

**Universitatea Tehnică a Moldovei**

**DEZVOLTAREA UNUI JOC 3D: ,,LUMEA SĂLBATICĂ”  
DEVELOPMENT OF A GAME 3D: "WILD WORD"**

|  |  |
| --- | --- |
| **Studenți:** | **gr.TI-231, TI-235  Basoc Cristian,Calaev Vitalic, Munteanu Serafim** |
| **Coordonator:** | **Andrievschi-Bagrin Veronica, Coser Catalin,Daria Cebotari asist.univ.** |

**Chişinău, 2024**

Cuprins

[Abrevieri și Definiții 3](#_Toc183679702)

[Introducere 4](#_Toc183679704)

[1 Analiza dimeniului de studiu 5](#_Toc183679705)

[1.1 Actualitatea temei 5](#_Toc183679706)

[1.2 Algoritmii jocului 6](#_Toc183679707)

[1.3 Argumentarea alegerii algoritmului 9](#_Toc183679708)

[1.4 Cauza alegerilor algoritmilor pentru solutionarea problemei 10](#_Toc183679709)

[2 Jocuri similare cu proiecte realizate 12](#_Toc183679710)

[2.1 Scopul și obiectivele propuse 13](#_Toc183679711)

[3 Realizarea jocului 13](#_Toc183679712)

[3.1 Implementarea algoritmului 15](#_Toc183679713)

[3.2 Interfața grafică a jocului 18](#_Toc183679714)

[3.3 Analiza rezultatelor obținute 19](#_Toc183679715)

[Concluzie 21](#_Toc183679716)

[Bibliografie 22](#_Toc183679717)  
[Anexa A 23](#_Toc183679717)

[Anexa B 25](#_Toc183679717)

[Anexa C 26](#_Toc183679717)

[Anexa D 28](#_Toc183679717)

**Abrevieri și Definiții**  
  
NPC-uri - se referă la personaje dintr-un joc video care nu sunt controlate de jucători, ci de inteligența artificială (IA) sau de scripturile jocului.  
Mobs - este un termen utilizat frecvent în jocurile video, pentru a se referi la creaturi sau entități dinamic animate care interacționează cu jucătorul sau cu lumea jocului.  
Vs - este o abreviere de la „versus”, un termen latin care înseamnă „împotriva” sau „contra”.  
gameplay - este un termen folosit pentru a descrie experiența generală a jucătorului într-un joc video.   
Framework - sunt structuri software predefinite care oferă un set de instrumente, reguli și funcționalități pentru a facilita dezvoltarea de aplicații, programe sau sisteme.  
Entertainment - reprezintă orice activitate, experiență sau mijloc creat pentru a oferi plăcere, recreere sau relaxare oamenilor.  
Post-Processing Stack - se referă la un set de efecte vizuale aplicate la finalul procesului de randare a unei scene 3D, înainte de afișarea imaginii pe ecran.

# **Introducere**

# Într-o eră în care opțiunile de divertisment sunt numeroase, dar adesea repetitive și lipsite de profunzime, tot mai mulți oameni își doresc o experiență captivantă care să le provoace inteligența, să le testeze limitele și să le ofere un sentiment autentic de realizare. Jocurile de supraviețuire reprezintă răspunsul perfect la această nevoie, oferind oportunități unice de imersiune și decizii semnificative într-un context virtual provocator. Pentru a răspunde acestei cereri crescânde, ne-am propus să dezvoltăm o platformă de promovare și un joc inovator, bazat pe o lume realistă, care să capteze atenția jucătorilor și să le ofere o experiență memorabilă. Printr-un proces de cercetare și analiză aprofundată a pieței de jocuri de supraviețuire, am identificat tendințele actuale, opiniile jucătorilor despre titluri consacrate și mecanismele care determină succesul unui joc captivant. Aceste informații ne-au ghidat în construirea unui concept unic, menit să satisfacă dorințele utilizatorilor.

# 

# 

# **1 Analiza dimeniului de studiu**

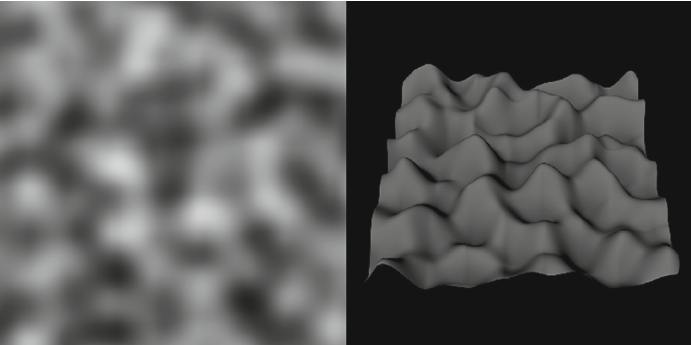
Jocurile de supraviețuire au câștigat o popularitate considerabilă, atrăgând jucători pasionați   
de provocări autentice și experiențe captivante, unde resursele limitate și deciziile strategice sunt esențiale. Această nișă din industrie oferă oportunități semnificative de inovare, combinând mecanici precum gestionarea resurselor, explorarea și supraviețuirea împotriva factorilor externi.   
 Un element central al jocurilor de supraviețuire este conceptul de bază: supraviețuirea într-un mediu ostil. Fie că este vorba de scenarii apocaliptice sau de medii naturale sălbatice, aceste jocuri testează capacitatea jucătorilor de a se adapta, de a lua decizii rapide și de a găsi soluții creative în condiții dificile. Aceasta implică gestionarea resurselor rare, explorarea mediilor periculoase și confruntarea cu pericole dinamice, de la animale sălbatice până la alți jucători în modul multiplayer.   
 Tehnologiile de ultimă generație joacă un rol esențial în crearea unei experiențe captivante și imersive. Motoarele grafice moderne, precum Unity, permit construirea unor lumi virtuale realiste, cu grafică detaliată și mecanici fluide. Un alt factor cheie este utilizarea inteligenței artificiale (IA). Comportamentul dinamic și imprevizibil al animalelor și al NPC-urilor (personaje non-jucătoare) oferă jucătorilor un sentiment de realism. IA este esențială pentru a crea interacțiuni care simulează situații reale, reducând elementele pre-scenariate și crescând nivelul de dificultate și imersiune. În plus, tehnologia multiplayer aduce o dimensiune socială jocurilor de supraviețuire, încurajând cooperarea sau competiția directă între jucători.   
 O analiză detaliată a domeniului jocurilor de supraviețuire evidențiază oportunitățile de a crea un joc captivant, bazat pe realism, mecanici bine gândite și tehnologii moderne. Jocurile de supraviețuire reușesc să îmbine acțiunea cu strategia și explorarea, oferind o experiență intensă și memorabilă. Prin integrarea unor elemente inovatoare și utilizarea resurselor tehnologice disponibile, un astfel de joc poate deveni o experiență de referință în această nișă competitivă. **1.1 Actualitatea temei**  
 Jocurile de supraviețuire au devenit astăzi un gen destul de mare în jocurile video și, pe bună dreptate, oferă un amestec captivant de provocare, strategie, control asupra mediului înconjurător, atâta profunzime pe care doriți să o accesați pentru imersivul. sentiment care vine doar din experimentarea unor obiective semnificative auto-stabilite. Unele dintre aceste jocuri sunt cele mai jucate și transmise în direct platforme de jocuri, cum ar fi Twitch[1], YouTube[2] sau chiar TikTok[3]. De la jocurile clasice precum Minecraft[5] și Rust[7], care au atras milioane de jucători activi, până la cele mai noi, cum ar fi Valheim[6], supraviețuirea a fost un gen în continuă schimbare, cu opțiune pentru diferite experiențe.  
 Magia jocurilor de supraviețuire este înțelegerea perpetuă a incertitudinii și a pericolului, ca în post-apocalipsă, natură sălbatică sau război pentru o resursă valoroasă deținută de jucători. Gestionarea resurselor și strângerea, fabricarea, construirea de adăposturi pe timp de noapte pentru supraviețuirea împotriva faunei sălbatice periculoase atât pe uscat, cât și sub apă sunt toate elementele încorporate în joc care utilizează cât de bine poate un jucător să gestioneze aceste aspecte pentru a reuși. Dar, de asemenea, jocurile de supraviețuire răsplătesc jucătorul pentru că le-a dat controlul: jucând într-o lume deschisă și creându-ne propriul traseu oferă spațiu pentru explorare fără limitări, putem înțelege cum funcționează o lume vastă și putem dezvolta moduri unice de a rămâne în viață.  
 Un alt factor care contribuie la popularitatea jocurilor de supraviețuire este creșterea comunității care evolueaza în jurul acestor jocuri. Platformele de streaming și forumurile de discuții sunt pline de sfaturi, ghiduri și experiențe împărtășite de jucători. Astfel, fiecare joc de supraviețuire ajunge să aibă o comunitate activă și loială care contribuie la succesul și longevitatea acestuia. În multe cazuri, aceste comunități creează și moduri care extind funcționalitățile jocului sau adaugă noi elemente, cum ar fi noi resurse, inamici sau instrumente de construire.

