UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Joc 3D de aventură cu generare procedurală

Constantin-Alexandru Savu

Coordonator științific:
As. drd. ing. Andrei-Cristian Lambru

BUCUREŞTI

2023

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

3D Adventure game with procedural generation

Constantin-Alexandru Savu

Thesis advisor:
As. drd. ing. Andrei-Cristian Lambru

BUCHAREST

2023

CUPRINS

SINOPSIS

Această lucrare prezintă un joc video de aventură 3D generat procedural care prezintă patru scene distincte: câmpii, deșert, insule tropicale și peșteri. Jocul utilizează algoritmi avansați de generare procedurală pentru a crea o lume vastă și captivantă pe care jucătorii o pot explora și descoperi. Scopul final al jocului este de a învinge o varietate de inamici în timp ce explorează lumea generată procedural.

Pe lângă faptul că oferă o experiență de joc captivantă, această lucrare servește, de asemenea, ca o explorare academică a tehnicilor de generare procedurală și a aplicării lor la proiectarea jocurilor video. Algoritmii utilizați în joc demonstrează potențialul imens al generării procedurale pentru a crea lumi care sunt atât imprevizibile, cât și captivante pentru jucători.

ABSTRACT

This paper presents a procedurally generated 3D adventure video game featuring four distinct scenes: plains, desert, tropical islands and caves. The game uses procedural generation algorithms to create a vast and immersive world for players to explore and discover. The ultimate goal of the game is to defeat a variety of enemies while exploring the procedurally generated world.

In addition to providing an immersive gameplay experience, this work also serves as an academic exploration of procedural generation techniques and their application to video game design. The algorithms used in the game demonstrate the immense potential of procedural generation to create rich and immersive worlds that are both unpredictable and engaging for players.

MULȚUMIRI

Vreau să îi mulțumesc domnului As. drd. ing. Lambru Andrei-Cristian pentru toată îndrumarea și pentru tot suportul acordat în procesul de realizare a lucrării de diploma.

De asemenea, vreau să le mulțumesc părinților și prietenilor mei pentru tot sprijinul acordat în timpul elaborării acestei lucrări.

1 INTRODUCERE

1.1 Context

În această lucrare, este descris un joc video de aventură 3D generat procedural care prezintă capacitățile și provocările generației procedurale în proiectarea jocurilor video. Jocul prezintă patru scene diferite: câmpie, deșert, insule tropicale și peșteri, fiecare cu un teren și structuri distincte. Jocul folosește algoritmi de generare procedurală pentru a crea o lume vastă și captivantă, care este diferită de fiecare dată când jucătorul alege un nou seed pentru generarea unei scene. Scopul jucătorului este de a explora lumea și de a învinge diverși inamici.

Generarea procedurală este o tehnică care utilizează algoritmi matematici și seed-uri aleatorii pentru a genera conținut în mod automat. Generarea procedurală a fost utilizată pe scară largă în jocurile video pentru a crea lumi diverse și dinamice care oferă jucătorilor o experiență unică și variată. Cu toate acestea, generarea procedurală prezintă, de asemenea, numeroase provocări și limitări, cum ar fi asigurarea coerenței, calității și varietății conținutului generat.

Lucrarea oferă, de asemenea, o descriere și o analiză detaliată a algoritmilor de generare procedurală utilizați în joc, precum și a avantajelor și dezavantajelor acestora. Documentul discută, de asemenea, alegerile de proiectare și compromisurile implicate în crearea unui joc video cu generare procedurală, precum și feedback-ul și evaluarea din partea jucătorilor care au testat jocul. Lucrarea își propune să contribuie la cunoștințele academice și practice privind generarea procedurală în proiectarea de jocuri video, precum și să demonstreze potențialul și limitele acesteia.

1.2 Problema

Jocurile video sunt o formă populară și influentă de divertisment și artă care poate oferi jucătorilor experiențe captivante și captivante. Unul dintre aspectele cheie ale designului de jocuri video este crearea lumii jocului, care reprezintă mediul și cadrul în care se desfășoară jocul. Lumea jocului poate avea un impact semnificativ asupra gameplay-ului, esteticii și narațiunii jocului, precum și asupra satisfacției jucătorului.

Cu toate acestea, crearea unei lumi de joc este, de asemenea, o sarcină complexă și dificilă care necesită mult timp, resurse și creativitate. În mod tradițional, lumile jocurilor sunt proiectate manual de către designeri și artiști, care trebuie să creeze fiecare detaliu și element al lumii, cum ar fi terenul, vegetația, clădirile, obiectele, personajele și evenimentele. Această abordare are unele avantaje, cum ar fi faptul că permite un control precis și o personalizare a lumii jocului, precum și asigurarea coerenței și calității conținutului. Cu toate acestea, această

abordare are și unele dezavantaje, cum ar fi faptul că necesită mult timp și forță de muncă, precum și că limitează dimensiunea și diversitatea lumii jocului.

Pentru a depăși aceste limitări, multe jocuri video folosesc generarea procedurală pentru a crea lumea de joc. Generarea procedurală poate oferi multe beneficii pentru proiectarea jocurilor video, cum ar fi reducerea timpului și a costurilor de dezvoltare, creșterea cantității și a varietății conținutului, îmbunătățirea posibilităților de reluare și explorare, precum și crearea de lumi infinite și dinamice.

Cu toate acestea, generarea procedurală prezintă, de asemenea, multe provocări și limitări pentru proiectarea jocurilor video. Unele dintre aceste provocări includ asigurarea coerenței, calității și varietății conținutului generat, echilibrarea caracterului aleatoriu și a controlului asupra procesului de generare, integrarea conținutului procedural cu elementele de joc, evaluarea și testarea conținutului procedural.

1.3 Obiective

Obiectivul principal al acestei lucrări este de a prezenta un joc video de aventură 3D generat procedural, care pune în evidență capacitățile și provocările generației procedurale în proiectarea jocurilor video.

În acest scop vor fi descrise și analizate sistemele de generare procedurală utilizate în joc, cum ar fi sistemul de generare a terenului sau sistemul de gestionare a modelelor 3D.

Un obiectiv major în implementarea acestor sisteme este de crea sisteme de generare procedurală a lumii ușor de folosit, ușor de extins și care pot genera lumi diverse și distincte.

Un alt obiectiv al lucrării este de a pune în evidență sistemele auxiliare sistemului de generare procedurală. Aceste sisteme includ: sistemele de gestionare a jocului, sistemele de inamici și au rolul de a crea o experiență completă de joc într-o lume generată procedural.

1.4 Structura lucrării

Lucrarea prezintă următoarele capitole:

Studiu de piață

În acest capitol vor fi analizate un număr de jocuri video care se folosesc de generare procedurală pentru a generarea de conținut. Aceste jocuri sunt emblematice pentru genurile de jocuri care se folosesc de generare procedurală.

Descrierea aplicației

Acest capitol prezintă arhitectura aplicației. Aceasta cuprinde diversele sisteme folosite pentru generarea procedurală a lumii, precum și alte sisteme necesare unui joc video.

Detalii de implementare

În acest capitol vor fi descrise detaliile de implementare ale aplicației. Acestea vor cuprinde, în mare parte, detalii legate de implementarea sistemelor de generare procedurală.

Evaluarea rezultatelor

Acest capitol va fi dedicat analizei sistemelor de generare procedurală descrise în capitolele anterioare. Sistemele vor fi analizate din punct de vedere impactului setărilor de generare asupra timpului de generare. De asemenea, sistemele vor fi analizate din perspectiva jucătorilor.

Concluzii

În acest capitol vor fi prezentate realizările și neajunsurile jocului așa cum au fost descrise în capitolele anterioare, precum și calea dorită pentru viitor.

2 STUDIU DE PIAȚĂ

În prezent, există o multitudine de jocuri video care utilizează tehnici de generare procedurală pentru a crea diverse elemente ale mediului de joc. Printre cele mai importante exemple se numără:

- Minecraft
- No Man's Sky
- Valheim

2.1 Minecraft

Minecraft [1] este un joc 3D de tip sandbox care permite jucătorilor să creeze și să exploreze lumi infinite făcute din blocuri. Jocul are două moduri principale: Survival și Creative [2]. În modul Survival, jucătorii trebuie să adune resurse, să confecționeze obiecte, să lupte cu inamicii și să își gestioneze diferite aspecte ce țin de supraviețuire. În modul Creativ, jucătorii au resurse nelimitate și pot construi orice își pot imagina, fără restricții sau pericole. Minecraft are și alte moduri, cum ar fi Adventure, Hardcore, Spectator și Multiplayer, unde jucătorii se pot alătura serverelor online și pot juca împreună sau unul împotriva celuilalt.

Minecraft a fost creat de dezvoltatorul suedez de jocuri Markus "Notch" Persson în 2009 și este întreținut de Mojang Studios, o parte a Xbox Game Studios, care la rândul său face parte din Microsoft. Jocul este disponibil pe diverse platforme, precum Windows, Mac, Linux, Android, iOS, Xbox, PlayStation, Nintendo Switch și altele. Jocul s-a vândut în peste 238 de milioane de exemplare la nivel mondial începând cu aprilie 2021 și are 141 de milioane de jucători activi lunar începând cu aprilie 2021 [3].

Minecraft este cunoscut pentru generarea sa procedurală, ceea ce înseamnă că lumea jocului este generată aleatoriu pe baza unui seed care poate fi ales de jucător sau generat aleatoriu. Acest lucru permite o varietate și o diversitate nesfârșită în lumea jocului, care poate include diferite biome-uri, structuri, peșteri, minereuri și vegetație. De asemenea, jucătorii pot să modifice lumea jocului cu diverse instrumente și obiecte care pot fi confecționate sau găsite în joc, precum târnăcoape sau explozibile.

2.2 No Man's Sky

No Man's Sky este un joc de supraviețuire science-fiction care le permite jucătorilor să exploreze o galaxie infinită generată procedural, în care fiecare stea, planetă, lună, creatură și plantă este unică. Jocul are patru activități principale: explorare, supraviețuire, luptă și

comerțul. Jucătorii pot zbura între planete și sisteme stelare folosind propria navă stelară sau pot folosi portaluri pentru a se teleporta prin galaxie și pot călători către alte galaxii.

