasare a obiectului către punctele de referință. Folosind funcția "MoveTowards" și variabila "speed", am asigurat o mișcare fluidă și controlată a obiectului de-a lungul traseului definit de punctele de referință.

De asemenea, am inclus și o condiție care verifică dacă obiectul a ajuns la punctul de referință curent. În caz afirmativ, se va selecta următorul punct de referință din lista "waypoints", asigurând astfel o mișcare ciclică și continuă. Această implementare a tehnologiei "Waypoints" aduce un plus de dinamică și interactivitate nivelului, oferind jucătorilor experiențe variate și captivante în timpul jocului.

```
void Update()
{
    if (Vector3.Distance(transform.position,
waypoints[currentWaypointIndex].transform.position) < .1f)
    { currentWaypointIndex++;
        if (currentWaypointIndex >= waypoints.Length)
            { currentWaypointIndex = 0; } }

    transform.position = Vector3.MoveTowards(transform.position,
waypoints[currentWaypointIndex].transform.position, speed * Time.deltaTime);
```

În cadrul funcției "Update()", am definit o verificare a distanței dintre poziția curentă a obiectului și poziția punctului de referință. Dacă distanța este mai mică decât 0.1f, se va trece la următorul punct de referință din lista "waypoints". În cazul în care s-a ajuns la ultimul punct de referință, se va reveni la primul punct, asigurând astfel o mișcare ciclică a obiectului pe traseul definit.

Pentru a face obiectul să se deplaseze către punctele de referință, am utilizat funcția "MoveTowards" în combinație cu variabila "speed" și timpul delta (Time.deltaTime). Astfel, obiectul se va deplasa cu o viteză constantă, independentă de numărul de frame-uri pe secundă al jocului. Acest lucru asigură o experiență consistentă pentru jucători, indiferent de specificațiile dispozitivelor pe care rulează jocul.

Un ultim pas constă în asocierea celor două obiecte goale create anterior cu variabilele din inspector (Figura 42). Observăm în figura 43 că platforma este pregătită și setată pentru mișcare.



Figura 42

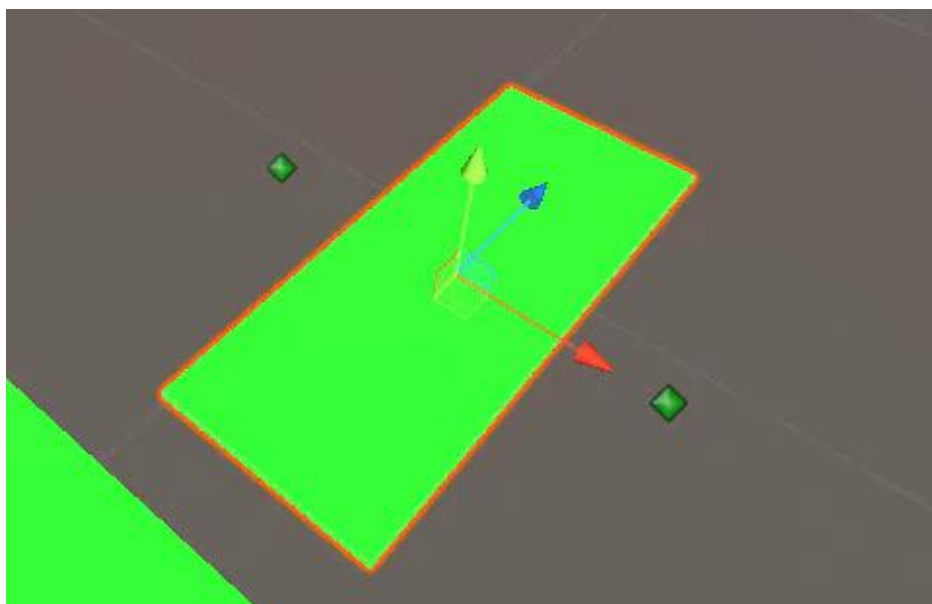


Figura 43

Cu toate acestea, am observat că atunci când jucătorul urcă pe platformă, nu se mișcă împreună cu aceasta. Pentru a rezolva această problemă și a facilita parcursul nivelului, am creat un script nou care verifică dacă jucătorul se află pe platformă. Dacă această afirmație este adevărată, jucătorul devine "copilul" platformei, ceea ce înseamnă că se va mișca împreună cu ea.

Această funcționalitate adăugată în joc contribuie la crearea unei experiențe mai dinamice și captivante pentru jucători. Prin posibilitatea de a interacționa cu platformele și de a se deplasa împreună cu acestea, jucătorii au ocazia de a explora și de a găsi soluții strategice pentru a înfrunta provocările nivelului. Totodată, adăugarea inamicilor oferă un element suplimentar de tensiune și suspans în joc, punând la încercare abilitățile și reflexele jucătorului. Cu fiecare modificare adusă și fiecare detaliu adăugat, nivelul capătă un aspect mai atrăgător și o complexitate crescută, sporind satisfacția și implicarea jucătorilor în experiența de joc.


```

public class StickyPlatform : MonoBehaviour
{
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.name == "Player")
        { collision.gameObject.transform.SetParent(transform); } }

    private void OnCollisionExit(Collision collision)
    {
        if (collision.gameObject.name == "Player")
        { collision.gameObject.transform.SetParent(null); } } }

```

Am procedat similar și cu inamicii, însă în cazul acestora jucătorul nu se va mișca împreună cu ei, adică script-ul de mai sus nu este asociat obiectelor de tip inamic. După câteva modificări ale culorilor și adăugarea inamicilor, una dintre platformele cu provocări arată astfel (Figura 44).

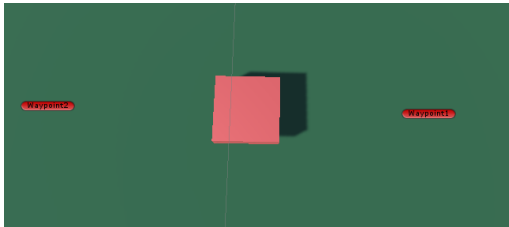


Figura 44

Este important de menționat că am aplicat tehnologia prefab menționată în capitolele anterioare pentru a eficientiza procesul de creare a inamicilor suplimentari, asemănători celor deja existenți (Figura 45).

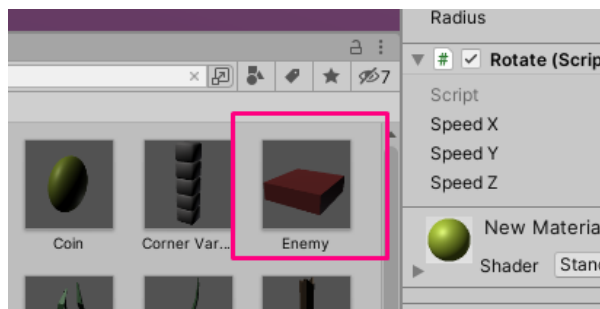


Figura 45

Prin utilizarea acestui sistem, pot adăuga rapid și ușor inamicii, păstrându-le caracteristicile și comportamentul deja definit, ceea ce economisește timp și efort în dezvoltarea jocului. Această abordare îmi oferă flexibilitatea de a ajusta și de a diversifica nivelul prin adăugarea de inamici în funcție de necesități și de gradul de dificultate dorit.

3.1.5 – Sistemul de viață și moarte al caracterului

Până în prezent, jucătorul poate să piardă jocul dacă pică încontinuu de pe platformă, însă nu există opțiunea de a reîncepe jocul. În plus față de automatizarea începerii unui nou joc atunci când caracterul moare, doresc să implementez și atacul din partea inamicilor.

```
public class PlayerLife : MonoBehaviour
{
    bool dead = false;
    private void Update()
    {
        if (transform.position.y < -10f && !dead) {
            Die(); }
    }
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Enemy Body"))
        {
            GetComponent<MeshRenderer>().enabled = false;
            GetComponent<Rigidbody>().isKinematic = true;
            GetComponent<PlayerMovement>().enabled = false;
            Die();
        }
    }
    void Die() {
        Invoke(nameof(ReloadLevel), 1.3f); dead = true; }

    void ReloadLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name); }
}
```

Funcționalitatea de viață a caracterului, din codul de mai sus, este implementată în funcția de tip void "ReloadLevel", care simulează începutul vieții caracterului prin reîncărcarea scenei nivelului. Aceasta este activată în funcția "Die", adică este apelată atunci când jucătorul îndeplinește anumite condiții.

Aceste condiții sunt următoarele:

- a) când caracterul cade de pe platforme (în programare, se verifică poziția caracterului să fie sub 10 pe axa Y);
- b) când inamicul atinge marginile laterale ale inamicilor (în programare, se verifică dacă cele două obiecte se ating folosind tag-ul de inamic, creat pentru a generaliza și a putea asocia diferite tipuri de inamici ca pericole pentru caracter).

Pentru a clarifica acțiunea de “moarte a caracterului” pentru jucător, la coliziunea acestuia cu un inamic, caracterul devine invizibil și nu se mai poate mișca, astfel încât utilizatorul să știe că a pierdut.

În scopul de a adăuga mai multă dinamică, am creat o mecanică nouă prin care jucătorul poate înfrânge inamicii prin săritura pe aceștia, similar cu jocul Mario, dar într-un format 3D. În acest proces, am creat un obiect gol nou (invizibil) deasupra inamicului (Figura 46) și i-am setat dimensiunile pentru a se potrivi în centrul acestuia. La coliziune, inamicul este învins (devine invizibil), iar caracterul principal este propulsat în sus (săritură automată).