De asemenea, jocurile de supraviețuire sunt subiecte populare pe canale de streaming și YouTube[2], unde creatorii de conținut își documentează experiențele, strategii de joc și încercările de a supraviețui în cele mai dificile condiții. În mod natural, aceste activități nu fac decât să atragă și mai mulți jucători și să mențină interesul pentru genul supraviețuirii.

**1.2 Algoritmii jocului**

Algoritmi de generare aleatorie a hărților (Random Map Generation)[9]:

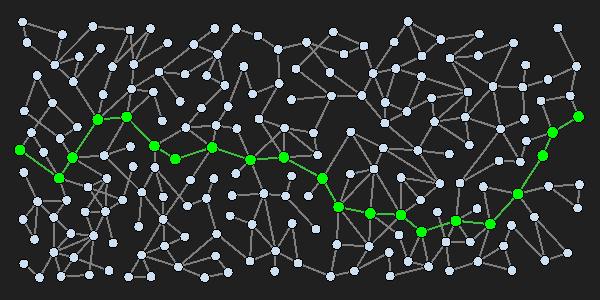
În calculatoare, Procedural Generation este o metodă de creare a datelor algoritmic, spre deosebire de manual, de obicei printr-o combinație de conținut și algoritmi generați de oameni, cuplate cu aleatorie și putere de procesare generate de computer. În grafica computerizată, este folosit în mod obișnuit pentru a crea texturi și modele 3D. În jocurile video, este folosit pentru a crea automat cantități mari de conținut într-un joc. În funcție de implementare, avantajele generării procedurale pot include dimensiuni mai mici ale fișierelor, cantități mai mari de conținut și caracter aleatoriu pentru un joc mai puțin previzibil.



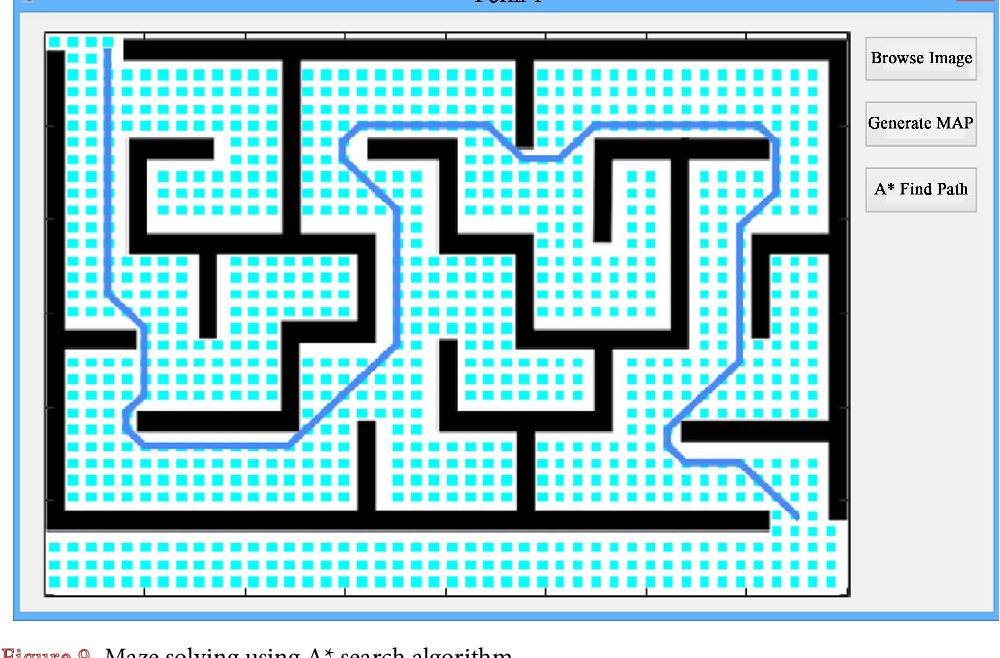
**Figura 2.1 – Noise Map**

1. Algoritmi pentru mob-uri (Mobs):

Pathfinding[11] este un nod de trasare pentru a găsi calea cea mai scurtă sau minimă între două puncte, care este de la sursă la destinație printr-o aplicație computerizată. Algoritmii de găsire a căii sunt utilizați pentru ca jucatorul sau personajul non-jucător să găsească o cale între punctul de origine și punctul obiectiv. Pathfinding[11] este una dintre cerințele pentru a crea un personaj realist non-jucător într-un joc video. Un joc video poate fi distractiv, mai ales atunci când personajul care nu este jucător este suficient de realist. Cu toate acestea, principala problemă pentru un joc video este necesitatea unei căi optime pentru personajele nejucatoare. Pathfinding este implementat în orice condiție, cum ar fi statică, dinamică și în timp real. Prima este identificarea statică a căii în care nodul țintă, adică jucătorul nu se mișcă. Al doilea mod este căutarea dinamică sau în timp real, unde nodul țintă, jucătorul, se mișcă liber și aleatoriu pe mapă.