No Man's Sky a fost dezvoltat și publicat de Hello Games, un studio independent cu sediul în Marea Britanie. Jocul a fost lansat pentru PlayStation 4 și Windows în august 2016, pentru Xbox One în iulie 2018, pentru PlayStation 5 și Xbox Series X/S în noiembrie 2020, pentru Nintendo Switch în octombrie 2022 și pentru macOS și iPadOS în iunie 2023. Jocul a primit mai multe actualizări majore de la lansare, adăugând noi caracteristici, cum ar fi construirea de baze, multiplayer, vehicule, misiuni, biome și creaturi [4].

No Man's Sky este unul dintre cele mai ambițioase și inovatoare jocuri care folosesc generarea procedurală. Această tehnică îi permite să creeze un număr mare de planete, până la 18 quintilioane [5]. Fiecare planetă are propriile sale caracteristici distinctive, inclusiv plantele și animalele care trăiesc pe ea. Acestea sunt, de asemenea, create prin generare procedurală, ceea ce le face să diferite de la o planetă altă planetă.

2.3 Valheim

Valheim [6] este un joc tip survival-sandbox care se desfășoară într-o lume generată procedural, inspirată de mitologia nordică și de cultura vikingă. Jocul a fost dezvoltat de Iron Gate Studio și publicat de Coffee Stain Studios. A fost lansat în early-access pentru Windows și Linux pe 2 februarie 2021, iar pentru Xbox One și Xbox Series X/S pe 14 martie 2023.

În Valheim, jucătorii își asumă rolul unor vikingi care au fost trimiși de Odin pe intr-un purgatoriu numit Valheim. Acolo, ei trebuie să exploreze, să construiască și să lupte pentru a supraviețui și a se dovedi demni de a intra în Valhalla. Lumea jocului este împărțită în diferite biome-uri, fiecare cu propriul teren, vreme, resurse, inamici și inamici de tip Boss. Lumea jocului este creată cu ajutorul generării procedurale, ceea ce înseamnă că este generată aleatoriu pe baza unor reguli matematice și a unor seed-uri, asigurându-se că fiecare jucător va avea parte de o experiență unică și variată.

3 DESCRIEREA APLICAȚIEI

3.1 Sistemului de gestionare a blocurilor

Cum o lume poate fi alcătuită din milioane de blocuri, instanțierea individuală a fiecărui bloc drept un model 3D nu este fezabilă, astfel, trebuie implementat un sistem care poate genera modele 3D alcătuite din mai multe blocuri de bază. În acest scop a fost implementat un sistem de gestionare a blocurilor care pot fi folosite pentru a genera mai departe un model 3D care agregă datele pentru mai multe blocuri.

Blocurile sunt de mai multe tipuri (ex. bloc de iarbă, bloc de pământ, bloc de apă, etc.).

Fiecare tip de bloc are trei texturi care definesc cele șase fețe ale unui bloc. Prima textură este folosită pentru fața de sus, a doua pentru fața de jos, iar a treia pentru fețele laterale. Texturile pentru toate blocurile se află într-un atlas de texturi.

Acestea au proprietăți diferite care afectează:

- aspectul modelului 3D din care fac parte (de ex. dacă este transparent, dacă este solid)
- comportamentul lor în spațiul de joc față de jucător (de ex. dacă blocul poate fi distrus de către jucător sau nu, dacă există coliziune între jucător și acel bloc)
- comportamentul lor în spațiul de joc față de inamici (dacă inamicii se pot mișca pe suprafața unui bloc)

Toate datele descrise mai sus sunt stocate într-o structura auxiliară care poate fi ușor modificată în editorul Unity, iar la runtime sunt convertite într-un dicționar pentru a avea un acces eficient la datele pentru fiecare tip de bloc.

Diagrama de clasă a sistemului este prezentată în Figura 1, aflată mai jos.

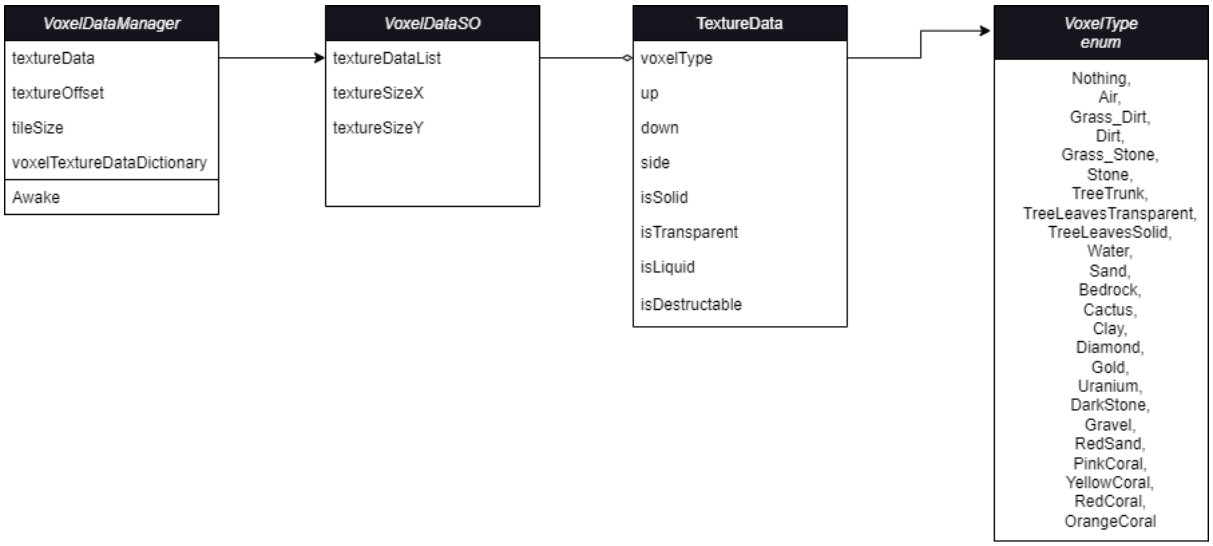


Figura 1 – Diagrama de clasă a sistemului de blocuri

3.2 Sistemul de gestionare a chunk-urilor

Pentru a putea agrega blocurile descrise în capitolul de mai sus este folosit un sistem de chunk-uri. Un chunk reprezintă mai multe blocuri care împreună definesc un paralelipiped dreptunghic. Mai multe chunk-uri alcătuiesc lumea pe care jucătorul o poate explora.

Din punct de vedere logic, sistemul de chunk-uri este separat în trei părți componente:

- Sistemul de gestionare a datelor
- Sistemul de generare a datelor modelului 3D
- Sistemul de desenare a modelului 3D

3.2.1 Sistemul de gestionare a datelor

Acest sistem are rolul de a stoca datele dintr-un chunk și oferă posibilitatea de a manipula datele dint-un chunk în funcție de nevoile altor sisteme. Un chunk are o dimensiune dată pentru toate cele trei axe de coordonate.

Sistemul se folosește de două tipuri de coordonate pentru a gestiona blocurile dintr-un chunk:

- Coordonate locale, folosite într-un chunk. Aceste coordonate depind de dimensiunea unui chunk
- Coordonate globale, folosite de alte sisteme pentru a accesa un bloc dintr-un chunk

Sistemul este alcătuit din doua clase, descrise și în figura 2:

- ChunkData, care are rolul de stocare a datelor relevante pentru un chunk
- Chunk, care are rolul de manipulare a datelor dintr-o clasă ChunkData, această clasă include metode prin care:
    - se poate seta un bloc la o anumită poziție, globală sau locală
    - se pot genera datele 3D ale unui chunk folosind sistemul de generare a datelor 3D
    - se pot obține chunk-urile vecine în spațiul global cu scopul de a efectua anumite operații

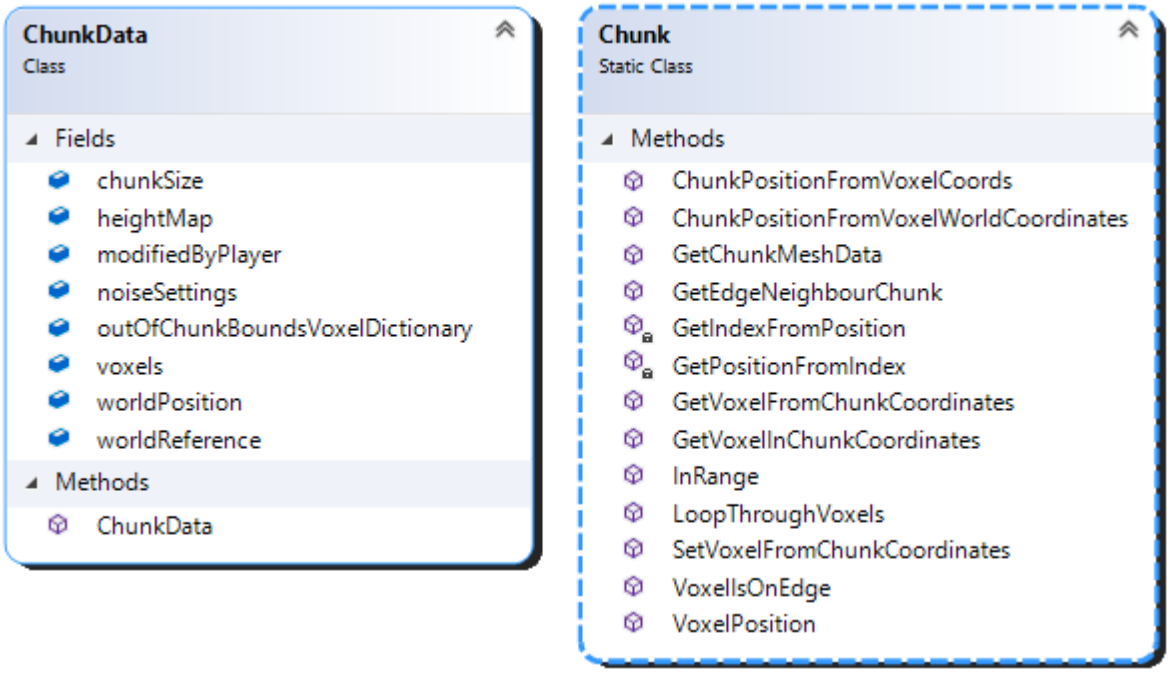


Figura 2 - Diagramele de clasă ale sistemului de gestionare a datelor

3.2.2 Sistemul de generare a datelor modelului 3D

Sistemul de generare a datelor modelului 3D al unui chunk are rolul de a genera, pentru fiecare chunk datele folosite pentru a desena modelul 3D al unui chunk.