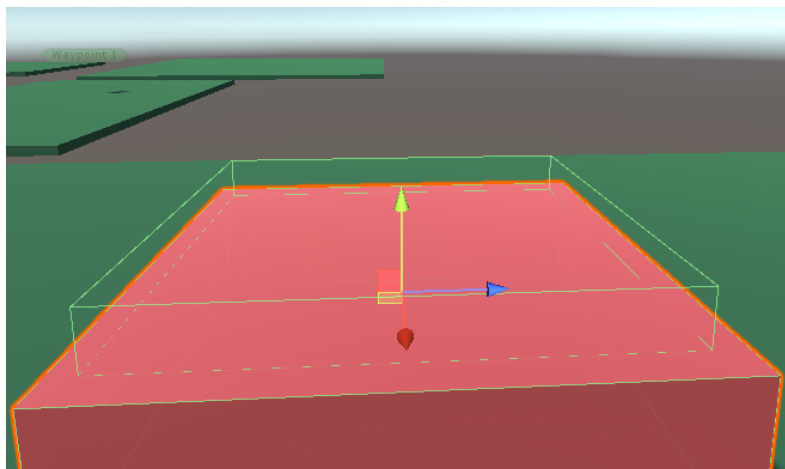


Figura 46



Figura 47

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Enemy Head"))
    {
        Destroy(collision.transform.parent.gameObject);
        Jump(); }
}
```

După cum se poate observa în codul de mai sus, am utilizat tag-ul "Enemy Head" / "Capul Inamicului" (Figura 47) pentru a putea face referire la obiect în timpul scrierii codului. Funcția este implementată în script-ul PlayerMovement și are următorul scop:

La coliziunea jucătorului cu obiectul care are atașat tag-ul "Enemy Head", se vor întâmpla următoarele acțiuni:

- Obiectul părinte al obiectului de coliziune (inamicul) va fi distrus folosind funcția "Destroy()". Astfel, inamicul va deveni invizibil și va fi eliminat din joc.
- Apelăm și funcția "Jump()", care realizează o săritură automată a caracterului principal, oferindu-i jucătorului posibilitatea de a continua parcurgerea nivelului cu un avantaj.

Această mecanică adaugă o nouă dimensiune de joc, în care jucătorul poate învinge inamicii prin săritura precisă pe capul acestora. Această acțiune cere atenție și abilitate din partea jucătorului, aducând o doză de strategie și satisfacție în gameplay.

Un aspect important de menționat este că această mecanică de săritură pe capul inamicilor poate fi utilizată și în combinație cu alte mișcări și acțiuni ale caracterului. De exemplu, jucătorul poate executa o săritură dublă pentru a ajunge la înălțimi mai mari sau poate combina săritura pe capul inamicului cu o acțiune de atac pentru a elimina mai mulți inamici într-un singur salt.

3.1.6 – Colectarea obiectelor

Colectarea obiectelor din joc oferă o modalitate distractivă și captivantă de a obține recompense în joc. Pentru implementarea acestora, am început prin a crea un obiect de tip sferă, micșorându-i proporțiile (Figura 48) și atașându-i un material pentru a-i oferi culoarea galbenă (Figura 49).

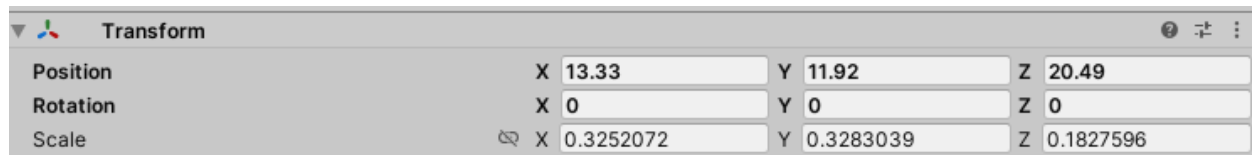


Figura 48

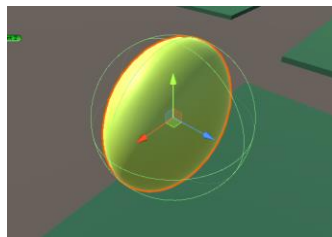


Figura 49

Un aspect important de menționat este că această mecanică de săritură pe capul inamicilor poate fi utilizată și în combinație cu alte mișcări și acțiuni ale caracterului. De exemplu, jucătorul poate executa o săritură dublă pentru a ajunge la înălțimi mai mari sau poate combina săritura pe capul inamicului cu o acțiune de atac pentru a elimina mai mulți inamici într-un singur salt. Această flexibilitate și interacțiune între mișcările caracterului și mecanica săriturii pe capul inamicilor adaugă adâncime și diversitate în gameplay, încurajând jucătorii să experimenteze și să găsească strategii unice de abordare a nivelului.

```
public class Rotate : MonoBehaviour
{
    [SerializeField] float speedX;
    [SerializeField] float speedY;
    [SerializeField] float speedZ;
    void Update() {
        transform.Rotate(360 * speedX * Time.deltaTime, 360 * speedY * Time.deltaTime, 360 *
speedZ * Time.deltaTime); } }
```

Apoi, prin interfața Unity, am modificat viteza pe axa de rotație Y (Figura 50). Prin modificarea valorii de viteză, putem obține o rotație mai rapidă sau mai lentă a sferei.

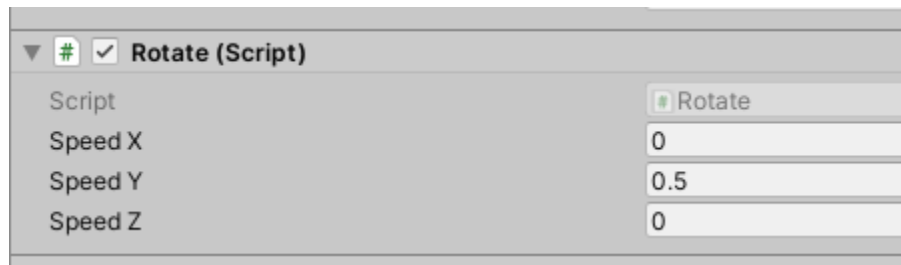


Figura 50

Pentru partea de U.I. („User Interface”, în română „Interfața Utilizatorului”), am creat un sistem logic pentru a afișa numărul de obiecte obținute, în timp real. Numărătoarea se reinițializează atunci când jucătorul pierde. Am utilizat funcționalitatea de „Trigger” / „Activare” din Unity, prin bifarea acesteia în interfața obiectului creat (Figura 51).

Acest lucru ne permite să detectăm când jucătorul intră în contact cu obiectul și să declanșăm acțiuni specifice în consecință. În cazul nostru, când jucătorul intră în contact cu obiectul de colectat, numărul de obiecte obținute se actualizează automat pe interfață. Astfel, jucătorul poate urmări progresul său în timp real și să se bucure de satisfacția de a aduna obiecte în joc.

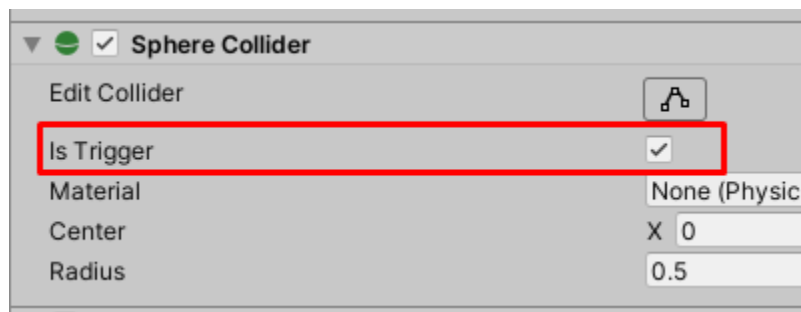


Figura 51

Urmatorul pas a fost să creez un nou script, pe care l-am numit ItemCollector / ColecteazaObiecte.

De asemenea, am avut nevoie de obiecte de tip canvas și text, pentru care am folosit librăria TextMeshProUGUI. Apoi, am setat textul inițial din interfața Unity și l-am încadrat pe selecția care a apărut a canvas-ului selectat.