**Figura 2.2 – Pathfinding**



**Figura 2.3 – Pathfinding**

c) Algoritm pentru gestionarea resurselor sau crafting:  
 Algoritmul greedy pentru crafting selectează, pas cu pas, cea mai bună rețetă disponibilă în funcție de un criteriu (de exemplu, consum minim de resurse sau valoare maximă a obiectului). La fiecare pas, verifică dacă există suficiente resurse pentru rețeta aleasă, o execută dacă este posibil și actualizează resursele rămase. Procesul continuă până când nu mai există resurse suficiente pentru nicio rețetă, luând mereu decizia locală cea mai avantajoasă, fără a garanta soluția optimă globală.

**1.3 Argumentarea alegerii algoritmului**  
 Crearea unui joc de supraviețuire 3D implică integrarea mai multor sisteme algoritmice pentru a simula lumea, mecanicile și interacțiunea jucătorului cu mediul. Exemplu cum algoritmii greedy, pathfinding-ul A\* și Noise Map contribuie la acest tip de joc,conform tabelului [1.1]:

a) Tehnici de generare aleatorie a hărților:

1) Odată cu generarea de hărți procedurale, în special metodele specifice aleatorii își obțin soluțiile corecte, ceea ce este fiecare un joc unic pentru el însuși și nu trebuie să fie scris în prealabil în fiecare sesiune de joc, obținând crearea de fișiere refuzate din cauza generațiilor de conținut dinamic.  
 2) Metoda de generare a hărților poate avea un impact mare asupra complexității și calității nivelurilor din jocul dvs.

b) Algoritmi pentru Pathfinding of Mobs:

1) Pathfinding este esențială pentru a oferi NPC-urilor un comportament realist și adaptativ,unul dintre cei mai folosiți algoritmi, deoarece calculează eficient cea mai scurtă cale de pe o hartă, putându-se adapta la obstacole.  
 2) Precizie și eficiență: algoritmul A calculează calea cea mai scurtă între punctele sursă și destinație, economisind timp de procesare.

3) Adaptabilitate la schimbările de mediu: Prin ajustarea în timp real a costurilor nodurilor, A poate replanifica traseul pentru a evita noi obstacole. aspectul este important într-un joc dinamic în care NPC-urile trebuie să răspundă la aceste schimbări de pe hartă.

c) Algoritm pentru gestionarea resurselor sau crafting:  
 1) Crafting-ul este esențial în jocurile de supraviețuire, iar un algoritm greedy poate fi folosit pentru gestionarea crafting-ului și utilizarea resurselor.  
 2) Simplificare a complexității: Un algoritm greedy permite selecția locală a celei mai bune opțiuni de crafting, fără a evalua toate combinațiile posibile,acest lucru este crucial pentru un gameplay fluid, evitând timpi lungi de procesare.

3) Flexibilitate: Sistemul poate fi ajustat cu priorități diferite pentru obiecte, resurse limitate sau crafting condiționat de rețete.

4) Imersiune: Jucătorul poate experimenta presiunea resurselor limitate, având libertatea de a decide ce să prioritizeze (unelte, arme, adăpost).

**Tabelul 1.1 – Avantaje si dezavantaje ale algoritmilor**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Algoritm** | | |  | | --- | | Avantaje | | |  | | --- | | Dezavantaje | |
| Generare Aleatorie a Hărților (Perlin Noise) | Creează terenuri naturale și fluente. Ofere o varietate mare și rejucabilitate. Consum redus de memorie. | Necesită parametri ajustați pentru rezultate optime.  Poate fi dificil de controlat în structuri fixe (ex. clădiri). |
| Pathfinding - Hierarchical Pathfinding | Reduce încărcarea procesorului pentru hărțile mari. Îmbunătățește eficiența pentru jocurile complexe. | Necesită divizarea hărții în regiuni, proces suplimentar de implementare. Complexitate mai mare în implementare. |
| Greedy | Ideal pentru crafting simplu, funcționează bine cu rețete puține și resurse limitate. | Lipsă de previziune, poate consuma resurse esențiale pentru obiecte mai puțin importante. |

**1.4 Cauza alegerilor algoritmilor pentru solutionarea problemei**  
 Algoritmii de căutare și optimizare, precum A\*, Dijkstra și tehnici Greedy, reprezintă piloni fundamentali în informatică și matematică, având aplicații extinse în navigație, inteligență artificială, grafuri și optimizare. Acești algoritmi sunt utilizați pentru a rezolva probleme complexe, cum ar fi găsirea celui mai scurt drum între două puncte, generarea de hărți sau optimizarea resurselor, fiecare abordând problema dintr-o perspectivă diferită.  
a) Perlin Noise vs. Simplex Noise

1.Calitatea texturii:  
 Perlin Noise produce hărți naturale, ceea ce este esențial în cazul aplicațiilor unde aspectul realist este important (de exemplu, terenuri pentru jocuri),iar în schimb, Simplex Noise este mai potrivit pentru texturi mai fluide, dar este mai puțin favorabil atunci când e nevoie de detalii specifice.

2.Complexitate computațională:  
 Deși Perlin Noise este mai lent datorită interpolării gradientelor, avantajul său constă în rezultatul artistic superior, ceea ce îl face o alegere ideală pentru proiecte de dimensiuni mici sau medii.

b) Greedy vs. Programare Dinamică

1) Abordare:  
 Algoritmul Greedy prioritizează soluțiile locale la fiecare pas, ceea ce îl face foarte eficient pentru probleme simple unde soluția optimă poate fi garantată (ex. probleme cu structură greedy-friendly). Spre deosebire de programarea dinamică, care necesită descompunerea problemei în subprobleme și un efort mai mare, Greedy este favorizat acolo unde rapiditatea contează mai mult decât perfecțiunea soluției globale.

2) Complexitate:  
 Greedy este mai rapid, având complexitate de obicei, conform formulei  
(1.1) se calculează parcurgerea tuturor datelor de intrare o singură dată:  
 O(n) (1.1)  
sau  în comparație cu programarea dinamică, care are operațiuni suplimentare, cum ar fi sortarea, care adaugă un cost logaritmic conform formulei (1.2) unde se calculează:  
 O(n log n) (1.2)  
c) Compararea A cu Dijkstra\*

1)Abordare și Euristică:  
 Algoritmul A\* folosește o funcție de evaluare conform formulei (1.3)   
unde se calculează:   
f(n)=g(n)+h(n)f(n)=g(n)+h(n) (1.3)   
unde este costul real al drumului de la nodul de start la nodul curent la destinație.  
Acest lucru face A\* mai direcționat și mai eficient decât Dijkstra atunci când  
euristica este bine proiectată.

2)Dijkstra:   
 Algoritmul Dijkstra calculează costul minim folosind doar (costul real până la un nod) și explorează toate nodurile în funcție de costul lor, fără a lua în considerare destinația, acest lucru îl face mai puțin eficient în probleme de căutare direcționată.