Datele generate pentru un model 3D al unui chunk sunt:

- Vârfurile care definesc blocurile dintr-un chunk
- Triunghiurile care definesc blocurile dintr-un chunk
- Coordonatele de textură care definesc blocurile dintr-un chunk

Fiecare model al unui chunk este împărțit în patru submodele, ce permit un nivel mai ridicat de control pentru comportamentul fiecărui chunk. De exemplu, pot le pot fi atribuite straturi de coluziune diferite astfel încât jucătorul poate detecta o coliziune cu un tip de submodel dintr-un chunk, dar nu cu alt submodel din același chunk.

Cele patru submodele sunt:

- Modelul solid, care este alcătuit din datele generate de blocurile de tip solid, aflate la suprafață
- Modelul lichid, care este alcătuit din datele generate de blocurile de tip lichid.
- Modelul subacvatic, care este alcătuit din datele generate de blocurile de tip solid, aflate la sub apă

- Modelul transparent, care este alcătuit din datele generate de blocurile de tip transparent.

Sistemul de generare a datelor modelului 3D este alcătuit din două clase, descrise și în figura 3:

- Clasa MeshData este responsabilă cu stocarea datelor
- Clasa VoxelHelper este responsabilă cu generarea datelor

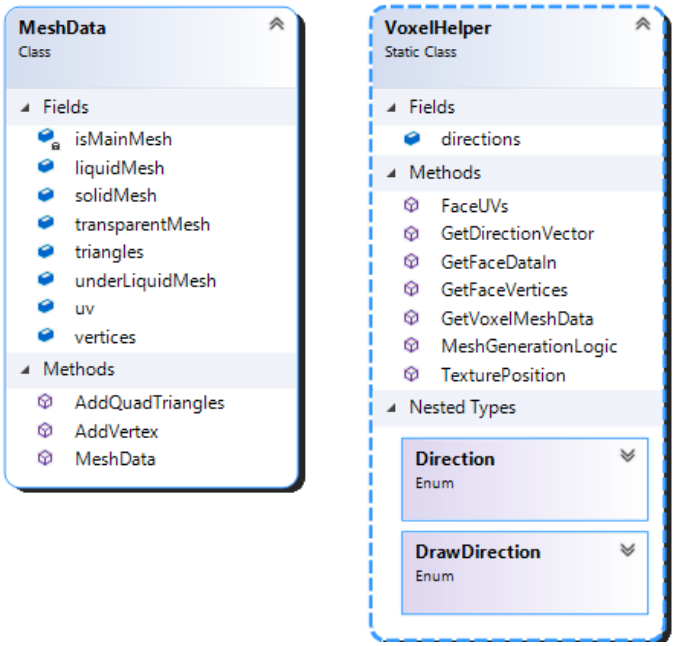


Figura 3 - Diagramele de clasă ale sistemului de generare a datelor modelului 3D

3.2.3 Sistemul de desenare a modelului 3D

Sistemul de desenare a modelului 3D al unui chunk are rolul de a folosi datele generate de sistemul de generare a datelor pentru modelul 3D împreună cu sistemele Unity de desenare a obiectelor din spațiul de joc pentru a desena pe ecran modelul 3D al unui chunk. Acest sistem are rolul de a desena pe ecran chunk-urile atunci când sunt generate, sau de a le actualiza modelul 3D atunci când jucătorul modifică terenul.

După cum a mai fost menționat mai sus, modelul 3D al unui chunk este alcătuit din patru sub-modele, după cum se poate vedea și în figura 4. Aceste sub-modele au proprietăți diferite, care ajută în diversificarea mediului de joc atât d.p.d.v. grafic, cât și tehnic după cum va fi descris în capitolele următoare. Fiecare sub-model are în compoziția acestuia:

- O componentă care permite desenarea pe ecran a modelului
- O componentă care permite atribuirea unui material modelului

- O componentă care permite configurarea unei meșe de coliziune

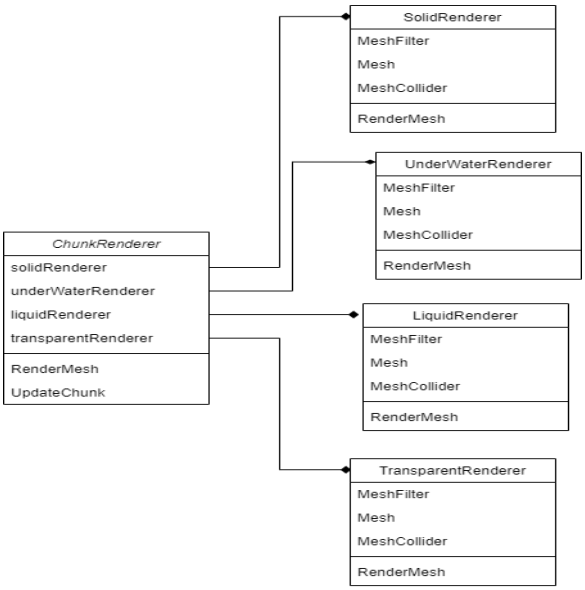


Figura 4 - Diagramele de clasă ale sistemului de desenare a modelului 3D

3.3 Sistemul de gestionare a jocului

Pentru a gestiona diferitele aspecte ale jocului, a fost implementat un sistem de gestionare a jocului. Acest sistem gestionează următoarele aspecte:

- Spawnarea jucătorului și generarea de chunk-uri suplimentare
- Generare a bordurilor din spațiul de joc
- Tranziție între starea normală de joc și starea de pauză a jocului
- Generarea lumii
- Generarea meșei de navigare a inamicilor
- Spawnarea inamicilor

Acest sistem se folosește de mai multe clase pentru a implementa diferitele aspecte menționate mai sus:

- Clasa GameManager, inițializează celelalte clase gestionare
- Clasa NavMeshManager gestionează meșa de navigație a inamicilor
- Clasa EnemySpawner spawnează inamicii pe hartă
- Clasa PlayerManager spawnează jucătorului și generează chunk-uri suplimentare
- Clasa BorderManager gestionează bordurile de la marginile spațiului de joc
- Clasa PauseManager gestionează tranzițiile între starea normală de joc și starea de pauză a jocului.

În figura 5 sunt descrise clasele care alcătuiesc acest sistem.

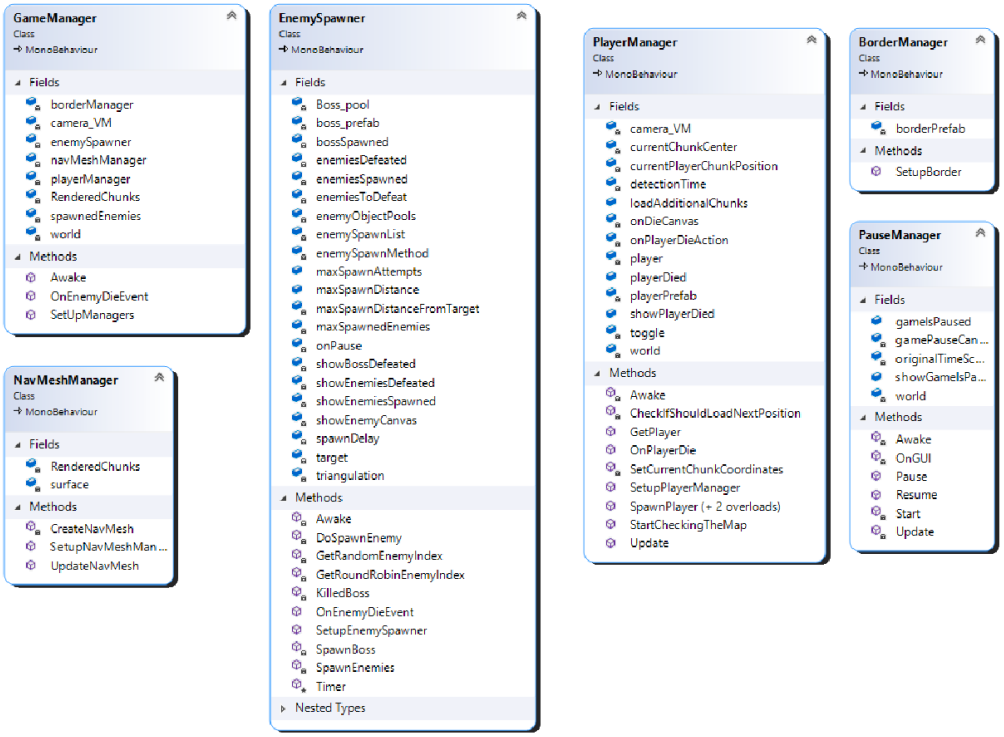


Figura 5 - Diagramele de clasă ale sistemului de gestionare a jocului

3.4 Sistemul de generare procedurală a datelor

Sistemul de generare procedurală are rolul de a genera date de tip bloc pentru toate pozițiile dintr-un chunk dat. Sistemul este responsabil atât pentru generarea de teren, cât și pentru generarea de structuri. Sistemul trebuie să genereze diferite tipuri de teren și de structuri în funcție de configurațiile setate pentru fiecare scenă a jocului.

Pentru generarea procedurală a terenului este necesar să existe un sistem de generare și de prelucrare de zgomot, un sistem de generare de teren și un sistem de generare de structuri.