```

public class ItemCollector : MonoBehaviour
{
    int coins = 0;
    [SerializeField] TextMeshProUGUI coinsText;
    [SerializeField] AudioSource collectionSound;
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Coin"))
        {
            Destroy(other.gameObject);
            coins++;
            coinsText.text = "scor:" + coins;
            collectionSound.Play();
        }
    }
}

```

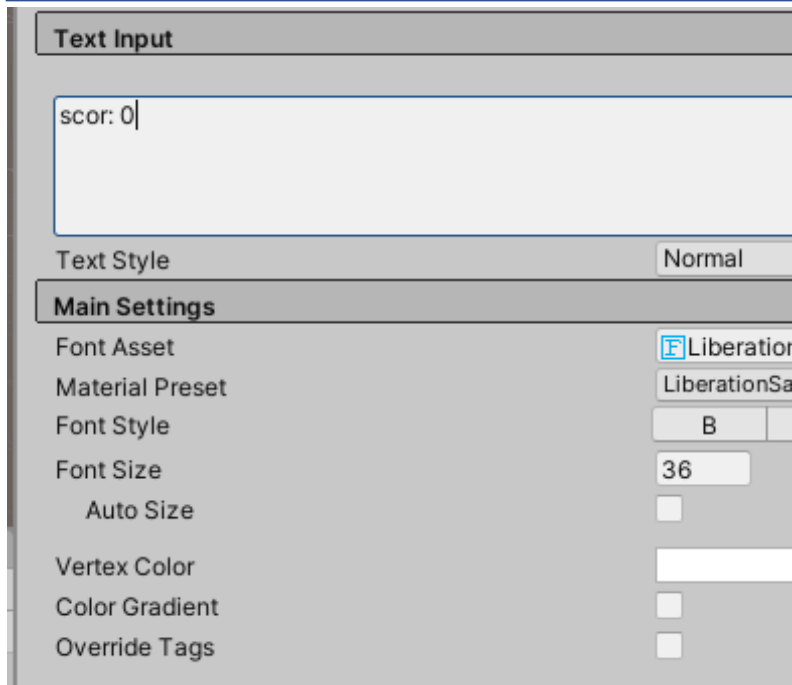


Figura 52

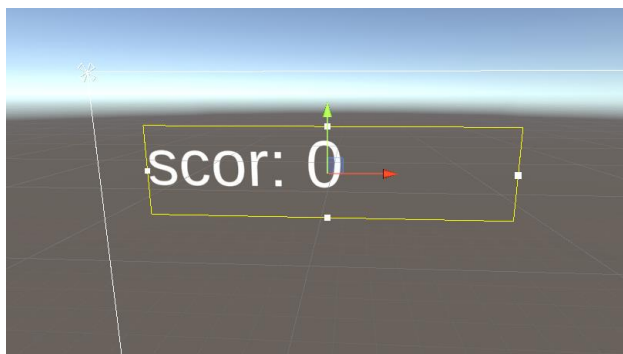


Figura 53

Dupa aceste modificări, așa arată jocul din perspectiva utilizatorului (*Figura 54 și 55*).

Prin testare, am observat că scorul crește cu succes atunci când atingem obiectele galbene.



Figura 54 – înainte de capturarea obiectelor

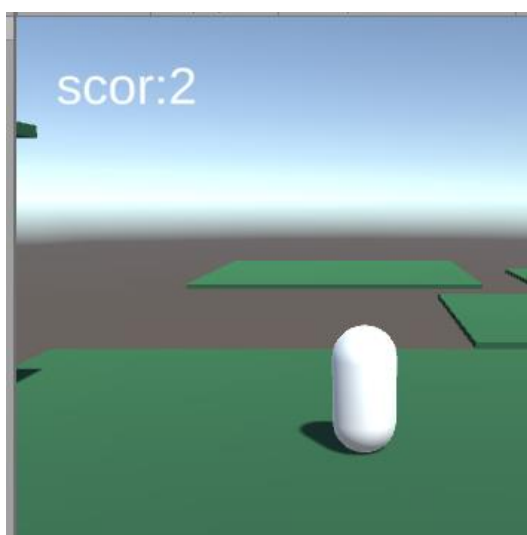


Figura 55 – după capturarea obiectelor

3.2 – Sunete și muzică

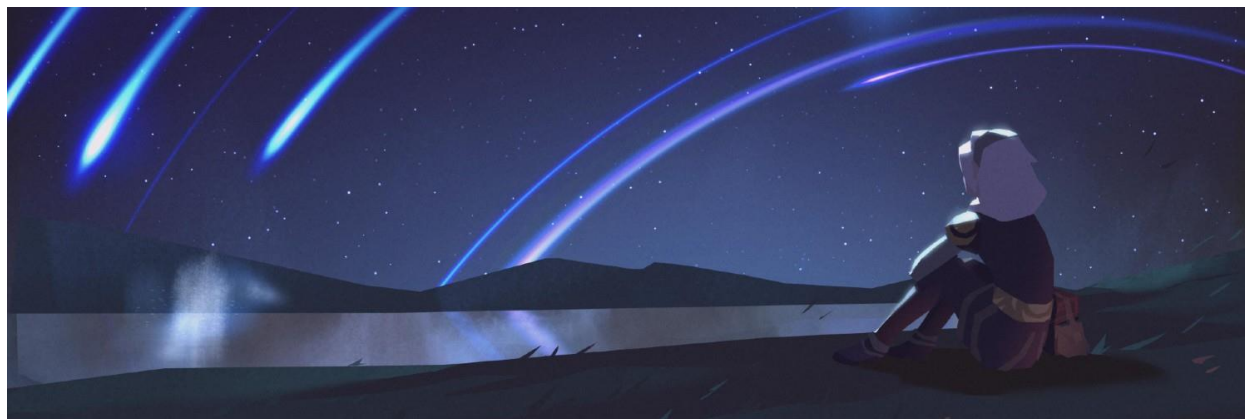


Figura 56 – Sesiuni: Diana

Ca și temă principală a atmosferei, am folosit ca inspirație „Sesiuni: Diana | O colecție pentru creatori | Riot Games Music”. Povestea este despre autocunoaștere și înțelegere, iar eu mi-am dorit să transmit asta prin sunetul și muzica jocului, care stimulează unul dintre simțurile importante prin care jocurile transmit emoții jucătorilor.

Pentru melodia de fundal, am descărcat gratuit de pe site-ul oficial <https://sessions.riotgames.com/en-us/event/sessions/> melodia „arc 4” realizată de artistul „weird inside” în colaborare cu League of Legends. Această melodie face parte din albumul „Sesiuni: Diana” și poate fi folosită de către creatori în scopuri comerciale și pentru proiecte personale.

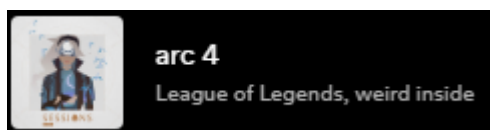


Figura 57



Figura 58

Pentru a oferi jucătorilor o experiență auditivă și mai complexă, am utilizat sunete. Descărcarea gratuită din Asset Store (magazinul de elemente) al lui Unity prin pachetul „FREE Casual Game SFX Pack” de la adresa <https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116> m-a ajutat să conectez mecanicile la sunete sugestive și să clarific acțiunile din joc.

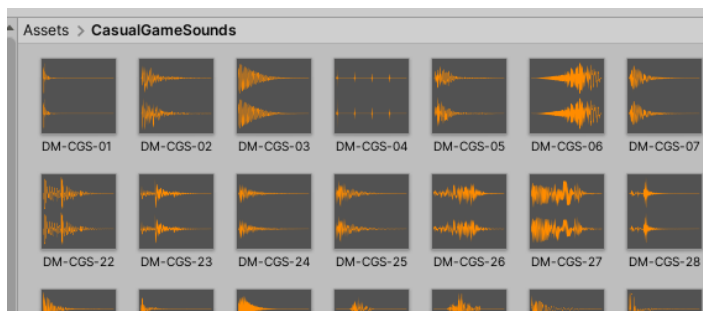


Figura 59

Pentru implementarea melodiei de fundal, am creat un obiect gol (folosit ca și folder în librărie) și l-am numit „Music” / „Muzica”, apoi i-am adăugat componentul „Audio Source”. Am atașat în rubrica „AudioClip” melodia importată anterior și am bifat caseta „Loop” pentru a porni melodia de la început atunci când se termină.

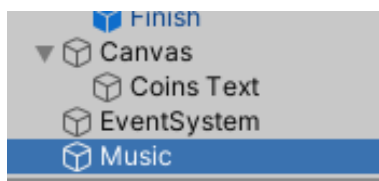


Figura 60

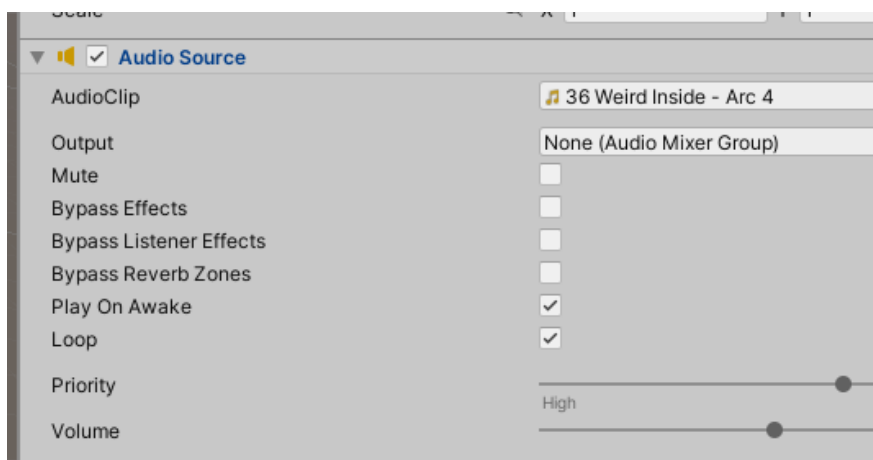


Figura 61

La sunete am procedat similar: același component l-am adăugat, însă, la diferite obiecte, în funcție de sursa acțiunii lor:

a.) sunetul săriturii: am accesat script-ul de mișcare și am adăugat următoarele linii de cod (evidențiate prin săgeți) cu definirea funcției și apelarea acesteia la săritură:

```

→ [SerializeField] AudioSource jumpSound;
...
void Jump()
{ rb.velocity = new Vector3(rb.velocity.x, jumpForce, rb.velocity.z);
→ jumpSound.Play(); }

```

Apoi, din interfața Unity a obiectului controlat de jucător, am atașat cu mouse-ul sunetul importat din magazinul Unity. De asemenea, am debifat toate casuțele și am modificat volumul și pitch-ul.

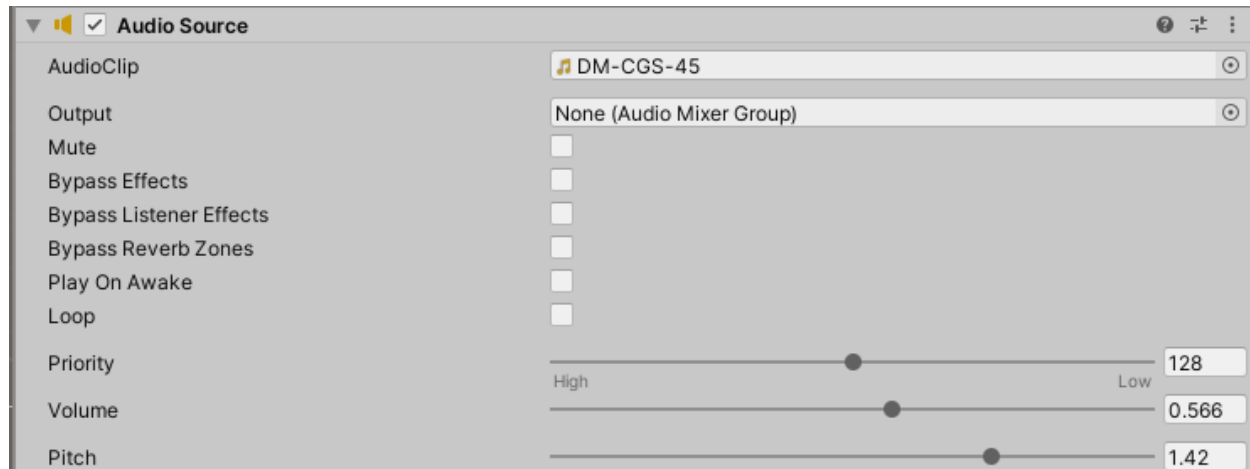


Figura 62

b.) sunetul de înfrângere: am accesat script-ul legat de viața caracterului și am adăugat următoarele linii de cod (evidențiate prin săgeți) care includ definirea unei funcții și apelarea acesteia la momentul acțiunii înfrângerii:

```

→ [SerializeField] AudioSource deathSound;
...
void Die()
{
    Invoke(nameof(ReloadLevel), 1.3f);
    dead = true;
→ deathSound.Play(); } ...

```

Urmând ca din interfața Unity, să atașez prin mouse sunetul cu numărul 10 importat din magazinul Unity. De asemenea, am debifat toate căsuțele și am modificat volumul:

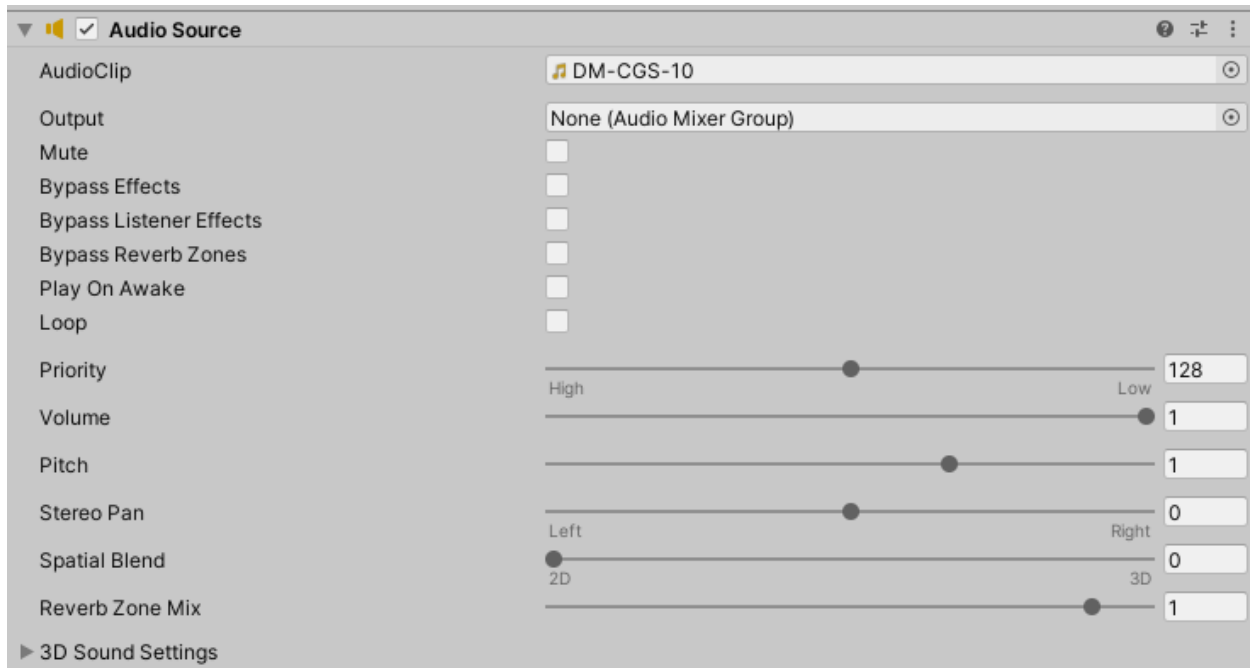


Figura 63

c.) sunetul de colectare a punctelor: am accesat script-ul legat de sistemul de colectare al punctelor și am adăugat următoarele linii de cod (evidențiate prin săgeți) care includ definirea unei funcții și apelarea acesteia la momentul acțiunii colectării (când se activează trigger-ul):