# **2 Jocuri similare cu proiecte realizate**

În primul pas am analizat jocurile de supraviețuire care sunt deja existente, pentru a identifica și compara tehnicile utilizate în algoritmii de Generare Randomizată a Hărții, Pathfinding pentru IA.Scopul acestei investigații este de a sublinia diferențele și îmbunătățirile aduse de proiectul nostru în raport cu aceste sisteme, evidențiind avantajele și inovațiile pe care le poate aduce jocul nostru pe piața actuală de jocuri de supraviețuire,referire la tabelul[2.1].

**Tabelul 2.1 – Compararea jocurilor deja existente cu jocul creat**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Criteriu | Minecraft | Terraria | Subnautica | |  | | --- | | Lumea Sălbatică | |
| Realism | Grafică stilizată, fără intenția de realism. | Design stilizat și bidimensional. | Atmosferă realistă subacvatică, dar specifică. | Realism avansat al mediului, al creaturilor și al interacțiunilor. |
| Supraviețuire | |  | | --- | | Axat pe crafting și explorare, dar pericolele sunt simple. | | |  | | --- | | Pericole medii, accent pe luptă și explorare liniară. | | |  | | --- | | Provocări subacvatice, resurse limitate. | | |  | | --- | | Provocări complexe, inspirate din situații reale, inclusiv lupta pentru resurse în medii diverse. | |
| Inteligență Artificială | |  | | --- | | Monștri simpli, IA de bază. |  |  | | --- | |  | | |  | | --- | | IA limitată, mai degrabă statică. |  |  | | --- | |  | | |  | | --- | | Comportamente de bază ale creaturilor. |  |  | | --- | |  | | |  | | --- | | IA avansată, cu comportamente imprevizibile și realiste pentru creaturi și NPC-uri. |  |  | | --- | |  | |
| Provocări | |  | | --- | | Dificultate scalabilă, dar mai relaxată. |  |  | | --- | |  | | |  | | --- | | Provocări moderate, axate pe luptă. |  |  | | --- | |  | | |  | | --- | | Axat pe supraviețuirea subacvatică. |  |  | | --- | |  | | |  | | --- | | Provocări constante și diversificate, situații reale de supraviețuire. |  |  | | --- | |  | |

**2.1 Scopul și obiectivele propuse**  
 Într-o eră în care jocurile video devin din ce în ce mai populare ca formă de divertisment, crearea unui joc de supraviețuire captivant reprezintă o provocare interesantă și o oportunitate de a atrage o audiență largă. Scopul este de a crea un joc de interteiment care să capteze atenția jucătorului, care combină explorarea, gestionarea resurselor și interacțiunea socială, toate acestea într-un mediu virtual complex.

Obiectivele Propuse:

a) cercetarea pieței de jocuri de supraviețuire;

b) elaborarea conceptului de joc;

c) stabilirea resurselor necesare pentru dezvoltarea jocului;  
d) crearea unei lumi virtuale detaliate;

e) proiectarea unui sistem de progresie;

f) implementarea IA pentru NPC-uri și fauna sălbatică;

g) elaborarea funcționalului de crafting și supraviețuire;

h) testarea sistemului de interacțiune cu mediul;

i) optimizarea performanței și a graficii jocului;

Aceste scopuri și obiective vor contribui la crearea unei experiențe captivante și educative în „Lumea sălbatică”, încurajând jucătorii să se implice activ în lumea jocului și să dezvolte abilități esențiale de supraviețuire.  
 În celee din urma, dezvoltarea unui joc de supraviețuire este un proces complex care implică cercetare, planificare detaliată și o atenție deosebită la detalii. Prin urmarea acestor pași și obiective, se poate crea un joc captivant care să capteze atenția și imaginația jucătorilor, oferindu-le o experiență unică de divertisment și provocare. Cu o lume bine construită, mecanici interesante și o comunitate activă, jocul va putea să-și găsească locul în peisajul competitiv al industriei de gaming.

# **3 Realizarea jocului**

Sistemul dezvoltat pentru jocul de supraviețuire „Lumea Sălbatică” constă în integrarea unei aplicații complexe realizate cu ajutorul motorului Unity[4] și al limbajului de programare C#. Platforma de dezvoltare utilizează resurse optimizate pentru a crea o experiență captivantă, iar designul și funcționalitățile au fost atent implementate pentru a răspunde cerințelor utilizatorilor.

a) Tehnologiile utilizate:

1) Unity:  
 Unity[4] este platforma centrală pe care a fost construit jocul, oferind suport avansat pentru grafică 3D, fizică realistă și mecanici complexe. Motorul permite dezvoltarea rapidă și integrarea facilă a diferitelor componente, precum animațiile, interacțiunile dintre obiecte și efectele speciale.

2) C#:  
Limbajul de programare utilizat pentru dezvoltarea logicii jocului este C#, datorită performanței sale ridicate și a integrării native cu Unity[4]. Prin intermediul acestuia, au fost implementate funcționalități precum:

-Sistemul de supraviețuire (monitorizarea resurselor: hrană, apă, sănătate).

-Interacțiunea cu mediul (colectarea resurselor, construirea adăposturilor).

-Inteligența artificială a inamicilor și a animalelor sălbatice.

b)Structura jocului:

1) Grafică și animații  
Pentru a oferi o experiență vizuală imersivă, elementele grafice au fost realizate cu ajutorul Unity, utilizând asset-uri de înaltă calitate. Animațiile personajelor și efectele speciale, precum schimbările de vreme sau focul, au fost optimizate pentru o performanță fluidă.

2) Interfața utilizatorului (UI)[4]:  
 Un design intuitiv și responsiv al interfeței a fost creat pentru a ghida utilizatorul în gestionarea resurselor și navigarea în joc. Butoanele interactive, meniurile dinamice și indicatorii vizuali au fost dezvoltate cu sistemele UI[4].

c)Framework-uri și biblioteci utilizate:

Pentru eficiența dezvoltării, au fost integrate diverse tool-uri și librării suplimentare:

1) Post-Processing Stack – pentru efecte vizuale avansate, cum ar fi iluminarea  
globală, umbrele realiste și profunzimea câmpului vizual.

2) Pathfinding  – pentru crearea comportamentului inteligent al inamicilor.

d)Funcționalități dinamice:

Jocul „Lumea Sălbatică” aduce un nivel de interactivitate ridicat prin:

1) Schimbări de mediu în timp real (zi-noapte, vreme dinamică).