În acest scop au fost implementate trei sub-sisteme care gestionează o parte diferită a procesului de generare procedurală:

1. Sistemul de generare și prelucrare de zgomot
2. Sistemul de generare de teren
3. Sistemul de generare de structuri

3.4.1 Sistemul de generare și prelucrare de zgomot

Sistemul se folosește de zgomot Perlin sau de zgomot Simplex pentru a genera valorile de bază ale zgomotului. Valorile generate se află în intervalul [1, 1]. Aceste valori sunt apoi prelucrate pentru a genera zgomot de tip roz (cunoscut și drept zgomot fractal). Valorile

generate se află în intervalul [1, 1]. Zgomotul de tip roz este generat folosind o serie de setări care controlează modelul de zgomot generat. Aceste setări sunt:

- Nivelul de zoom al zgomotului
- Deplasamentul local și global
- Persistența
- Lacunaritate
- Numărul de octave

De asemenea, pentru a avea un control mai fin asupra valorilor de zgomot, acestea pot fi redistribuite prin înmulțiri și ridicări la putere.

Sistemul de generare și prelucrare de zgomot este folosit și pentru a genera pozițiile structurilor. Aceste poziții sunt interpretate drept maximele locale dintr-o hartă de zgomot generată.

Pentru a implementa comportamentul descris mai sus sunt folosite clasele din figura 6.

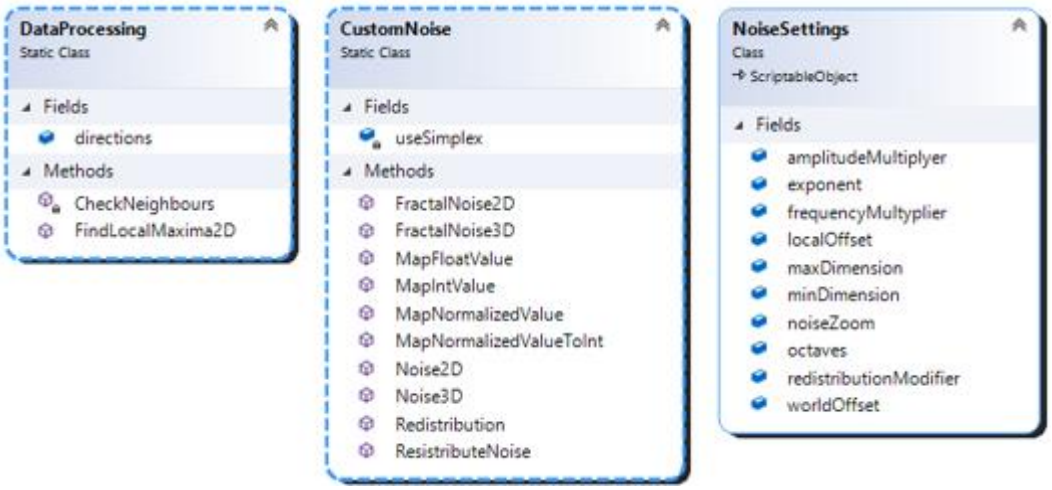


Figura 6 - Diagramele de clasă ale sistemului de generare și prelucrare de zgomot

3.4.2 Sistemul de generare de teren

Pentru a genera terenul unui chunk zgomotul generat de către sistemul de către sistemul de generare și prelucrare este folosit în doua moduri:

1. Pentru a genera harta de înălțime a lumii folosind zgomot generat folosind coordonate 2D
2. Pentru a genera filoane (în engleză vein) tridimensionale de blocuri folosind coordonate 3D. În funcție de blocul ales pentru aceste filoane, terenul generat poate fi considerat fie ca o peșteră (dacă blocul ales este de tipul Aer), fie pentru a genera un filon propriu-zis de blocuri.

Pentru a controla modul în care terenul se generează este folosită o ierarhie de straturi logice aplicate succesiv care se folosesc de harta de înălțime a unui chunk. Aceste straturi au rolul de a amplasa un bloc pentru o poziție dată dacă anumite condiții sunt îndeplinite, iar în caz contrar să cedeze responsabilitatea de amplasare a unui bloc următorului strat din ierarhie. Acest comportament descrie design pattern-ul Chain Of Responsability [7].

În figura 7 sunt descrise clasele folosite pentru ierarhia de straturi.

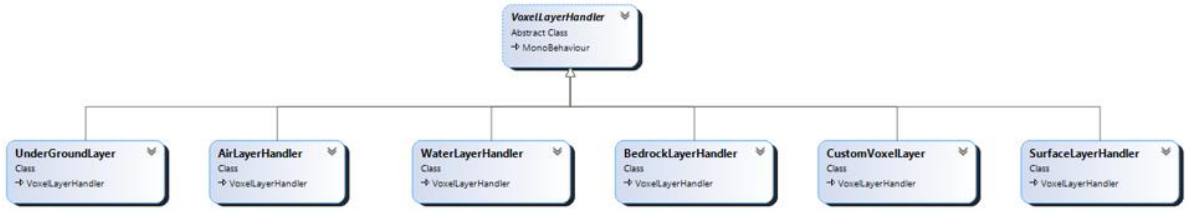


Figura 7 - Diagramele de clasă ale sistemului de generare de teren

3.4.3 Sistemul de generare de structuri

Sistemul de generare de structuri are rolul de a amplasa structuri într-un chunk.

Pozițiile unde sunt amplasate structurile sunt determinate de pozițiile maximelor locale într-o hartă de zgomot. Aceste poziții sunt calculate folosind sistemul de generare și prelucrare a zgomotului. Densitatea maximelor locale depinde de setările folosite pentru generarea hârții de zgomot. Astfel, putem genera structuri care se pot află foarte aproape unele de altele, care se află departe unele de altele, sau care au densități foarte mari, dar în zone restrânse.

Pentru a controla modul în care structurile sunt generate este folosită o ierarhie de straturi logice aplicate succesiv. Aceste straturi au rolul de a amplasa blocurile componente ale unei structuri începând de la o poziție dată dacă anumite condiții sunt îndeplinite, iar în caz contrar să cedeze responsabilitatea de amplasare a structurii următorului strat din ierarhie. Acest comportament descrie design pattern-ul Chain Of Responsability [7].

Structurile sunt alcătuite dintr-o mulțime de poziții care au asociate un bloc. Aceste structuri pot fi rotite de către stratul logic care amplasează structura. De asemenea, stratul logic care amplasează structura poate adăuga blocuri suplimentare structurii.

În figura 8 sunt descrise clasele folosite pentru ierarhia de straturi.

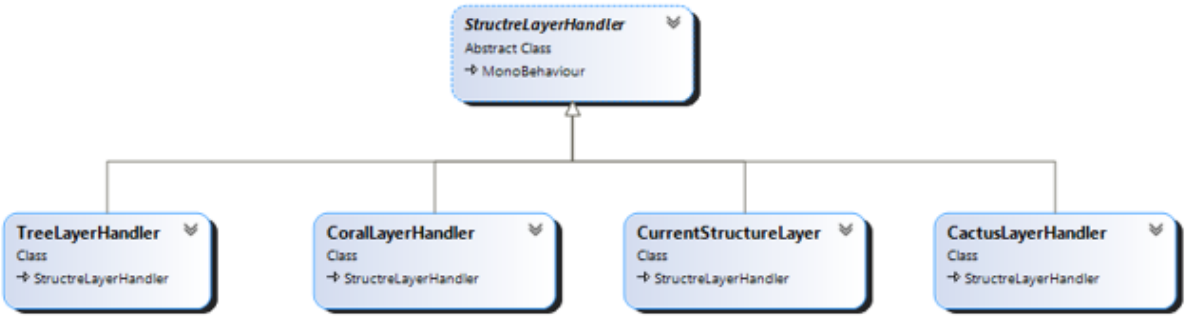


Figura 8 - Diagramele de clasă ale sistemului de generare de structuri

3.5 Sistemul de gestionare a lumii

Pentru a gestiona chunk-urile din care este alcătuită lumea generată în jurul jucătorului este nevoie de un sistem de gestionare a lumii. Acest sistem are următoarele roluri:

- de a determina pozițiile chunk-urilor aflate în jurul jucătorului
- de a determina pozițiile unde trebuie să se genereze chunk-uri în jurul jucătorului
- de a începe procesul de generare a chunk-urilor, acest proces incluzând: generarea datelor de tip bloc ale unui chunk și generarea datele pentru modelului 3D
- de a începe procesul de desenare a chunk-urilor
- de a începe procesul de modificare a chunk-urilor atunci când acestea sunt modificate de jucător

Sistemul se folosește de mai multe setări care determină cum va fi generată lumea. Aceste setări sunt:

- poziția de pornire a jucătorului
- dimensiunea unui chunk
- dimensiunea unui bloc
- dimensiunile hărții în chunk-uri
- decalajul global folosit pentru generarea de zgomot neted
- pragul de apă
- dimensiunile hărții în blocuri.

Sistemul de gestionare a lumii generează lumea din jurul jucătorului în următorii pași și sunt descriși de asemenea de diagrama de workflow din figura 9:

1. Se determină pozițiile de generare a chunk-urilor din jurul jucătorului. Acestea apoi sunt filtrate pentru a exclude chunk-urile deja create pentru pozițiile respective.

2. Se generează datele pentru blocurile chunk-urilor de la pozițiile date folosind sistemul de generare de teren
3. Se generează datele pentru modelele 3D ale chunk-urilor folosind datele generate anterior folosind sistemul de generare de date 3D
4. Se desenează datele generate anterior folosind sistemul de desenare al chunk-urilor

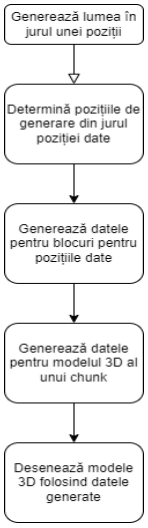


Figura 9 – Diagrama de flux a sistemului de gestionare a lumii

Acești pași se repetă de fiecare dată când jucătorul părăsește limitele unui chunk, dacă setarea de generare adițională de chunk-uri a sistemului de gestionare a jucătorului este activată.

3.6 Meniurile

Pentru a naviga între scene și pentru a avea posibilitatea de a controla unele aspecte ale generării procedurale de către utilizator, au fost folosite o serie de meniuri tipice pentru un joc video.

Aceste meniuri includ:

- Un meniu principal
- Un meniu de selecție a scenei
- Un meniu README ce are scopul de a familiariza jucătorul cu scopul jocului și cu modul de control
- Un meniu de generare a lumii în care jucătorul poate introduce un string ce reprezintă seed-ul cu care este generată lumea. În cazul în care nu este introdus un seed, acesta este generat aleator.