```

→ [SerializeField] AudioSource collectionSound;
...
if (other.gameObject.CompareTag("Coin"))
{
    Destroy(other.gameObject);
    coins++;
    coinsText.text = "scor:" + coins;
→ collectionSound.Play();
}

```

Am atașat componenta AudioSource și am adăugat sunetul de colectare din biblioteca Unity în interfața obiectului responsabil cu colectarea. De asemenea, am setat volumul și pitch-ul și am ajustat alte setări specifice sunetului pentru a crea o experiență auditivă plăcută pentru jucători (Figura 64).

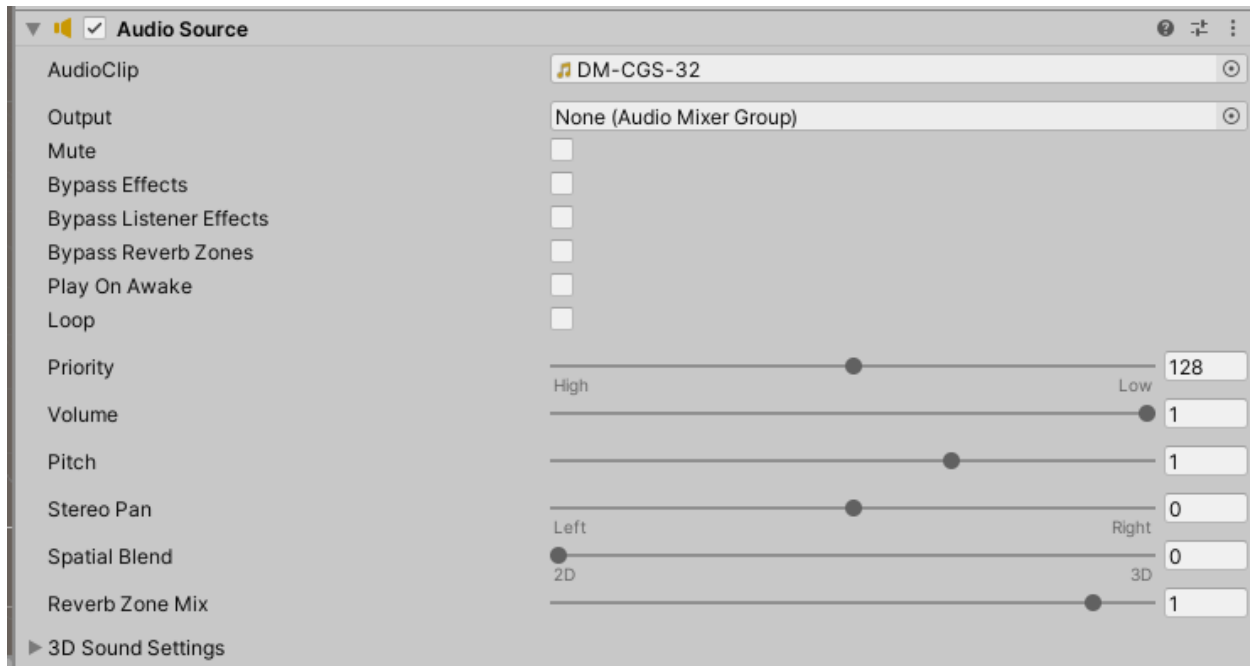


Figura 64

Muzica în jocurile video joacă un rol esențial în crearea unei atmosfere liniștitoare și captivante. Ea poate transmite emoții și poate transforma întregul mediu de joc într-o experiență memorabilă. Soundtrack-ul potrivit poate intensifica suspansul, poate crea o stare de relaxare sau poate sublinia momentele-cheie, adăugând astfel profunzime și subtilitate jocului. Muzica în jocurile video nu doar completează vizualul, ci adaugă o dimensiune unică, contribuind la conectarea emoțională a jucătorului cu universul jocului.

Din aceste motive, consider că adăugarea muzicii liniștitoare și a efectelor sonore intensifică sentimentul de confort pe care doream să îl transmit jucătorilor și, în concluzie, aceștia se pot bucura într-o mai mare măsură de atmosfera creată cu ajutorul implementării corecte a componentelor muzicale din Unity și a script-urilor.

3.3 – Elementele vizuale

Grafica în jocurile 3D are un rol esențial în a oferi experiențe de gaming de neuitat. Este un limbaj vizual prin care dezvoltatorii pot transmite emoții, pot construi lumi virtuale incredibile și pot implica jucătorii în aventuri fantastice. Fără grafica de calitate, jocurile noastre preferate nu ar avea aceeași putere de a ne teleporta în lumi ireale și de a ne provoca imaginația și inteligența.

Am descărcat pachetul AllSky Free din magazinul de elemente al lui Unity de la adresa <https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014> și am schimbat cerul lumii virtuale create pentru a oferi jucătorilor o atmosferă mai realistă.

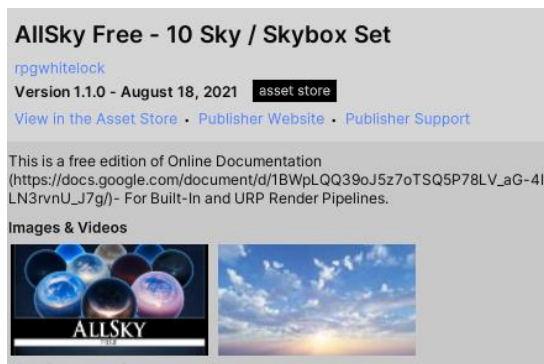


Figura 65

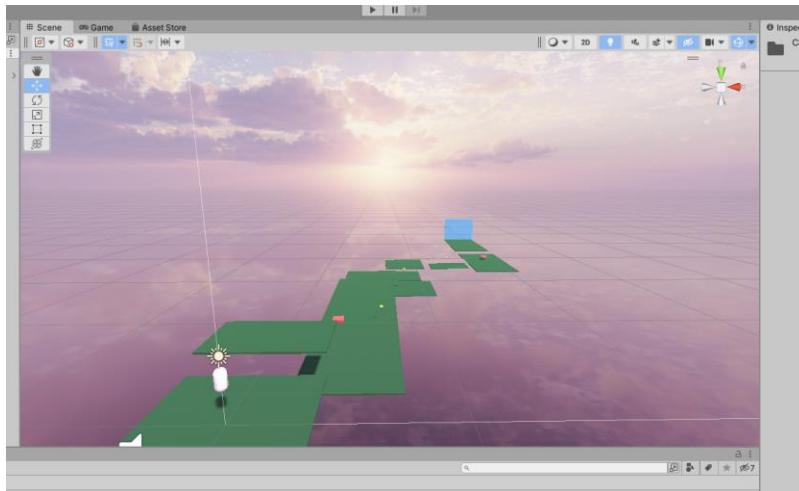


Figura 66

Următorul pas a fost să schimb culoarea platformei în mov închis, care este o culoare complementară cerului, iar culoarea caracterului principal am lăsat-o în alb. Prima platformă am colorat-o în verde pentru a crea un mic spațiu de natură.

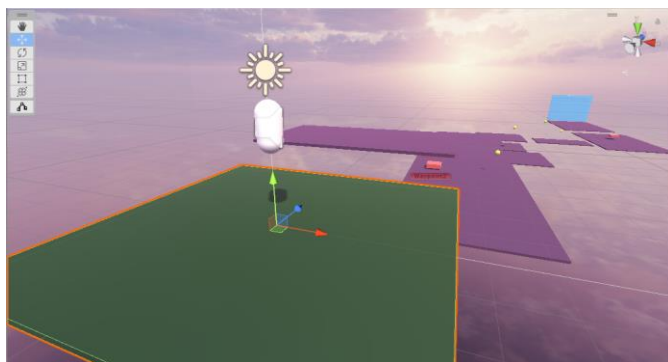


Figura 67

Cu scopul de a adăuga elemente naturale pe această platformă, am importat următorul pachet din magazinul de elemente a lui Unity de la adresa <https://assetstore.unity.com/packages/3d/environments/nature-low-poly-pack-240452>.

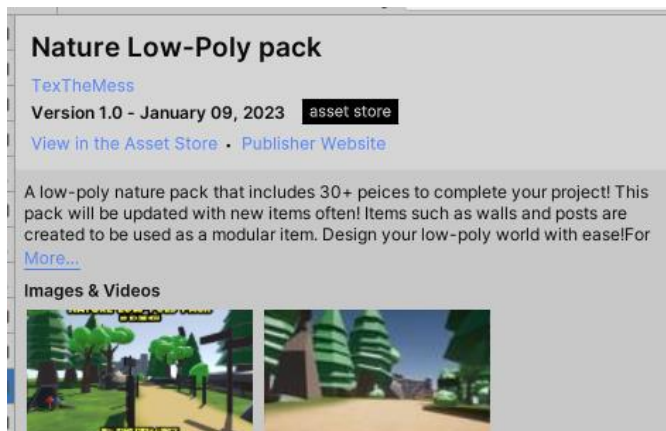


Figura 68

Am așezat obiectele din natură peste prima platformă și le-am aranjat și orientat pentru a da viață lumii, astfel:



Figura 69

3.4 – Tranziția și meniul

În ansamblu, tranzițiile și meniul în jocurile video 3D sunt esențiale pentru a asigura o interacțiune fluidă și intuitivă între jucător și joc. Aceste elemente contribuie la atmosferă, ritm și navigarea jocului, permitând jucătorilor să controleze și să personalizeze experiența lor în funcție de preferințe.

a.) Tranziția

Pentru a crea o tranziție, am început prin a adăuga la nivel un obiect cubic cu rol de punct final al nivelului. Am schimbat dimensiunile și locația lui, apoi am adăugat un material de culoare albastră pentru a-l diferenția de restul obiectelor din joc și am bifat căsuța care îl face să fie de tip trigger (care activează).

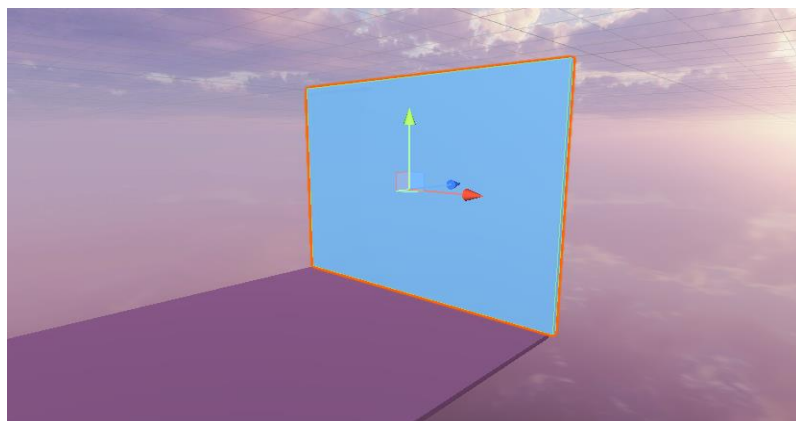


Figura 70

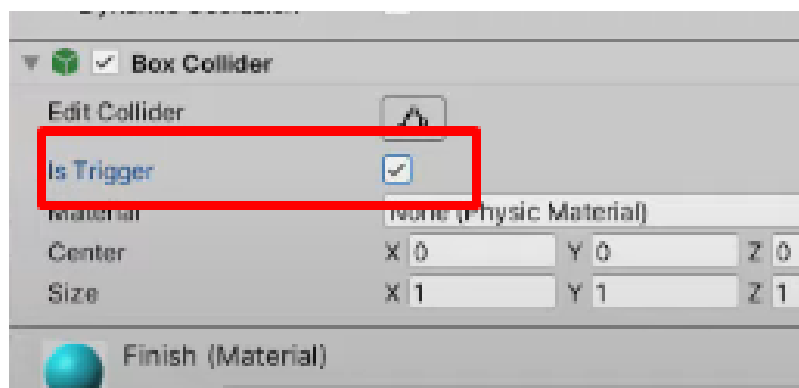
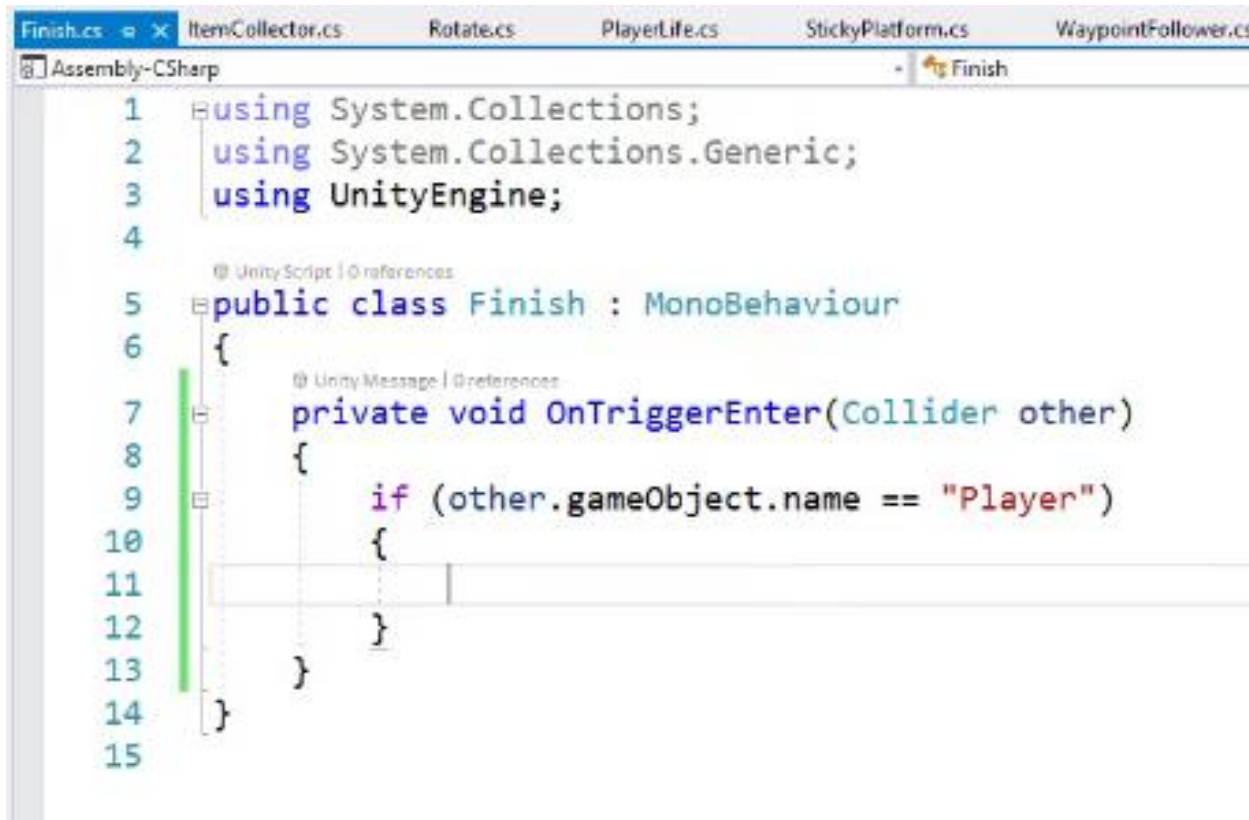


Figura 71

În continuare, am adăugat un script nou, l-am denumit sugestiv și l-am asociat obiectului recent creat. Am lăsat intenționat loc liber de scris în funcția if, la care mă voi întoarce mai încolo.



Am ajuns la pasul în care este nevoie să creez un al doilea nivel și o să fac asta prin a multiplica prima scenă la care am lucrat, dar nu înainte să salvez nivelul curent pentru a nu pierde progresul.

Pentru a deschide al doilea nivel, am dat dublu-click pe el, apoi am rearanjat nivelul, schimbând cerul și culoarea primei platforme. Am ales să fac un nivel mai ușor în ideea că îl voi putea modifica mai târziu, pentru a mă putea întoarce mai repede la implementarea tranziției.

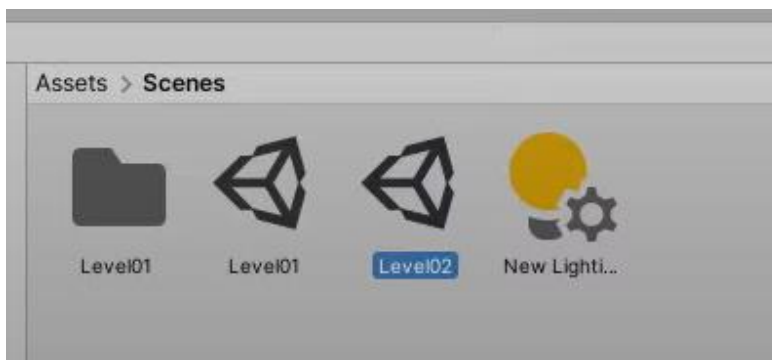


Figura 72

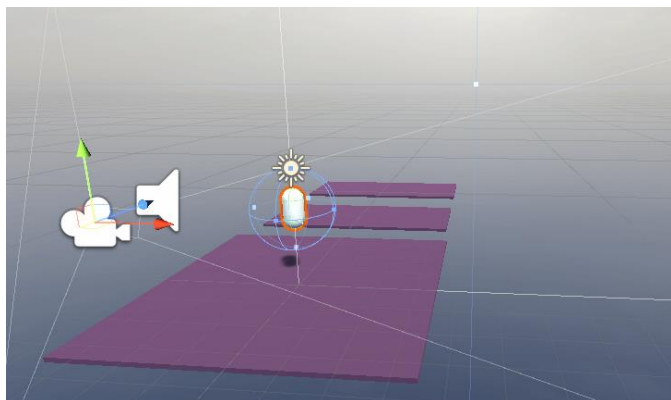


Figura 73

Din interfața Unity, am selectat nivelele pe rând, apoi le-am adăugat în build-ul curent. Am schimbat ordinea, am selectat platformele care mă interesează și am salvat modificările. Prin acest pas important, comunic cu Unity și îi spun ce nivele are jocul meu și în ce ordine sunt ele:

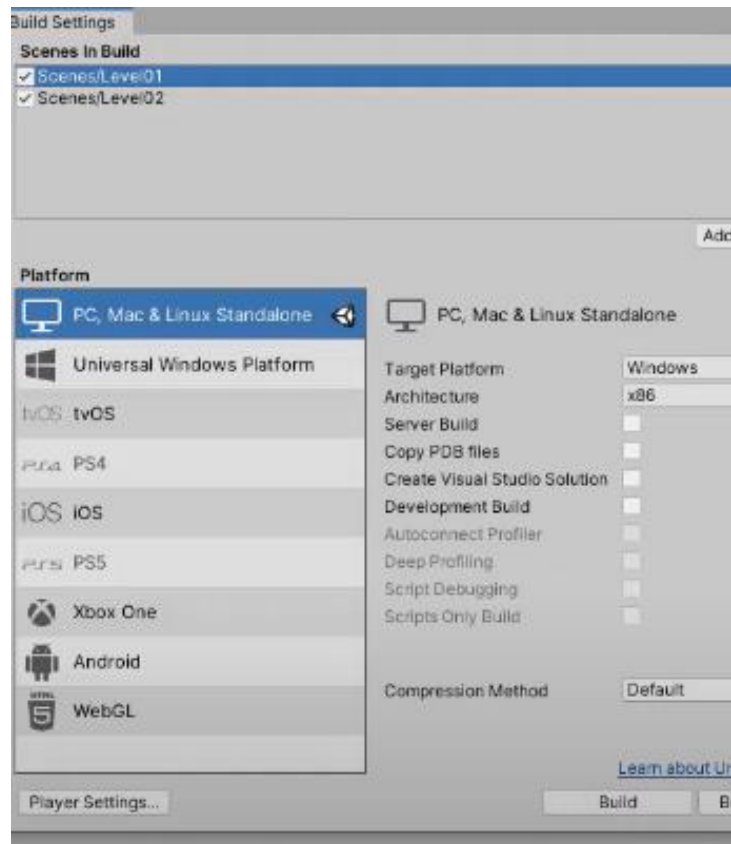


Figura 74

Acum mă pot întoarce la script-ul „Finish” să-l termin, așa că îl deschid (nu are importanță nivelul din care îl deschid) și fac referire la scenele din build-ul menționat anterior, adăugând la început biblioteca `UnityEngine.SceneManagement`. La momentul activării trigger-ului la final de nivel, încărcăm scena care urmează în ordinea selectată de noi din setările build-ului:

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class Finish : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.name == "Player")
        {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        }
    }
}