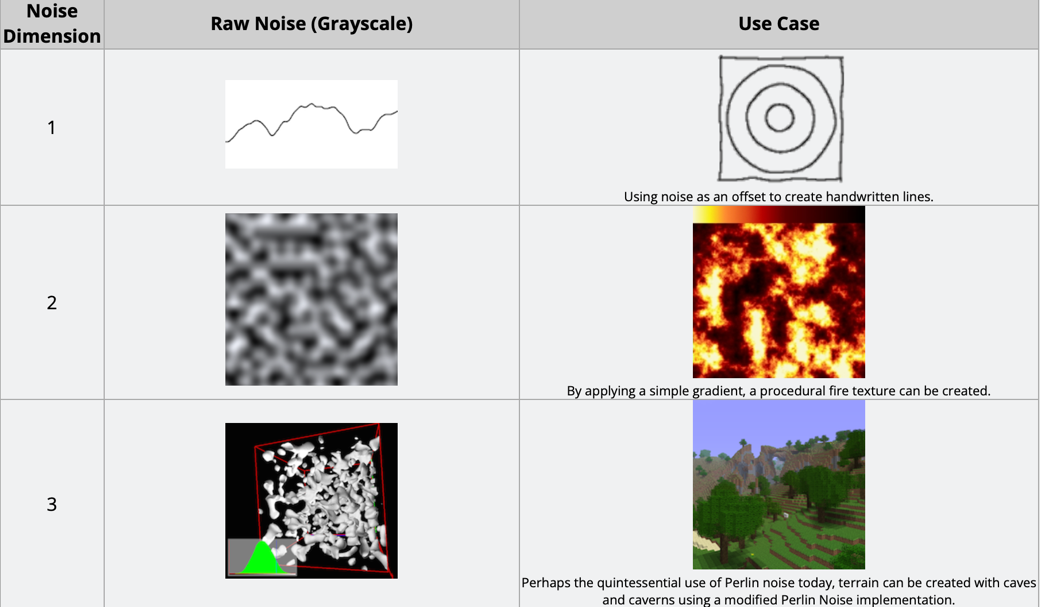
2) Reacții contextuale la acțiunile jucătorului (animații, sunete, efecte vizuale).

3) Personalizarea strategiilor de supraviețuire prin crafting și upgrade-uri.

Această combinație între Unity[4]  și C#[8] asigură o platformă robustă, capabilă să susțină mecanicile complexe și grafica imersivă a jocului „Lumea Sălbatică”, oferind o experiență memorabilă utilizatorilor.

**3.1 Implementarea algoritmului**

Implementarea algoritmului Perlin Noise în Lumea sălbatică, un joc 3D de supraviețuire, a fost realizată pentru a crea un mediu variat și captivant, cu terenuri naturale care să ofere o provocare autentică jucătorilor.   
De asemenea, dacă extindeți Perlin Noise într-o dimensiune suplimentară și luați în considerare dimensiunea suplimentară ca timp, îl puteți anima. De exemplu, zgomotul 2D Perlin poate fi interpretat ca Teren, dar zgomotul 3D poate fi interpretat în mod similar ca valuri ondulate într-o scenă oceanică. Mai jos sunt câteva imagini ale Zgomotului în diferite dimensiuni și câteva dintre utilizările lor în timpul execuției:

  
**Figura 3.1 – Perlin Noise imagini ale Zgomotului în diferite dimensiuni**

Scopul generării terenului in Lumea sălbatică, mediul de joc trebuia să includă elemente precum:

a)Munți și dealuri pentru a oferi obstacole naturale.

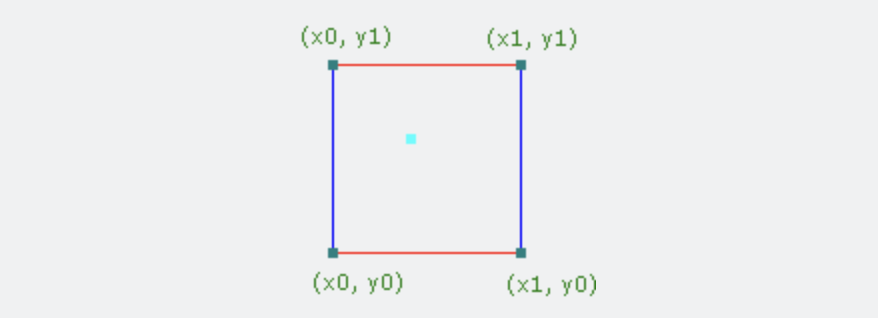
b)Văi și câmpii pentru zone de explorare și resurse.

c)Detalii naturale (precum denivelări subtile și stânci) pentru un aspect realist.

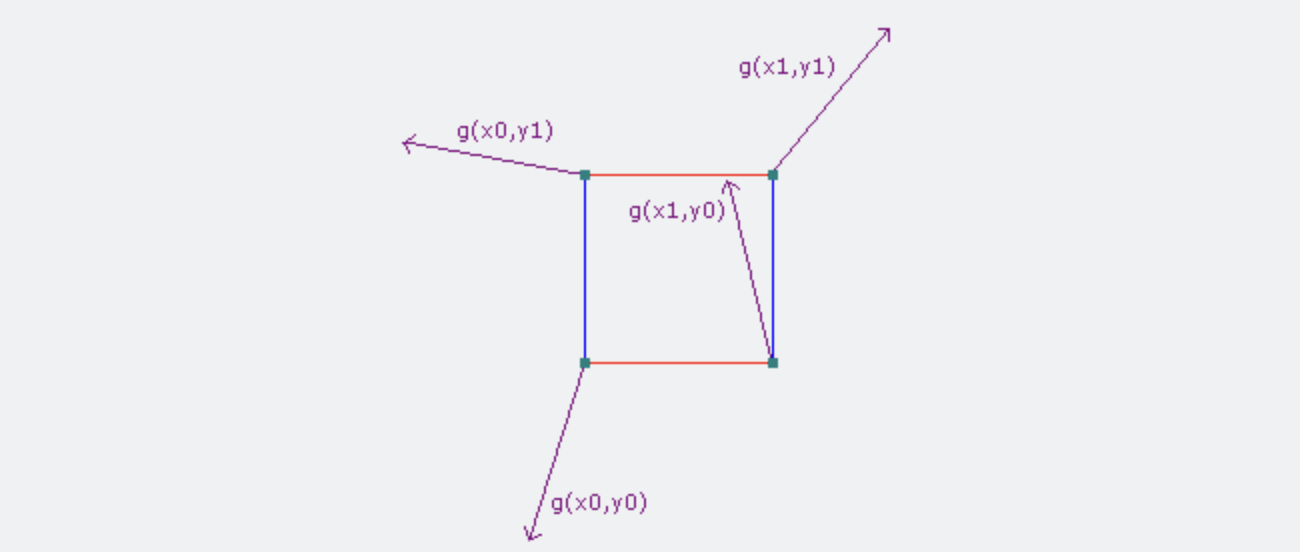
d)Continuitate între secțiunile terenului, fără artefacte sau tranziții abrupte.