- Un meniu de pauză

3.7 Scenele

Jocul este împărțit în cinci scene:

1. Scena pentru meniul principal
2. Scena care generează teren de tip Câmpie (Plains)
3. Scena care generează teren de tip Deșert (Desert)
4. Scena care generează teren de tip Insule Tropicale (Islands)
5. Scena care generează teren de tip Peșteri (Caves)

Fiecare scenă care generează teren are următoarele sisteme:

- Sistemul de gestionare a blocurilor
- Sistemul de gestionare a chunk-urilor
- Sistemul de gestionare a lumii
- Sistemul de gestionare a jocului
- Sistemul de generare procedurală
- Sistemul de meniuri

Este de la sine înțeles că setările și configurațiile sistemelor componente fiecărei scene diferă de la scenă la scenă pentru a genera diferite tipuri de teren. De exemplu, Pentru scena de peșteri este folosit un interval mai mare de desenare a chunk-urilor pe axa y, deoarece pentru scena respectivă este dezirabil un spațiu de explorare mai mare pe verticală.

Fiecare scenă are opțiunea de a genera chunk-uri suplimentare oprită. Acest lucru se datorează faptului că generarea de chunk-uri noi duce la probleme de performanță semnificative în timpul jocului.

3.8 Personajul jucabil

Personajul jucabil este caracterul din joc controlat de către jucător. Cu ajutorul acestui caracter, jucătorul poate explora lumea și poate învinge inamicii spawnați.

Acest personaj are un model importat din pachetul Animated Characters 2 [8], distribuit gratuit pe platforma Kenney Asset Store. Acest personaj este animat folosind animațiile din pachetul menționat anterior. Presonajul ține în mâna o sabie al cărui model este importat din pachetul Melee Warrior Animations FREE [9], aflat pe platforma Unity Asset Store.

Personajul se folosește de clasele:

- Character, pentru a determina comportamentul jucătorului și pentru a anima modelul 3D. Aceste comportamente includ aspecte precum: viteza de atac a jucătorului, atacul inamicilor, comportamenul jucătorului atunci când se alfă pe uscat, comportamenul jucătorului atunci când se alfă în apă
- PlayerHealthSystem, pentru a gestiona nivelul de viață al jucătorului
- PlayerMovement, pentru a controla mișcarea caracterului în spațiul 3D
- PlayerCamera, pentru a controla camera jucătorului

Clasele de mai sus sunt descrise în figura 10.

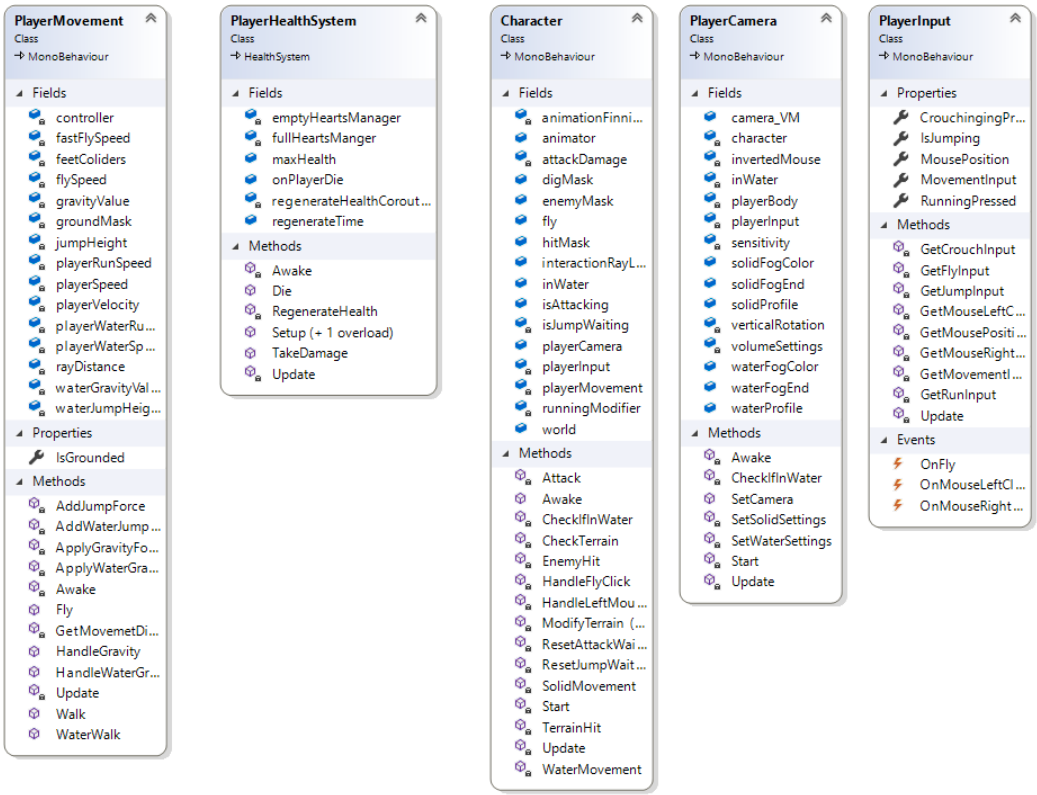


Figura 10 – Diagramele de clasă ale personajul jucabil

3.9 Inamicii

Inamicii au rolul de a oferi jucătorului un obiectiv și de a ridica gradul de dificultate al jocului. Aceștia pot să găsească o cale către jucător dacă acesta se află în raza lor de căutare. Inamicii pot ataca jucătorul atunci când se află în raza lor de acțiune. Inamicii au diferite animații pentru tipuri diferite de comportament (animații de locomoție, animații de atac, etc.).

Pentru a se mișca pe hartă, inamicii au în compoziția lor componenta NavMeshAgent, care poate mișca un obiect aflat pe un NavMeshSurface.

Există trei tipuri de inamici:

1. Un inamic de tip melee, care folosește modelul deb_Goblin01 din pachetul Goblin01 [10] aflat pe platforma Unity Asset Store. Animațiile folosite pentru acest tip de inamic au fost preluate de pe platforma Mixamo [11].
2. Un inamic de tip ranged care folosește modelul PT_Skeleton_Male_Modular din pachetul Lowpoly Medieval Skeleton [12] aflat pe platforma Unity Asset Store. Inamicul ține o arbaletă în mâna al cărui model se găsește în pachetul Low Poly Weapons [13] de pe platforma Unity Asset Store. Animațiile folosite pentru acest tip de inamoc au fost preluate de pe platforma Mixamo [11].
3. Un inamic de tip boss care folosește modelul PT_Plague_Doctor_Lantern din pachetul Lowpoly Medieval Plague Doctor [14] aflat pe platforma Unity Asset Store. Animațiile folosite pentru acest tip de inamoc au fost preluate de pe platforma Mixamo [11].

Un inamic se folosește de următoarele clase:

- Clasa Enemy, care inițializează toate celelalte clase cu valorile necesare
- Clasa EnemyAttack, care definește tipurile de atac ale inamicului
- Clasa EnemyDamageDealer, care care determină valorile daunelor provocate pentru jucător atunci când este lovit
- Clasa EnemyHealthSystem, care gestionază nivelul de viață al inamicului
- Clasa NavMeshEnemyMovement care gestionează cum se mișcă inamicul de tip NavMeshAgent

În figura 11 se pot observa diageamele claselor descrise mai sus.

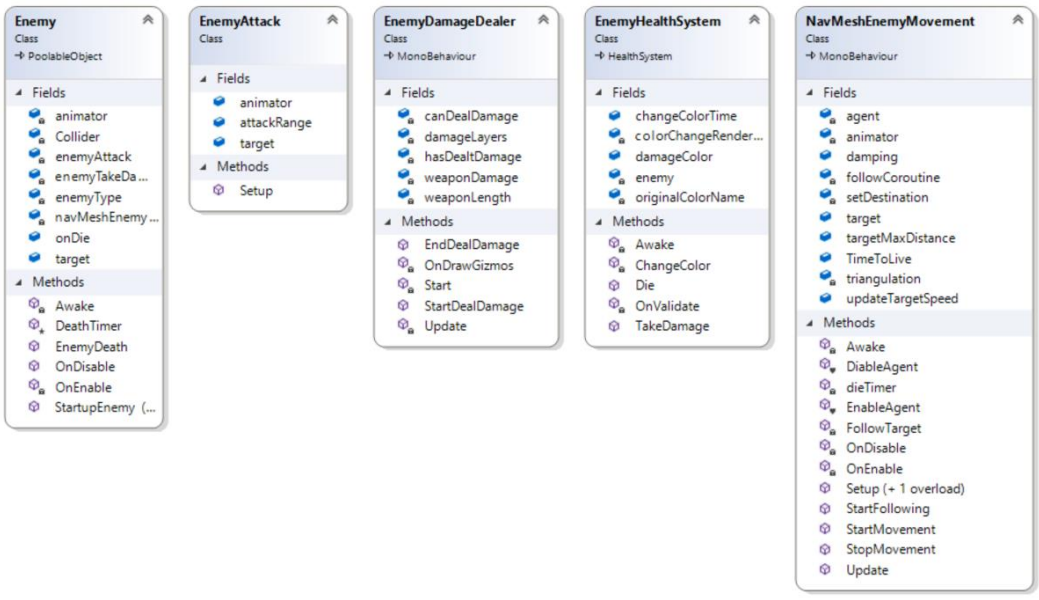


Figura 11 – Diagramele de clasă ale inamicilor

În timpul jocului primele două tipuri de inamici se vor spawna până când jucătorul a înfrânt un număr dat de inamici. În acel moment se va spawna inamicul de tip Boss. Odată înfrânt inamicul de tip boss, jocul intră în starea de pauză, iar jucătorul decide dacă trece la o altă scena, sau dacă explorează în continuare nivelul generat.

4 DETALII DE IMPLEMENTARE

În subcapitolele ce urmează vor fi prezentate implementările diferitelor sisteme relevante folosite pentru a genera procedural lumea în jurul jucătorului. Vor fi parcurse aspecte legate de implementarea blocurilor, implementarea chunk-urilor, implementarea sistemului de generare procedurală a lumii și implementarea sistemelor care gestionează lumea generată.