```

Pentru început apelăm funcția privată de tip void „OnTriggerEnter” cu parametrul „Collider other” la care adăugăm condiția de tip „if” și verificăm dacă obiectul cu care se atinge are numele „Player” / „Jucător”. În cazul în care această condiție este adevărată, încărcăm cu ajutorul funcției „SceneManager.LoadScene” următoarea scenă (prin incrementarea indexului scenei active cu o unitate).

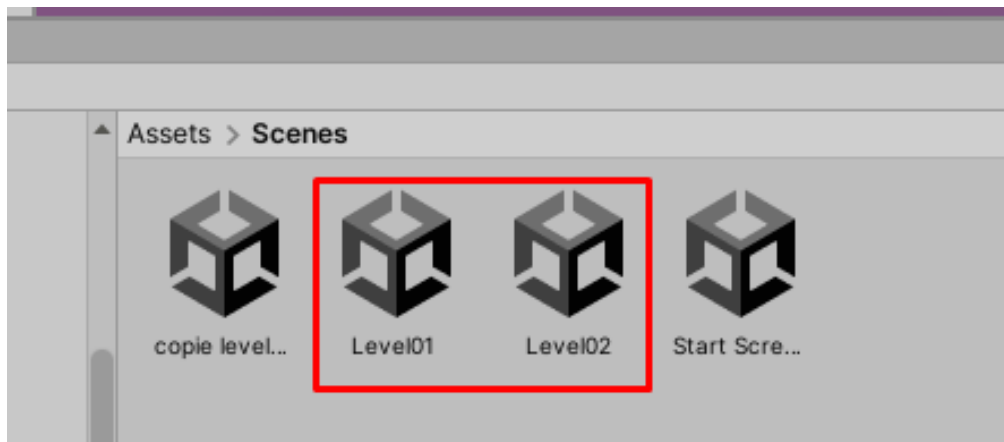


Figura 75

b.) Meniul

Încep prin a crea o scenă nouă cu scopul unui ecran de început, prin click dreapta din biblioteca Unity > Create > Scene (Figura 76).

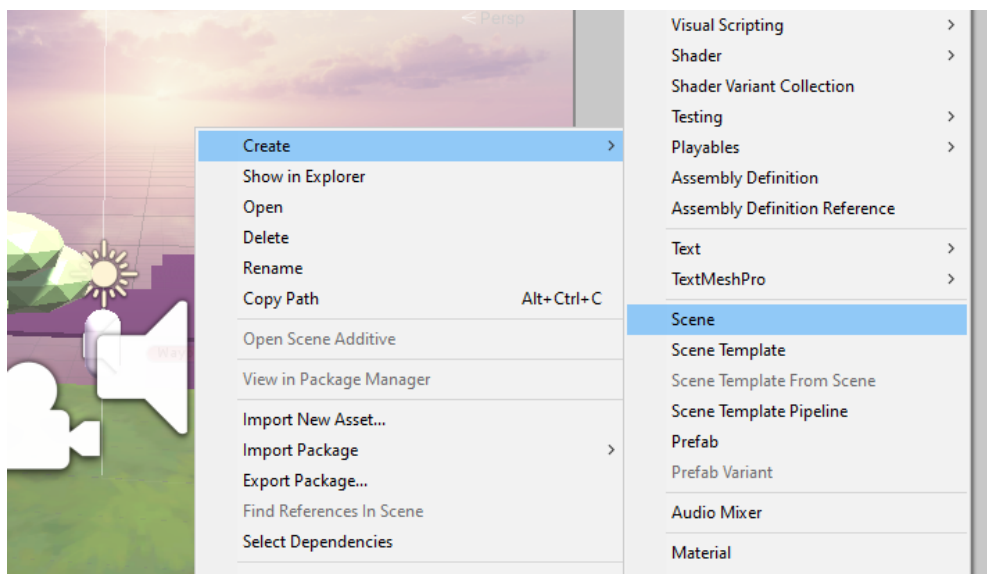


Figura 76

Redenumesc scena nouă, o selectez și o deschid, apoi creez un obiect nou de tip „Panel” și schimb proprietățile de la source image în „None” și transparența culorii la cea mai mare valoare.

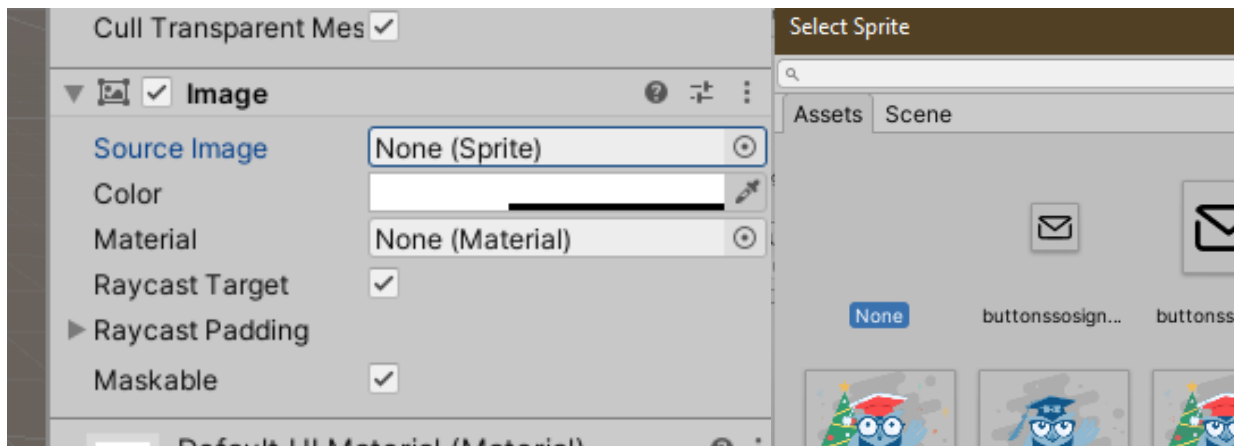


Figura 77

Pasul următor este să creez un obiect „copil” de tip Text la panel-ul menționat anterior, să introduc setul de caractere pe care vreau să apară pe pagina de pornire și să modific proprietățile sale de aliniere pentru a-l centra.

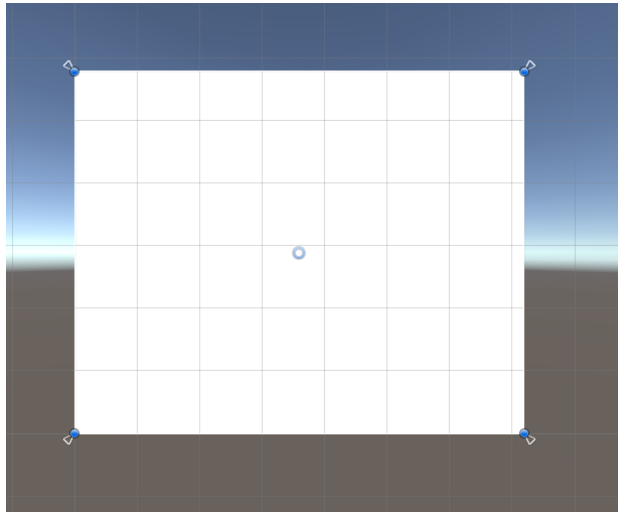


Figura 78



Figura 79

Am adăugat un nou obiect de tip „buton” din categoria de obiecte U.I. și îl setez ca și „copil” la panel. Apoi am introdus textul dorit și modific dimensiunile butonului și ale textului.

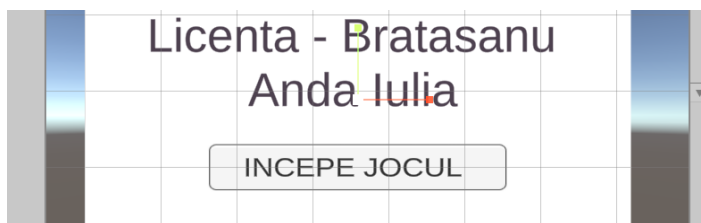


Figura 80

Pentru a face posibilă trecerea de la buton la scena următoare, o să creez un script nou și voi schimba din nou ordinea scenelor din Unity Build Settings:

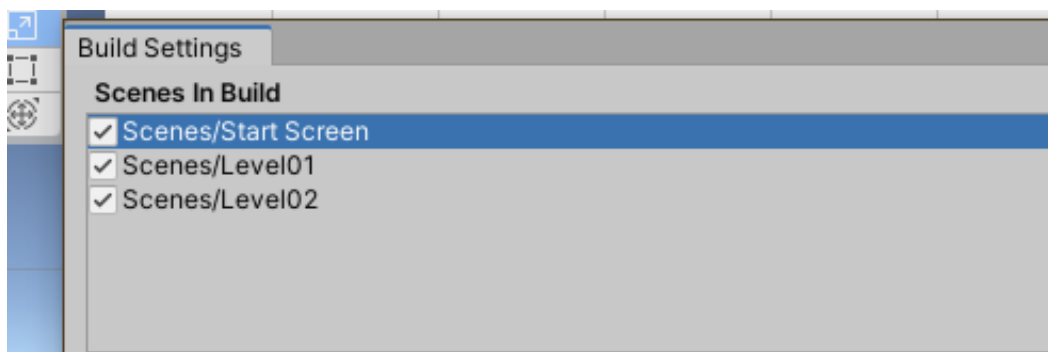


Figura 81

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class StartMenu : MonoBehaviour
{
    public void StartGame() {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}

```

Pasul final este să tragem din interfața Unity acest script la noua funcție adăugată a butonului și să selectăm funcția pe care am creat-o (Figura 82).

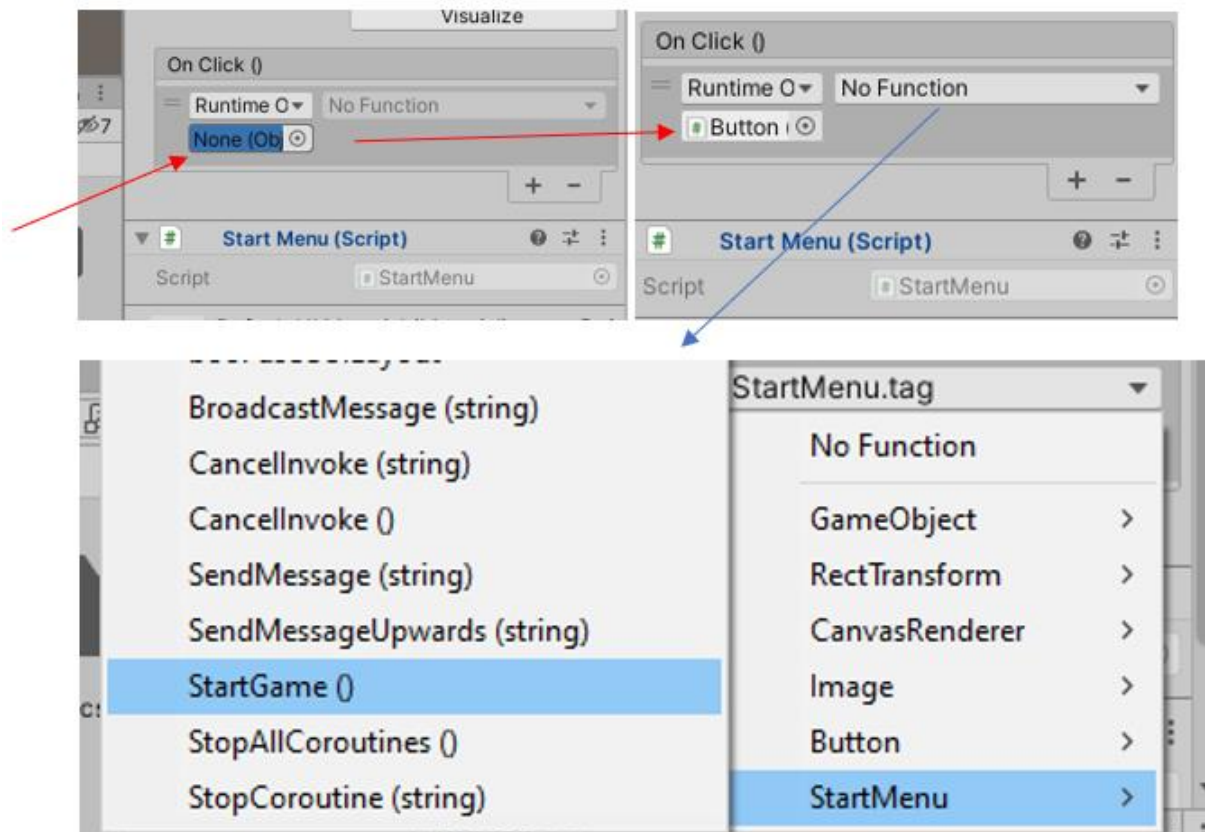


Figura 82

CAPITOLUL 4 – VERIFICAREA APLICAȚIEI

În timpul dezvoltării unui joc video, un aspect fundamental îl reprezintă testarea sa. Acest proces asigură că toate elementele jocului funcționează așa cum au fost proiectate, asigurându-se totodată că acestea oferă o experiență plăcută jucătorilor. În cadrul acestei lucrări de licență, am testat fiecare mecanică a jocului în parte, pornind de la activarea tranziției, audio, mișcarea jucătorului, până la sistemul de viață și moarte.

Pentru organizarea și monitorizarea rezultatelor testărilor, am folosit un sistem de management al ciclului de viață al aplicațiilor (ALM). Acesta m-a ajutat să gestionez și să urmăresc evoluția fiecărei funcționalități în parte, începând cu dezvoltarea sa inițială, trecând prin etapa de testare și până la implementarea finală.

Instalarea aplicației ALM

Pentru a descărca aplicația, am accesat adresa <https://www.microfocus.com/marketplace/appdelivery/content/alm-client-launcher> și am apăsă butonul "Download Newest" / "Descarcă cea nouă versiune" și mi-am făcut un cont nou.

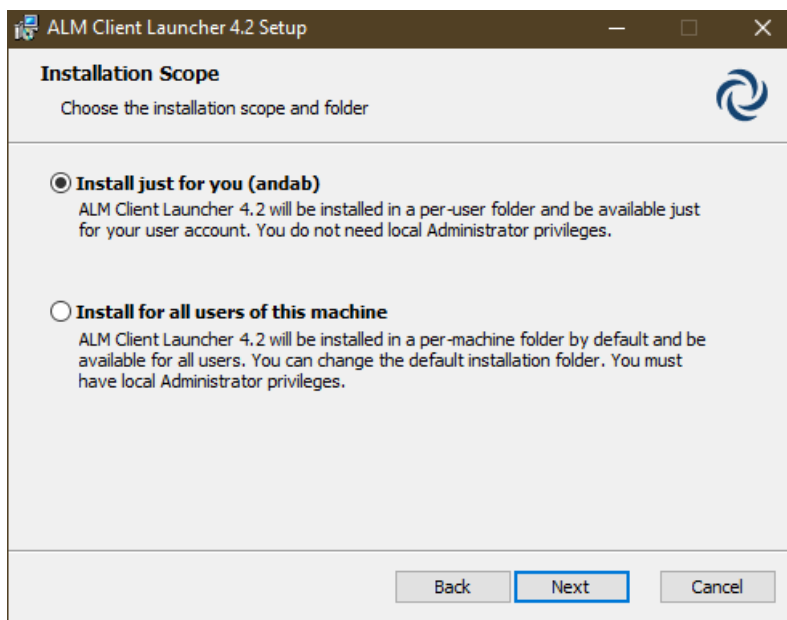


Figura 83 – Instalarea aplicației

Pasul următor a fost să accesez <http://193.230.11.105:8090/qcbin/> și să introduc contul primit de la facultate (Figura 84).

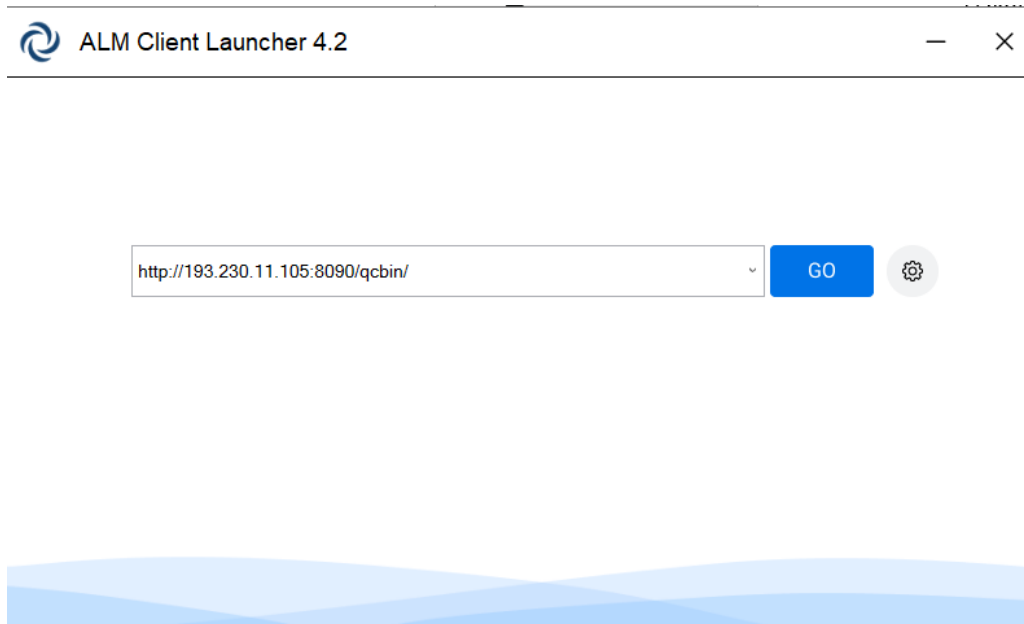


Figura 84

După ce s-a deschis aplicația ALM, am creat un folder nou cu numele meu în "Test Plan" cu patru teste noi și le-am dat numele următoare: "activarea tranziției", "audio", "player movement" (mișcarea jucătorului) și "sistemul de viață și moarte" (Figura 85).

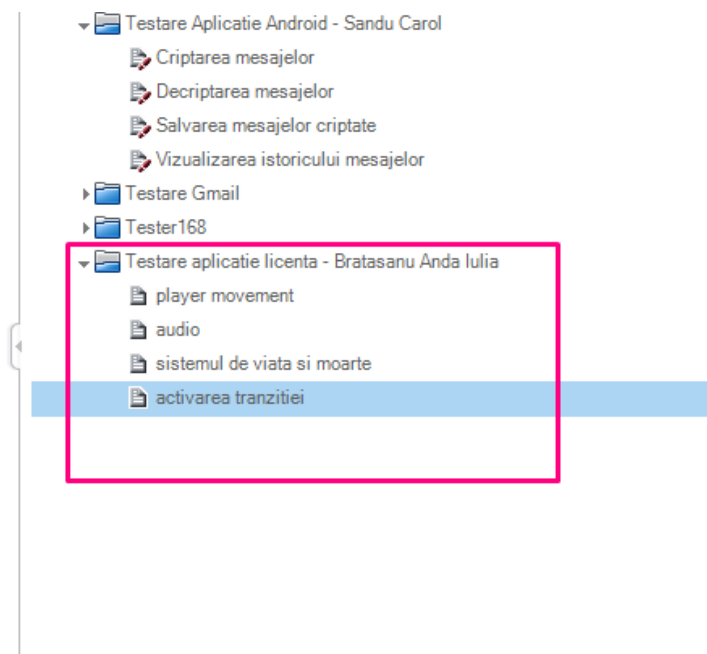


Figura 85

În continuare, în tabelul secțiunii "Test Lab", am încărcat testele create anterior pentru a putea începe procesul de verificare.

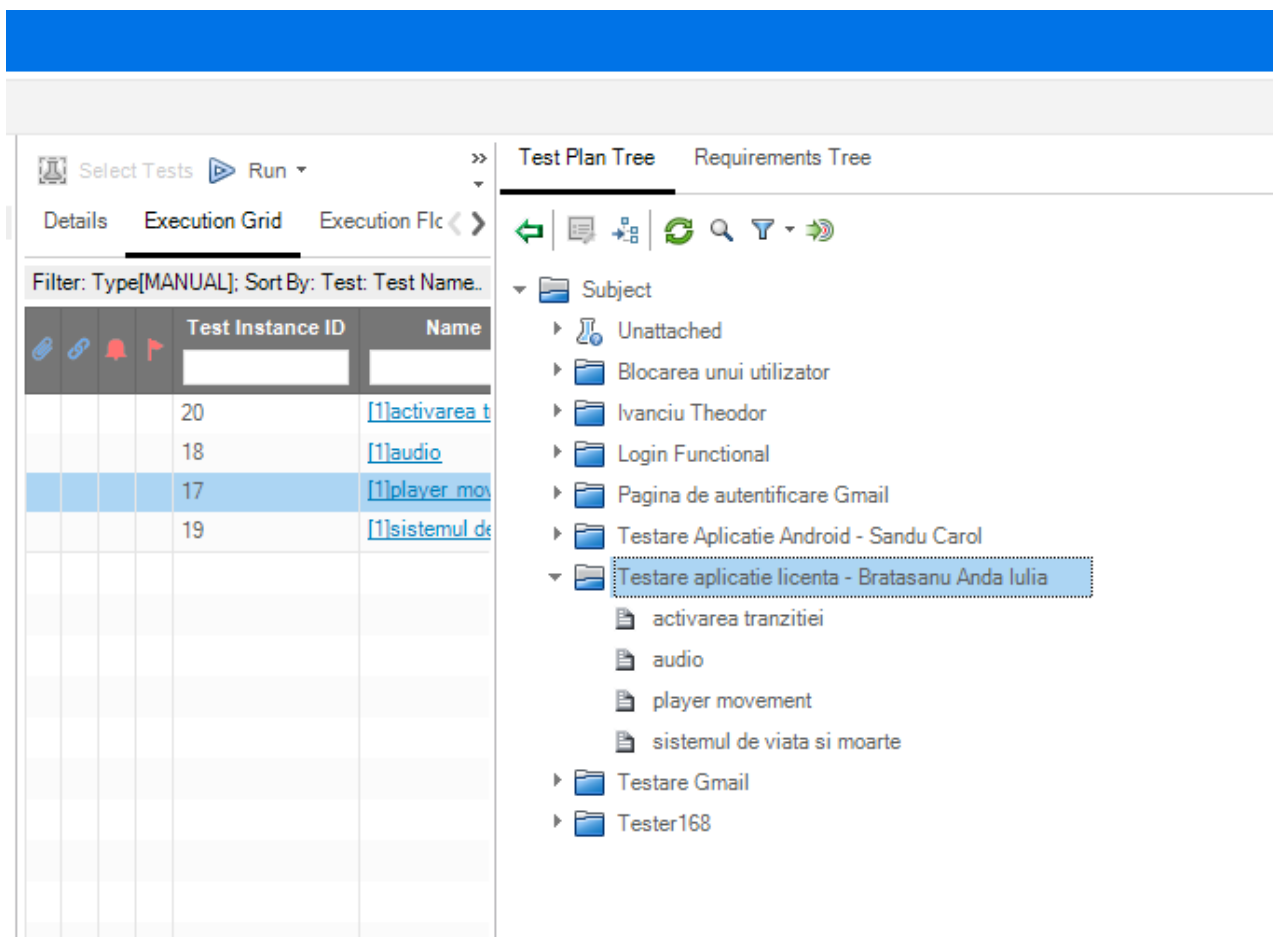


Figura 86

La început, testele aveau statusul (N/A), adică statusul nu a fost introdus încă, după cum se poate observa în Figura 87.

Test Instance ID	Name	Test: Test Name	Type	Status
			MANUAL	
28	[1]activarea tranzitiei	activarea tranzitiei	MANUAL	N/A
26	[1]audio	audio	MANUAL	N/A
25	[1]player movement	player movement	MANUAL	N/A
27	[1]sistemul de viata si moarte	sistemul de viata si moarte	MANUAL	N/A

Figura 87

Procesul de verificare

a.) Activarea Tranziției

Pentru verificarea activării tranziției între diferitele scene ale jocului, am efectuat o serie de teste. Acestea au implicat atingerea obiectelor de tip pentru a activa trecerea către următoarea scenă. Rezultatele au fost pozitive, tranzițiile fiind activate în mod corespunzător.

b.) Audio

Testarea audio a avut în vedere două aspecte: muzica de fundal și efectele sonore (SFX). Am verificat dacă sunetul de fundal se redă continuu pe parcursul scenei și dacă efectele sonore corespunzătoare se declanșează în momentul potrivit (cum ar fi atunci când personajul sare, colectează un obiect sau atinge un inamic). În urma testelor, sunetele au funcționat corespunzător.

c.) Mișcarea Jucătorului

Mișcarea jucătorului a fost testată pentru a verifica dacă comenzile răspund corespunzător. Am verificat funcționarea butoanelor pentru mers, sărit și interacțiunea cu obiectele (platformele, inamicii, punctele). După o serie de teste, s-a confirmat faptul că toate aceste mecanici funcționează corect, oferind o experiență fluidă și intuitivă.

d.) Sistemul de Viață și Moarte

Sistemul de viață și moarte a fost, de asemenea, testat. Am verificat dacă scena este resetată atunci când personajul atinge un inamic sau cade de pe platformă. Am testat și dacă imediat după ce inamicul este înfrânt, timp de câteva secunde dispare de pe hartă și nu am sesizat erori.



Sort By: Test: Test Name[Ascending]				
Instance ID	Name	Test: Test Name	Type	Status
			MANUAL	
	[1]activarea tranzitiei	 activarea tranzitiei	MANUAL	✓ Passed