Perlin Noise a fost alegerea ideală datorită capacității sale de a genera zgomot coerent, ceea ce face ca terenurile să pară fluide și naturale.  
 Să începem cu funcția de bază Perlin Noise:

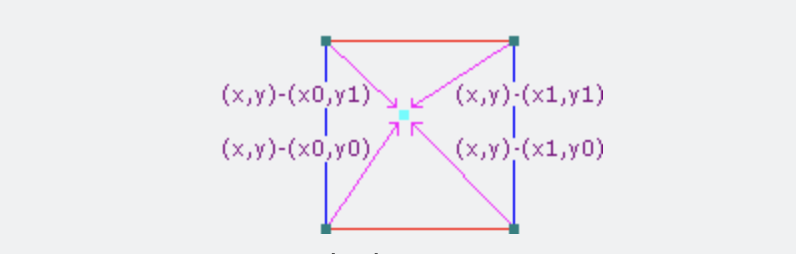
Deci avem o coordonată x, y și z ca intrare, iar ca ieșire obținem un dublu între 0,0 și 1,0. Mai întâi, împărțim coordonatele x, y și z în cuburi de unitate. Cu alte cuvinte, găsiți [x,y,z] % 1,0 pentru a găsi locația coordonatei în cub. Mai jos este o reprezentare a acestui concept în 2 dimensiuni:

**  
Figura 3.2 –** **Punctul albastru aici reprezintă o coordonată de intrare, iar celelalte 4 puncte sunt coordonatele unității integrale din jur.**

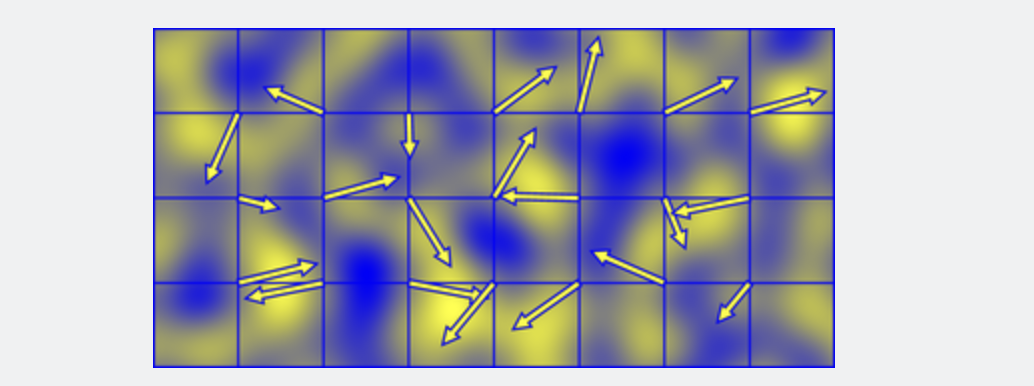
Pe fiecare dintre cele 4 unități de coordonate (8 în 3D), generăm ceea ce se numește un vector de gradient pseudoaleator. Acest vector de gradient definește o direcție pozitivă (în direcția către care indică) și, bineînțeles, o direcție negativă (în direcția opusă către care indică). Pseudoradom înseamnă că, pentru orice set de numere întregi introduse în ecuația vectorială gradient, același rezultat va ieși întotdeauna. Astfel, pare întâmplător, dar nu este în realitate. În plus, aceasta înseamnă că fiecare coordonată integrală are „propriul” gradient care nu se va schimba niciodată dacă funcția de gradient nu se schimbă.

**  
Figura 3.3 – Un vector de gradient pseudoaleator**

În continuare, trebuie să calculăm cei 4 vectori (8 în 3D) de la punctul dat la cele 8 puncte înconjurătoare de pe grilă. Un exemplu de acest caz în 2D este prezentat mai jos.

  
**Figura 3.3 – Cei 4 vectori (8 în 3D)**

Apoi, luăm produsul scalar dintre cei doi vectori (vectorul gradient și vectorul distanță). Aceasta ne oferă valorile noastre conform formulei (3.1) se calculează finale de influență:  
  
grad.x \* dist.x + grad.y \* dist.y + grad.z \* dist.z (3.1)

  
**Figura 3.4 – O reprezentare a acestor influențe în zgomotul 2D**

Deci, acum tot ce trebuie să facem este să interpolăm între aceste 4 valori, astfel încât să obținem un fel de medie ponderată între cele 4 puncte ale grilei (8 în 3D). Soluția la aceasta este ușoară: media mediile așa (acest exemplu este în 2D):

// Mai jos sunt 4 valori de influență în aranjament:// [g1] | [g2]

// -----------

// [g3] | [g4]

int g1, g2, g3, g4;

int u, v; // Aceste coordonate sunt locația coordonatei de intrare în pătratul său de unitate.

// De exemplu, o valoare de (0,5,0,5) se află în centrul exact al pătratului său.

int x1 = lerp(g1,g2,u);

int x2 = lerp(g3,h4,u);

int average = lerp(x1,x2,v);  
 Există o ultimă piesă a acestui puzzle: cu media ponderată de mai sus, rezultatul final ar arăta rău, deoarece interpolarea liniară, deși ieftină din punct de vedere computațional, pare nefirească. Avem nevoie de o tranziție mai lină între degrade. Deci, folosim o funcție de estompare, numită și curbă de ușurință:

**3.2 Interfața grafică a jocului** Jocul „Lumea Sălbatică” utilizează motorul grafic Unity[4] scris în C#, pentru a crea un mediu captivant de supraviețuire 3D. Interfața jocului prezintă o combinație de elemente vizuale și funcționalități interactive care gestionează personajele, obiectele și evenimentele din joc.  
 Se creează un proiect Unity[4] nou.Structura folderelor include :Assets , Scripts,  
Prefabs și Scenes.

Se configurează o scenă de bază cu elementele esențiale (teren, lumină, cameră). Se utilizează un teren 3D generat procedural cu Perlin Noise, ajustând textura pentru realism (iarbă, pământ).Terenul este personalizat în Unity[4] folosind componenta Terrain Editor pentru detalii suplimentare (înălțimi, materiale).Se adaugă obiecte statice: copaci și roci (prefabricate din folderul Prefabs).Obiectele sunt plasate manual în scena principală pentru varietate vizualăreferinta la Anexa A:  
  
using UnityEngine;

public class TerrainGenerator : MonoBehaviour

{

public int width = 256; // Lățimea terenului

public int depth = 20; // Adâncimea terenului

public int height = 256; // Lungimea terenului

public float scale = 20f; // Controlul detaliilor

void Start()

{

Terrain terrain = GetComponent<Terrain>();

terrain.terrainData = GenerateTerrain(terrain.terrainData);

}

TerrainData GenerateTerrain(TerrainData terrainData)

{

terrainData.heightmapResolution = width + 1;

terrainData.size = new Vector3(width, depth, height);

terrainData.SetHeights(0, 0, GenerateHeights());

return terrainData;

}

float[,] GenerateHeights()

{

float[,] heights = new float[width, height];

for (int x = 0; x < width; x++)

{

for (int z = 0; z < height; z++)

{

float xCoord = (float)x / width \* scale;

float zCoord = (float)z / height \* scale;

heights[x, z] = Mathf.PerlinNoise(xCoord, zCoord);

}

}

return heights;

}

}  
  
 Scrierea Scripturilor in C#unde mișcarea jucătorului:  
  
using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

public float speed = 5f;

void Update()

{

float moveHorizontal = Input.GetAxis("Horizontal");

float moveVertical = Input.GetAxis("Vertical");

Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

transform.Translate(movement \* speed \* Time.deltaTime, Space.World);

}

}  
  
 Testare se verifică interacțiunile dintre obiecte și animațiile animalelor.

Optimizări:  
a) Reducerea distanței de randare.

b) Optimizarea coliziunilor pentru a îmbunătăți performanța.  
 Acest proces oferă un cadru complet pentru construirea jocului, începând de la teren și obiecte de mediu până la logica entităților dinamice și controlul personajului.