4.1 Descrierea sistemului de blocuri

Pentru a facilita crearea unei lumi generate procedural, trebuie stabilite câteva componente fundamentale. Acestea pot fi clasificate în linii mari în două categorii: un mecanism de generare și stocare a datelor în mod procedural și o metodă de interpretare a datelor generate pentru a construi o reprezentare tridimensională a lumii.

În această lucrare, metodologia utilizată implică folosirea unei cantități prestabilite de blocuri, denumite voxeli. Fiecare voxel este asociat cu texturi specifice și posedă atribute suplimentare care sunt utilizate în timpul generării plaselor tridimensionale. Aceste blocuri sunt organizate în bucăți pentru a permite paralelizarea atât a generării datelor, cât și a construcției ochiurilor tridimensionale. În capitolele următoare vor fi oferite mai multe detalii despre aceste sisteme.

4.1.1 Descrierea enumerației VoxelType

Pentru a avea o reprezentare bună a blocurilor, care să fie atât ușor de extins, cât și ușor de citit, și care sa nu ocupe un spațiu prea mare în memorie, s-a ales o enumerare pentru a ține evidența diferitelor tipuri de blocuri. Fiecare valoare din enumerare reprezintă un tip diferit de bloc.

Valoarea Nothing reprezintă un spațiu gol în care nu există niciun bloc. Valoarea "Air" reprezintă un voxel de aer. Celelalte valori reprezintă diferite tipuri de bloc lichid, cum ar fi Water, sau solid, cum ar fi Grass_Dirt, Dirt, Grass_Stone, Stone, TreeTrunk, TreeLeavesTransparent, TreeLeavesSolid, Sand și așa mai departe.

4.1.2 Descrierea clasei TextureData

Pentru a stoca texturile pentru fiecare bloc se folosește o clasă numită TextureData.

Această clasă are următoarele câmpuri:

- Câmpul voxelType este un VoxelType care stochează tipul de voxel pe care îl reprezintă aceste date de textură.
- Câmpurile up, down și side sunt Vector2Int care stochează coordonatele texturii pentru fețele de sus, de jos și, respectiv, laterale ale voxelului.
- Câmpul isSolid este un boolean care indică dacă voxelul este solid sau nu.
- Câmpul isTransparent este un boolean care indică dacă voxelul este sau nu transparent.
- Câmpul generatesCollider este un boolean care indică dacă voxelul generează sau nu o coliziune.
- Câmpul isLiquid este un boolean care indică dacă voxelul este lichid sau nu.
- Câmpul isDestructable este un boolean care indică dacă voxelul poate fi distrus de către jucător.

4.1.3 Atlasul de texturi

Pentru texturarea blocurilor din spatiul 3D s-a folosit un atlas de texturi. Un atlas de texturi este o imagine de mari dimensiuni care conține mai multe texturi mai mici dispuse într-un model de tip grilă [15]. Acesta este utilizat pentru a optimiza redarea graficii 2D și 3D prin reducerea numărului de comutări de texturi necesare în timpul redării. În loc să se lege texturi individuale pentru fiecare obiect, se leagă un singur atlas de texturi, iar coordonatele texturii pentru fiecare obiect sunt ajustate pentru a face referire la subregiunea corespunzătoare din atlas. Această tehnică poate îmbunătăți în mod semnificativ performanța prin reducerea numărului de schimbări de stare și de apeluri de desenare solicitate de pipeline-ul grafic.

Pentru a putea extrage texturile dorite din atlasul de texturi este necesar sa cunoastem coordonatele texturilor dorite. Coordonatele texturii, cunoscute și sub numele de coordonate UV, sunt o pereche de valori care specifică poziția unui punct pe o imagine de textură. Ele sunt utilizate pentru a cartografia textura pe suprafața unui obiect 3D. Coordonata orizontală este de obicei notată cu U, iar cea verticală cu V. Valorile U și V variază de la 1 la 1, (1,1) reprezentând colțul din stânga jos al texturii și (1,1) colțul din dreapta sus. Prin specificarea coordonatelor texturii pentru fiecare vertex al unui obiect 3D, pipeline-ul grafic poate determina ce parte a imaginii texturii trebuie aplicată fiecărei părți a suprafeței obiectului.

Atlasul de texturi folosit în această lucrare face parte din pachetul „Voxel Pack” [16], (a se vedea figura 12) oferit gratuit de care platforma Kenney Asset Store. Pachetul se află sub licență de tip Creative Commons CC0. Kenney Asset Store este un site web care oferă mii de asset-uri de joc gratuite pe care dezvoltatorii le pot folosi. Asset-urile disponibile pe magazin includ grafică 2D și 3D, audio și texturi 2. Magazinul oferă, de asemenea, un pachet all-in-one și instrumente pentru crearea de modele 3D care pot fi utilizate în majoritatea motoarelor de jocuri.

Figura 12 - Atlas-ul de texturi original

Acest atlas de texturi a fost modificat, fiind adăugate patru texturi noi folosite pentru partea subacvatică a jocului (a se vedea figura 13).

Figura 13 – Atlas de texturi modificat

4.1.4 Descrierea clasei VoxelDataSO

Clasa VoxelDataSO este folosită pentru a stoca date legate despre blocurile din joc. Această clasa moștenește clasa ScriptableObject din Unity. Clasa Scriptable Object „este un container de date care poate fi utilizat pentru a salva cantități mari de date, independent de instanțele de clasă” [17].

Câmpurile textureSizeX și textureSizeY sunt câmpuri float care stochează dimensiunea texturilor utilizate pentru blocurile din lumea jocului.

Câmpul textureDataList este o listă de obiecte TextureData care stochează datele de textură pentru fiecare tip de bloc.

Deoarece această clasă moștenește ScriptableObject, se poate crea un fișier care conține clasa pentru a stoca toate informațiile relevante referitoare la voxeli. Acest lucru poate fi realizat prin adăugarea atributului [CreateAssetMenu(fileName = "Voxel Data", menuName = "Data/Voxel Data")]] la clasa VoxelDataSO. Apoi, fișierul poate fi creat în cadrul meniului Unity, la fel ca și în cazul creării oricărui alt asset.

Aceste date sunt folosite de către clasa VoxelDataManager pentru a crea un dicționar care atribuie fiecărui tip din VoxelType o textura pentru toate cele 6 fețe ale unui bloc, precum și celelalte atribute semnificative pentru generarea modelului 3D (de exemplu daca blocul reprezintă un bloc lichid sau solid, dacă poate fi distrus de către jucător, ș.a.m.d.).

4.1.5 Descrierea clasei VoxelDataManager

Clasa VoxelDataManager are rolul de a converti lista stocată în VoxelDataSO într-un dictionar care poate fi folosit la runtime de catre algoritmii de generare procedurală a datelor și a modelului 3D a chunk-urilor. Această clasă moștenește clasa MonoBehaviour din Unity, deci poate fi atașată unui obiect din scenă.

Această soluție a fost folosită deoarece, în motorul de joc Unity, structura de date Dictionary nu poate fi accesată direct din Editor, iar complexitatea constantă în timp (O(1)) a accesului la datele furnizate de un dicționar este avantajoasă pentru recuperarea rapidă a datelor de textură conținute în obiectul VoxelDataSO. De asemena un alt avantaj în a folosi un dictionar este pentru a elimina cazul în care pentru un VoxelType au fost asociate doua TextureData în lista din VoxelDataSO. În acest caz prima asociere din listă este folosită, iar cea redundantă este ignorată.

Câmpul textureOffset este un float static care stochează valoarea de decalaj utilizată la calcularea coordonatelor texturii. Această valoare este folosită pentru a elimina liniile care pot să apară la granița dintre texturi.

Câmpul tileSize este un Vector2 static care stochează dimensiunea plăcilor de textură utilizate pentru blocurile din lumea jocului.

Câmpul voxelTextureDataDictionary este un dicționar static care mapează valorile VoxelType în obiecte TextureData. Acesta stochează datele de textură pentru fiecare tip de bloc din lumea jocului.

Câmpul textureData este un obiect VoxelDataSO care stochează datele blocului pentru lumea jocului.

Metoda Awake este apelată atunci când scriptul este activat. Aceasta inițializează câmpurile voxelTextureDataDictionary și tileSize folosind datele din câmpul textureData.

4.2 Descrierea sistemului de chunk-uri

4.2.1 Descrierea clasei ChunkData

Clasa ChunkData simbolizează o secțiune de date într-un mediu virtual bazat pe voxeli. Un chunk este o secțiune a mediului de joc reprezentată de o grilă 3D de voxeli.

Tipul fiecărui bloc din chunk este stocat în câmpul voxels, care este o matrice de VoxelType de dimensiune chunkSize.x * chunkSize.y * chunkSize.z.

Înălțimea terenului este păstrată în câmpul heightMap, care este un array de numere întregi, pentru fiecare punct x-z din chunk. Această matrice este folosită pentru a evita calcule redundante pentru aceleași poziții x-z dintr-un chunk, dar care au poziția y diferită.

Dimensiunile pe bază de voxel ale bucății sunt păstrate în câmpul chunkSize, care este un Vector3Int. Aceste valori sunt identice pentru fiecare obiect ChunkData și sunt atribuite la generarea lumii.

Obiectul World din care face parte acest chunk este menționat în proprietatea worldReference. Prin intermediul acestei variabile un chunk poate accesa date legate de alte chunk-uri vecine. Aceaste informații vor fi utile pentru generarea modelului 3D, acolo unde un bloc se află la granița unui chunk.

Poziția globală a unui chunk este retinută în worldPosition, care este un Vector3Int. Această poziție nu reprezintă poziția finală în scenă a unui chunk, ci reprezintă poziția unui chunk atunci când dimensiunea unui bloc este de 1x1x1.

Parametrii de producere a zgomotului care este utilizat pentru a construi topografia sunt stocați în câmpul noiseSettings.

Folosind un dicționar, câmpul outOfChunkBoundsVoxelDictionary sunt stocate pozitiile si tipul de date al bloc-urilor dar care se află în afara limitelor chunk-ului curent, dar care trebuie plasate într-un chunk vecin.