	[1]audio	 audio	MANUAL	✓ Passed
	[1]player movement	 player movement	MANUAL	✓ Passed
	[1]sistemul de viata si moarte	 sistemul de viata si moarte	MANUAL	✓ Passed

Figura 88

CAPITOLUL 5 – CONCLUZII

Dezvoltarea proiectului de licență a reprezentat un drum fascinant și provocator, în egală măsură, care a permis evaluarea, compararea și înțelegerea aprofundată a diferitelor aspecte legate de proiectarea și implementarea unui joc video 3D. În ceea ce privește rezultatele și obiectivele propuse în cadrul proiectului, consider că acestea au fost atinse. Jocul îmbină cu succes elemente simple de mecanică inspirate de Mario cu o atmosferă relaxantă inspirată de Diana-sessions ale lui Riot Games, satisfăcând astfel dorința de a crea un spațiu în care utilizatorii să se poată destinde și să se bucure de frumusețea naturală.

Obiectivul inițial a fost de a dezvolta un joc relaxant, în care jucătorii să se poată retrage de la presiunile și grijele zilnice. Acest lucru a fost atins prin crearea unei atmosfere vizuale calmante, cu cerul roz și platformele mov flotante în spațiu, ce au fost menite să inducă o stare de calm și relaxare. Din punct de vedere al nivelurilor de dificultate, am decis ca primul nivel să fie mai greu, cu scopul de a stimula interesul jucătorilor și de a le oferi o provocare înainte de a trece la al doilea nivel, mai puțin provocator.

Procesul de dezvoltare a jocului mi-a permis nu doar să-mi aplic competențele tehnice acumulate pe parcursul studiilor de informatică, ci și să-mi îmbunătățesc și să-mi dezvolt noi abilități. Am avut oportunitatea de a îmbina pasiunea mea pentru artă cu aptitudinile tehnice, fapt ce mi-a permis să creez o atmosferă vizuală unică în cadrul jocului. În plus, dezvoltarea acestui joc a fost și un proces de învățare semnificativ. Am învățat despre importanța detaliilor fine și a atenției acordate detaliilor în design-ul grafic, precum și cum acestea pot influența starea jucătorilor și pot contribui la experiența globală a jocului.

De asemenea, am înțeles mai bine cum funcționează echilibrul dintre dificultate și relaxare într-un joc video și cum acesta poate fi ajustat pentru a satisface diferite tipuri de jucători. Am acumulat și experiență în ceea ce privește gestionarea timpului și a resurselor, un aspect esențial în orice proiect de dezvoltare de software. Crearea unui joc video nu constă doar în programare și design, ci implică și planificare, stabilirea de obiective și priorități și monitorizarea constantă a progresului.

Acest proiect m-a pregătit nu doar pentru a înfrunta provocările tehnice care mă așteaptă în cariera profesională, ci m-a ajutat să înțeleg cât de puternică poate fi intersecția dintre artă și tehnologie. În plus, am învățat valoarea feedback-ului și a testării continue.

BIBLIOGRAFIE

1. Unity Technologies. (s.d.). Documentația Unity. Obținută de la <https://docs.unity3d.com/>
2. Whitaker, R. B. (2022). The C# Player's Guide (5th Edition). Paperback – January 14, 2022
3. Salen, K., & Zimmerman, E. (2004). Regulile jocului: Fundamentele designului de jocuri. Cambridge, MA: MIT Press.
4. Smith, J., Johnson, A., & Brown, L. (2022). Tendințe emergente în jocurile de realitate virtuală: o analiză a cercetărilor actuale. Journal of Gaming Studies, 8(2), 145-162.
5. Whitaker, R. B. (2019). Începerea cu Unity. Gamasutra. Recuperat de la: https://www.gamasutra.com/blogs/RBWhitaker/20190304/338263/Getting_Started_with_Unity.php
6. [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))
7. <https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/>
8. https://en.wikipedia.org/wiki/Visual_Studio
9. <https://learn.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/using-visual-studio-tools-for-unity?pivots=windows>
10. <https://learn.unity.com/>
11. <https://unity.com/how-to/beginner/10-game-design-tips-new-developers>
12. <https://www.javatpoint.com/unity-components>
13. <https://www.makeuseof.com/player-movement-in-unity-explained/>
14. <https://gamedevacademy.org/unity-start-menu-tutorial/>
15. <https://www.psu.com/news/why-graphics-are-so-important-in-gaming/>
16. <https://explodingtopics.com/blog/gaming-trends=>
17. <https://medium.com/nerd-for-tech/moving-platforms-in-unity-86d0c6f05d85>