**3.3 Analiza rezultatelor obținute** Proiectul și-a propus să dezvolte un joc de supraviețuire captivant care să combine explorarea, gestionarea resurselor și interacțiunea socială într-un mediu virtual complex. Obiectivele principale includ cercetarea pieței, crearea unei lumi virtuale detaliate, implementarea IA pentru fauna sălbatică și NPC-uri, și dezvoltarea unui sistem de crafting avansat.

Metodologia și implementarea tehnică:

a) Echipa a utilizat motorul grafic Unity[4] și limbajul C# pentru a dezvolta funcționalitățile principale ale jocului. Printre realizările tehnice notabile.

b) Generarea procedurală a hărților cu algoritmul Perlin Noise, care a permis crearea unor medii naturale variate, cu detalii subtile.

c) Sistemul de Pathfinding, ce oferă NPC-urilor capacitatea de a reacționa realist la mediul înconjurător și la acțiunile jucătorului.

d) Interfața grafică a utilizatorului (UI) intuitivă, care facilitează navigarea și gestionarea resurselor.

e) Optimizarea performanței jocului prin reducerea distanței de randare și gestionarea eficientă a coliziunilor.

Rezultate obținute

Calitatea grafică și interactivitatea:

f) Jocul oferă un mediu realist și captivant, datorită utilizării unor texturi detaliate și animații fluide.

g) Funcționalitățile dinamice, precum schimbările de vreme și ciclurile zi-noapte, adaugă un strat suplimentar de imersiune.

Algoritmi performanți:

h) Generarea aleatorie a hărților asigură rejucabilitate, iar optimizările implementate au menținut un echilibru între complexitate și resursele hardware necesare.

i) Sistemele IA demonstrează adaptabilitate și realism în comportamentul entităților dinamice.

Relevanța pentru utilizatori:

j) Jocul a fost conceput pentru a răspunde așteptărilor utilizatorilor care caută experiențe de supraviețuire autentice și provocatoare.

k) Comunitatea potențială din jurul jocului poate contribui la îmbunătățiri prin feedback și posibile modificări (moduri).

l) Provocări și soluții

Artefacte vizuale în generarea hărților: Algoritmul Perlin Noise a necesitat ajustări pentru a elimina structurile repetitive.

Optimizarea IA: Implementarea unor comportamente mai complexe a fost dificilă, dar abordarea modulară a facilitat testarea și îmbunătățirile ulterioare. **Concluzie** Proiectul „Lumea sălbatică” a reușit să îndeplinească obiectivele propuse, transformând oidee ambițioasă într-un produs funcțional și promițător. Jocul îmbină armonios tehnologiile moderne, cum ar fi generarea procedurală, inteligența artificială avansată și optimizările grafice, cu mecanici de joc captivante, oferind o experiență de supraviețuire unică.

Una dintre realizările notabile este crearea unui mediu virtual complex și dinamic, care reușește să simuleze fidel provocările supraviețuirii într-o lume sălbatică. Utilizarea algoritmului Perlin Noise pentru generarea hărților, combinată cu implementarea unui sistem de Pathfinding IA, a permis dezvoltarea unor scenarii imprevizibile și rejucabile, esențiale pentru atragerea jucătorilor pasionați de genul survival.   
 De asemenea, proiectul a demonstrat capacitatea echipei de a înfrunta provocările tehnice și conceptuale, precum eliminarea artefactelor vizuale, optimizarea performanței pe diverse dispozitive și dezvoltarea unui comportament adaptativ pentru NPC-uri. Soluțiile creative aplicate acestor dificultăți au contribuit la consolidarea unui produs robust și bine structurat.

Din punct de vedere al relevanței pentru public, „Lumea sălbatică” răspunde cererii tot mai mari pentru jocuri care oferă nu doar divertisment, ci și provocări autentice, realism și o comunitate activă. Jocul are potențialul de a atrage o gamă diversificată de utilizatori.

Proiectul nu doar că și-a atins scopul principal – acela de a oferi o experiență autentică și memorabilă – ci deschide și perspective promițătoare pentru viitor. Posibilitatea de a adăuga noi caracteristici, de a extinde lumea jocului și de a cultiva o comunitate activă în jurul său indică un potențial semnificativ de creștere. Cu îmbunătățiri continue și marketing eficient, „Lumea sălbatică” poate deveni un competitor serios pe piața globală a jocurilor de supraviețuire.

Pe viitor, „Lumea sălbatică” are potențialul de a evolua prin extinderea lumii virtuale, implementarea unor noi funcționalități și dezvoltarea unui ecosistem de conținut creat de utilizatori, cum ar fi modurile personalizate. În plus, construirea unei comunități active în jurul jocului poate spori valoarea produsului și poate asigura longevitatea acestuia.

Marketingul eficient, susținut de campanii de promovare axate pe canale de streaming, precum Twitch[1] și YouTube[2], ar putea ajuta la creșterea notorietății jocului, consolidându-i poziția pe piața globală a jocurilor de supraviețuire.

În concluzie, „Lumea sălbatică” reprezintă un exemplu convingător al modului în care o idee bine documentată, susținută de o echipă dedicată și de utilizarea creativă a tehnologiilor.  
 **Bibliografie**[1] Algoritmi pentru generarea procedurală a hărților, accesat 11.10.2024, [resursă electronică] - Regim de acces: <https://www.gamedeveloper.com/programming/procedural-map-generation-techniques>[2] - Studiul comportamentului jucătorilor în jocurile de supraviețuire, accesat 26.10.2024, [resursă electronică] - Regim de acces: <https://www.sciencedirect.com/science/article/abs/pii/S1071581920300519>  
[3] - Documentația C# pentru scriptarea în Unity, accesat 20.11.2024, [resursă electronică] - Regim de acces: <https://docs.unity3d.com/Manual/UsingTheCSharpCompiler.html>[4] Unity - Motor de joc, accesat 22.11.2024, [resursă electronică] - Regim de acces:  
<https://unity.com/>  
[5] Minecraft – Este un joc de supraviețuire și construcție,accesat 12.10.24, [resursă electronică] - Regim de acces: https://www.minecraft.net/en-us  
[6] Valheim - Un joc de supraviețuire open-world inspirat din mitologia nordică,accesat 12.10.24, [resursă electronică] - Regim de acces: https://www.valheimgame.com  
[7] Rust - E un joc de supraviețuire multijucător,accesat 12.10.24, [resursă electronică] - Regim de acces: https://store.steampowered.com/app/252490/Rust/

[8] Limbajul de programare C#, accesat 22.11.2024, [resursă electronică] - Regim de acces:  
<https://learn.microsoft.com/en-us/dotnet/csharp/>

[9] Algoritmul Perlin Noise, accesat 01.11.2024, [resursă electronică] - Regim de acces:  
<https://en.wikipedia.org/wiki/Perlin_noise>