O valoare booleană în câmpul modifiedByPlayer arată dacă jucătorul a modificat acest chunk.

Constructorului i se furnizează trei argumente: chunkSize, world și worldPosition. Acesta creează noi array-uri pentru câmpurile voxels și heightMap pe baza dimensiunii specificate pentru chunk și inițializează câmpurile cu datele furnizate, iar campului worldReference ii este atribuită o referință care world.

4.2.2 Descrierea clasei statice Chunk

Pentru a putea procesa datele din ChunkData, a fost folosită o clasă statică Chunk care oferă metode de utilitate pentru lucrul cu obiectele ChunkData.

Astfel, avem următoarele metode:

1. Metoda LoopThroughVoxels primește ca argumente un obiect ChunkData și un delegat Action<Vector3Int>. Aceasta parcurge în buclă toți voxelele din chunk și efectuează acțiunea dată pe pozițiile lor.

2. Metoda GetEdgeNeighbourChunk primește ca argumente un obiect ChunkData și o poziție globală Vector3Int. Aceasta returnează o listă de obiecte ChunkData care reprezintă chunk-urile vecine de pe marginile bucății date, la poziția globală dată.

3. Metoda VoxelIsOnEdge primește ca argumente un obiect ChunkData și o poziție globală Vector3Int. Aceasta returnează un boolean care indică dacă voxelul aflat în poziția globală dată se află pe marginea bucății chunk-ului.

4. Metoda InRange primește ca argumente un obiect ChunkData și o poziție locală Vector3Int. Aceasta returnează un boolean care indică dacă poziția locală dată se află în limitele chunk-ului dat.

5. Metoda GetVoxelFromChunkCoordinates primește ca argumente un obiect ChunkData și o poziție locală Vector3Int. Aceasta returnează VoxelType la poziția locală dată în cadrul chunk-ului dat. În cazul în care poziția locală este în afara intervalului, se utilizează metoda WorldDataHelper.GetVoxelFromWorldCoorinates pentru a obține voxelul din coordonatele globale.

6. Metoda ChunkPositionFromVoxelWorldCoordinates primește ca argumente un obiect ChunkData și o poziție globală Vector3Int. Aceasta returnează poziția chunk-ului care conține poziția bloc-ul de la poziția dată.

7. Metoda SetVoxelFromChunkCoordinates primește ca argumente un obiect ChunkData, o poziție locală Vector3Int și un VoxelType. Aceasta stabilește voxelul din poziția locală dată în cadrul chunk-ului dat la tipul de voxel dat. În cazul în care poziția locală este în afara intervalului, se utilizează metoda WorldDataHelper.SetVoxelFromWorldCoordinates pentru a seta voxelul la coordonatele globale.

8. Metoda GetPositionFromIndex primește ca argumente un obiect ChunkData și un index de tip întreg. Aceasta returnează poziția locală a voxelului ca un Vector3Int la indexul dat în cadrul chunk-ului dat.

9. Metoda GetIndexFromPosition primește ca argumente un obiect ChunkData și o poziție locală Vector3Int. Aceasta returnează indexul voxelului aflat în poziția locală dată în cadrul obiectului ChunkData. Practic, este operația inversă a metodei GetPositionFromIndex.

10. Metoda GetVoxelInChunkCoordinates primește ca argumente un obiect ChunkData și o poziție Vector3Int. Aceasta returnează poziția locală a voxelului în cadrul chunk-ului dat care corespunde poziției globale date.

11. Metoda VoxelPosition primește ca argumente un obiect ChunkData și o poziție Vector3Int. Aceasta returnează poziția globală a voxelului la poziția locală dată în cadrul chunk-ului dat. Practic, este operația inversă a metodei GetVoxelInChunkCoordinates.

12. Metoda GetChunkMeshData primește ca argument un obiect ChunkData. Aceasta returnează un obiect MeshData care reprezintă datele de modelului 3D pentru chunk-ul dat. Aceasta utilizează metoda LoopThroughVoxels pentru a parcurge în buclă toți voxelele din chunk și metoda VoxelHelper.GetVoxelMeshData pentru a obține datele modelului 3D pentru fiecare voxel.

13. Metoda ChunkPositionFromVoxelCoords primește ca argumente un obiect World și o poziție Vector3Int. Aceasta returnează poziția chunk care conține poziția lumii voxelului dat.

4.3 Descrierea sistemului de desenare a chunk-urilor

4.3.1 Descrierea clasei MeshData

Clasa MeshData este folosită pentru a stoca informații legate despre modelul 3D al unui Chunk.

Câmpul verices este o listă de Vector3 care stochează pozițiile vârfurilor în modelul 3D.

Câmpul triangles este o listă de numere întregi care stochează indicii vârfurilor care alcătuiesc triunghiurile din modelul 3D.

Câmpul uv este o listă de Vector2 care stochează coordonatele texturii pentru fiecare vârf modelul 3D.

Deoarece un chunk poate fi compus din mai multe tipuri de teren și deoarece ne dorim un ca acele tipuri de teren să aibă un comportament diferit în spațiul de joc, clasa MeshData are următoarele câmpuri tot de tip MeshData: solidMesh, liquidMesh, underLiquidMesh și transparentMesh. Aceste date vor crea diferite modele 3D care vor avea un comportament diferit în spațiul de joc. De exemplu, materialul folosit pentru modelul generat din datele din liquidMesh va fi transparent, pe când materialele pentru celelalte modele 3D va fi opac. De asemenea, ne dorim această separație intre datele generate pentru a avea un control mai exact asupra tipurilor de geometrie peste care se poate genera meșa de navigare folosită de către inamici.

Constructorul primește un argument boolean isMainMesh. Dacă este adevărat, acesta creează noi obiecte MeshData pentru câmpurile solidMesh, liquidMesh, underLiquidMesh și transparentMesh. Câmpurile nou creeate vor fi sub-mesh-uri, și vor apela constructorul cu argumentul boolean isMainMesh setat false.

Această clasa are, de asemenea două metode, una pentru a adaugă vârfuri, iar cealaltă pentru a adăuga triunghiuri în listele lor respective:

1. Metoda AddVertex primește ca argument un vertex Vector3. Aceasta adaugă vârful dat la lista de vârfuri.

2. Metoda AddQuadTriangles adaugă două triunghiuri la lista de triunghiuri pentru a forma un quad folosind ultimele patru vârfuri adăugate la lista de vârfuri.

   Mai întâi se obține numărul curent de vârfuri din lista de vârfuri și este stocat în verticesCount. Apoi adaugă șase indici la lista de triunghiuri pentru a forma două triunghiuri. Primul triunghi este format din vârfurile de la indicii verticesCount - 4,

verticesCount - 3 și verticesCount - 2. Al doilea triunghi este format din vârfurile de la indicii verticesCount - 4, verticesCount - 2 și verticesCount - 1.

4.3.2 Descrierea clasei statice VoxelHelper – Generarea datelor pentru modelul 3D

Clasa statică VoxelHelper are rolul de a crea datele necesare pentru modelele 3D ale chunk-urilor. Cu ajutorul acestei clase se generează date doar pentru blocurile vizibile jucătorului pentru a nu irosi resurse desenând fețe pe care jucătorul nu ar trebui să le vadă. Această clasă este responsabilă pentru generarea vârfurilor, triunghiurilor și pentru maparea texturilor pentru fiecare față a blocurilor din spațiul de joc.

Pentru a putea genera aceste date într-un mod cât mai facil este nevoie de o enumerație auxiliara Direction care definește cele șase direcții posibile în spațiul 3D: sus, jos, dreapta, stânga, înainte și înapoi. Aceste direcții sunt apoi stocate într-un vector peste care se poate itera atunci când sunt generate datele pentru modelul 3D. Adițional, exista o funcție, GetDirectionVector, care primește ca argument o valoare Direction și returnează un Vector3Int care reprezintă vectorul unitar în direcția dată.

Metoda TexturePosition primește ca argumente o valoare Direction și o valoare VoxelType. Aceasta returnează un Vector2Int care reprezintă coordonatele texturii pentru direcția și tipul de voxel date. Aceasta utilizează o expresie switch pentru a returna coordonatele de textură corespunzătoare din VoxelDataManager.voxelTextureDataDictionary.

Metoda GetVoxelMeshData o metodă statică care primește ca argumente un obiect ChunkData, o poziție Vector3Int, un obiect MeshData și o valoare VoxelType. Aceasta returnează un obiect MeshData care reprezintă datele pentru generarea geometriei pentru blocul dat în cadrul chunk-ului dat.

Metoda verifică mai întâi dacă voxelType-ul dat este Air sau Nothing. În caz afirmativ, aceasta returnează meshData fără modificări. Nu este desenat nimic pentru aceste tipuri de blocuri.

Apoi, metoda trece în buclă prin fiecare direcție din matricea de direcții. Pentru fiecare direcție, se calculează coordonatele blocului vecin din direcția respectivă și se obține tipul de bloc cu ajutorul metodei Chunk.GetVoxelFromChunkCoordinates. Apoi se verifică dacă blocul vecin este solid, dacă are același tip de bloc ca și blocul dat sau dacă nu este nimic. Dacă oricare dintre aceste condiții este adevărată, se trece la următoarea iterație a buclei. Nu este desenat nimic în aceste cazuri.

În cazul în care niciuna dintre aceste condiții nu este adevărată, metoda apelează metoda MeshGenerationLogic pentru a actualiza meshData pentru direcția și tipul de voxel date.

În cele din urmă, după parcurgerea tuturor direcțiilor, metoda returnează datele actualizate pentru generarea modelului 3D pentru acel tip de bloc de la acea pozitie.

Metoda MeshGenerationLogic este o metodă statică care primește ca argumente o direcție, un obiect ChunkData, o poziție Vector3Int, un obiect MeshData, o valoare VoxelType și o valoare VoxelType pentru blocul vecin. Acesta returnează un obiect MeshData care reprezintă datele actualizate folosite pentru modelul 3D pentru blocul dat în cadrul chunk-ului dat. Un aspect important de reținut este ca această metodă calculează datele necesare pentru o singură față a unui bloc în direcția dată.