[10] Simplex Noise Algorithm, accesat 01.11.2024, [resursă electronică] - Regim de acces:  
https://cs.nyu.edu/~perlin/noise/  
[11] Pathfinding Algorithms in Games, accesat 01.11.2024, [resursă electronică] - Regim de acces: <https://www.redblobgames.com/pathfinding/a-star/introduction.html>   
[12] Inteligența artificială în jocurile video, accesat 11.11.2024, [resursă electronică] - Regim de acces: <https://www.gamasutra.com/category/ai>  
[13] Algoritmul Greedy, accesat 01.11.2024, [resursă electronică] - Regim de acces: <https://en.wikipedia.org/wiki/Greedy_algorithm>  
 **Anexa A - Noise Map**using UnityEngine;

using System.Collections;

public static class Noise {

public enum NormalizeMode {Local, Global};

public static float[,] GenerateNoiseMap(int mapWidth, int mapHeight, int seed, float scale, int octaves, float persistance, float lacunarity, Vector2 offset, NormalizeMode normalizeMode) {

float[,] noiseMap = new float[mapWidth,mapHeight];

System.Random prng = new System.Random (seed);

Vector2[] octaveOffsets = new Vector2[octaves];

float maxPossibleHeight = 0;

float amplitude = 1;

float frequency = 1;

for (int i = 0; i < octaves; i++) {

float offsetX = prng.Next (-100000, 100000) + offset.x;

float offsetY = prng.Next (-100000, 100000) - offset.y;

octaveOffsets [i] = new Vector2 (offsetX, offsetY);

maxPossibleHeight += amplitude;

amplitude \*= persistance;

}

if (scale <= 0) {

scale = 0.0001f;

}

float maxLocalNoiseHeight = float.MinValue;

float minLocalNoiseHeight = float.MaxValue;

float halfWidth = mapWidth / 2f;

float halfHeight = mapHeight / 2f;

for (int y = 0; y < mapHeight; y++) {

for (int x = 0; x < mapWidth; x++) {

amplitude = 1;

frequency = 1;

float noiseHeight = 0;

for (int i = 0; i < octaves; i++) {

float sampleX = (x-halfWidth + octaveOffsets[i].x) / scale \* frequency;

float sampleY = (y-halfHeight + octaveOffsets[i].y) / scale \* frequency;

float perlinValue = Mathf.PerlinNoise (sampleX, sampleY) \* 2 - 1;

noiseHeight += perlinValue \* amplitude;

amplitude \*= persistance;

frequency \*= lacunarity;

}

if (noiseHeight > maxLocalNoiseHeight) {

maxLocalNoiseHeight = noiseHeight;

} else if (noiseHeight < minLocalNoiseHeight) {

minLocalNoiseHeight = noiseHeight;

}

noiseMap [x, y] = noiseHeight;

}

}

for (int y = 0; y < mapHeight; y++) {

for (int x = 0; x < mapWidth; x++) {

if (normalizeMode == NormalizeMode.Local) {

noiseMap [x, y] = Mathf.InverseLerp (minLocalNoiseHeight, maxLocalNoiseHeight, noiseMap [x, y]);

} else {

float normalizedHeight = (noiseMap [x, y] + 1) / (maxPossibleHeight/0.9f);

noiseMap [x, y] = Mathf.Clamp(normalizedHeight,0, int.MaxValue);

}

}

}

return noiseMap;

}

}  
  
 **Anexa B – Texture Generator**

using UnityEngine;

using System.Collections;

public static class TextureGenerator {

public static Texture2D TextureFromColourMap(Color[] colourMap, int width, int height) {

Texture2D texture = new Texture2D (width, height);

texture.filterMode = FilterMode.Point;

texture.wrapMode = TextureWrapMode.Clamp;

texture.SetPixels (colourMap);

texture.Apply ();

return texture;

}

public static Texture2D TextureFromHeightMap(float[,] heightMap) {

int width = heightMap.GetLength (0);

int height = heightMap.GetLength (1);

Color[] colourMap = new Color[width \* height];

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

colourMap [y \* width + x] = Color.Lerp (Color.black, Color.white, heightMap [x, y]);

}

}

return TextureFromColourMap (colourMap, width, height);

}

}

**Anexa C – IA Movement**using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class AI\_Movement : MonoBehaviour

{

Animator animator;

public float moveSpeed = 0.2f;

Vector3 stopPosition;

float walkTime;

public float walkCounter;

float waitTime;

public float waitCounter;

int WalkDirection;

public bool isWalking;

void Start()

{

animator = GetComponent<Animator>();

walkTime = Random.Range(3,6);

waitTime = Random.Range(5,7);

waitCounter = waitTime;

walkCounter = walkTime;

ChooseDirection();

}

void Update()

{

if (isWalking)

{

animator.SetBool("isRunning", true);

walkCounter -= Time.deltaTime;

switch (WalkDirection)

{

case 0:

transform.localRotation = Quaternion.Euler(0f, 0f, 0f);

transform.position += transform.forward \* moveSpeed \* Time.deltaTime;

break;

case 1:

transform.localRotation = Quaternion.Euler(0f, 90, 0f);

transform.position += transform.forward \* moveSpeed \* Time.deltaTime;

break;

case 2:

transform.localRotation = Quaternion.Euler(0f, -90, 0f);

transform.position += transform.forward \* moveSpeed \* Time.deltaTime;

break;

case 3:

transform.localRotation = Quaternion.Euler(0f, 180, 0f);

transform.position += transform.forward \* moveSpeed \* Time.deltaTime;

break;

}

if (walkCounter <= 0)

{

stopPosition = new Vector3(transform.position.x, transform.position.y, transform.position.z);

isWalking = false;

transform.position = stopPosition;

animator.SetBool("isRunning", false);

waitCounter = waitTime;

}

}

else

{

waitCounter -= Time.deltaTime;

if (waitCounter <= 0)

{

ChooseDirection();

}

}

}

public void ChooseDirection()

{

WalkDirection = Random.Range(0, 4);

isWalking = true;

walkCounter = walkTime;

}

}

**Anexa D – Player Movement**using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

public CharacterController controller;

public float speed = 12f;

public float gravity = -9.81f \* 2;

public float jumpHeight = 3f;

public Transform groundCheck;

public float groundDistance = 0.4f;

public LayerMask groundMask;

Vector3 velocity;

bool isGrounded;

// Update is called once per frame

void Update()

{

//checking if we hit the ground to reset our falling velocity, otherwise we will fall faster the next time

isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

if (isGrounded && velocity.y < 0)

{

velocity.y = -2f;

}

float x = Input.GetAxis("Horizontal");

float z = Input.GetAxis("Vertical");

//right is the red Axis, foward is the blue axis

Vector3 move = transform.right \* x + transform.forward \* z;

controller.Move(move \* speed \* Time.deltaTime);

//check if the player is on the ground so he can jump

if (Input.GetButtonDown("Jump") && isGrounded)

{

//the equation for jumping

velocity.y = Mathf.Sqrt(jumpHeight \* -2f \* gravity);

}

velocity.y += gravity \* Time.deltaTime;

controller.Move(velocity \* Time.deltaTime);

}

}