Metoda verifică mai întâi dacă voxelType dat este de tip Air sau Nothing. În caz afirmativ, aceasta returnează meshData dat neschimbat.

Metoda calculează apoi mai multe valori de decalaj pentru desenarea blocului și a texturii sale. Aceste valori depind de faptul dacă blocului este lichid sau nu și dacă are un voxel de aer deasupra sa. Aceste valori de decalaj sunt folosite pentru a desena nivelul apei mai jos decât restul blocurilor (dacă un bloc normal are înălțimea de 1 metru, atunci un bloc de apa cu vecinul în poziția de sus aer va avea înălțimea de 1.65 metrii) și pentru a desena fețele adiacente unui bloc de apa cu vecinul de sus aer diferit în funcție de nivelul apei.

Metoda verifică apoi dacă voxelul dat este lichid. Dacă vecinul aflat în poziția de sus a blocului curent este aer, atunci partea superioară a acelei fețe este desenată mai jos. În caz afirmativ, aceasta actualizează câmpul liquidMesh din datele de rețea date utilizând metoda GetFaceDataIn și returnează datele pentru modelul 3D actualizate.

În cazul în care voxelul dat nu este lichid și nici voxelul vecin nu este lichid, metoda verifică dacă voxelul dat este transparent. În caz afirmativ, aceasta actualizează câmpul transparentMesh din datele de rețea date, utilizând metoda GetFaceDataIn și returnează datele pentru modelul 3D actualizate. În caz contrar, se actualizează câmpul solidMesh al datelor meshData date cu ajutorul metodei GetFaceDataIn și se returnează datele meshData actualizate.

În cazul în care voxelul vecin este lichid, metoda calculează coordonatele voxelului de deasupra acestuia și obține tipul de voxel al acestuia. În cazul în care acest voxel nu este aer, metoda actualizează câmpul "underLiquidMesh" din datele pentru modelul 3D date cu ajutorul metodei GetFaceDataIn și returnează datele pentru modelul 3D actualizate.

În continuare, se verifică dacă direcția dată este în sus sau în jos. În caz afirmativ, aceasta actualizează câmpul underLiquidMesh din datele pentru modelul 3D date cu ajutorul metodei GetFaceDataIn și returnează datele pentru modelul 3D actualizate.

În această parte a metodei ne aflăm în situația în care fața pentru care calculăm datele este adiacentă unui bloc de apă care se află la suprafața (are vecinul de sus aer). Acest lucru înseamnă că pentru acea fată a blocului vor exista două seturi de triunghiuri care vor alcătui acea față. Partea superioară a feței se va afla în datele pentru modelul solid sau transparent, iar partea inferioară se va afla în datele pentru modelul subacvatic. Această separare ne oferă posibilitatea de a aplica diferite proprietăți modelelor 3D și materialelor aplicate modelelor 3D care vor fi generate. Astfel, putem asigura faptul că tranziția dintre partea subacvatică și

cea de uscat este una fără probleme care să aibă un impact major asupra graficii. Un exemplu de astfel de problemă este dacă partea superioara a unui bloc, care pentru jucător apare deasupra apei deci ar trebui sa facă parte din uscat, să folosească materialul pentru modelul subacvatic.

Metoda actualizează apoi valorile de decalaj pentru desenarea blocului și a texturii acestuia astfel încât să se deseneze doar partea superioara a feței. Se verifică dacă blocul dat este transparent. În caz afirmativ, actualizează câmpul transparentMesh din meshData dat, utilizând metoda GetFaceDataIn. În caz contrar, se actualizează câmpul solidMesh al datelor meshData date prin metoda GetFaceDataIn.

Apoi, metoda actualizează din nou valorile de decalaj astfel încât să se deseneze doar partea inferioară a feței și actualizează câmpul "underLiquidMesh" al datelor pentru modelul 3D date cu ajutorul metodei GetFaceDataIn. În cele din urmă, metoda returnează datele actualizate ale modelului 3D.

Procesul descris mai sus poate fi regăsit în figura 14.

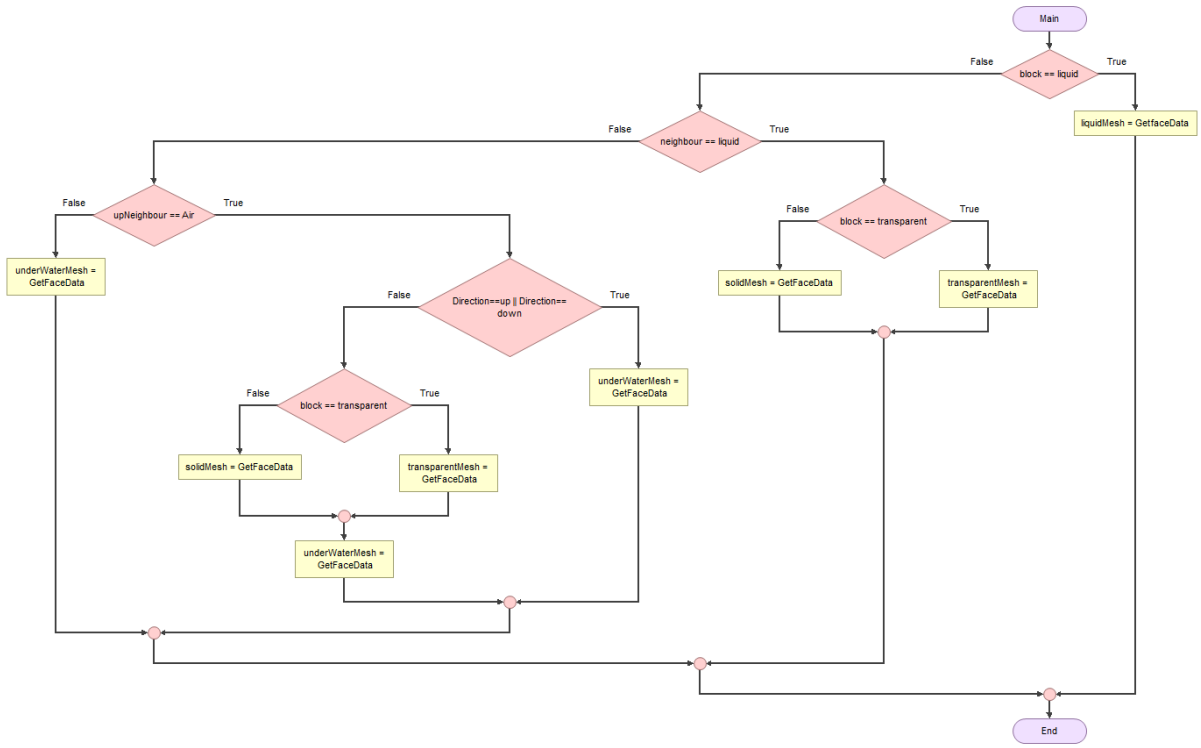


Figura 14 – Flowchart-ul MeshGenerationLogic

Metoda GetFaceDataIn primește ca argumente o direcție, un obiect ChunkData, o poziție Vector3, un obiect MeshData, o valoare VoxelType, două valori de decalaj Vector3 și două

valori de decalaj de textură Vector2. Aceasta returnează un obiect MeshData care reprezintă datele actualizate ale modelului 3D pentru blocul dat în cadrul chunk-ului dat.

Metoda apelează metoda GetFaceVertices pentru a adăuga vârfuri la datele meshData date. Apoi apelează metoda AddQuadTriangles pentru a adăuga triunghiuri la datele de meșă date. În cele din urmă, adaugă coordonate de textură la datele de plasă date cu ajutorul metodei FaceUVs și returnează datele de plasă actualizate.

Metoda GetFaceVertices este o metodă statică care primește ca argumente o direcție, un obiect ChunkData, o poziție Vector3, un obiect MeshData, o valoare VoxelType și două valori de decalaj Vector3. Aceasta adaugă vârfuri la obiectul MeshData dat pentru a reprezenta fața blocului dat în direcția dată.

Metoda calculează mai întâi coordonatele x, y și z ale poziției date. Apoi utilizează o instrucțiune de tip switch pentru a adăuga patru vârfuri la obiectul meshData dat, în funcție de direcția dată. Ordinea în care sunt adăugate vârfurile este importantă pentru calcularea normalelor și pentru redarea modelului 3D.

Pentru fiecare direcție, metoda calculează pozițiile celor patru vârfuri folosind valorile date ale poziției și ale decalajului. Apoi apelează metoda "AddVertex" din meshData pentru a adăuga fiecare vertex.

Metoda FaceUVs este o metodă statică care primește ca argumente o direcție, o valoare VoxelType și două valori de offset Vector3. Aceasta returnează o matrice de valori Vector2 care reprezintă coordonatele texturii pentru fața blocului dată în direcția dată.

Metoda creează mai întâi o matrice de valori Vector2 pentru a stoca coordonatele texturii. Apoi calculează poziția plăcii de textură pentru direcția și tipul de voxel date, utilizând metoda TexturePosition.

Metoda calculează apoi pozițiile și decalajele celor patru colțuri ale plăcii de textură: sud-est, nord-est, nord-vest și sud-vest. Se utilizează aceste poziții și decalaje pentru a calcula coordonatele texturii pentru fiecare colț.

Apoi, metoda scalează și decalează fiecare coordonată de textură folosind câmpurile tileSize și textureOffset din clasa VoxelDataManager. Metoda stochează coordonatele texturii calculate în matricea UVs.

Cum atlasul de texturi are coordonate între 1 și 1 pe ambele axe, iar numărul de texturi pe ambele axe este de 10x10, variabila tileSize va avea valoarea (1.1, 1.1). Valoarea textureOffset este folosită pentru a evita folosirea eronată a unor pixeli din altă textură adiacentă pentru textura curentă. Aceasta valoare de decalaj este adăugată sau scăzută in funcție de coltul pentru care se fac calculele. De exemplu, pentru colțul dreapta-jos, este scăzut acel decalaj

pentru coordonata x, iar pentru coordonata y este adăugat acel decalaj. Valoarea pentru acest decalaj este 1.001.

4.3.3 Descrierea clasei CustomRenderer

Clasa ChunkRenderer este folosită pentru a desena modelele 3D care fac parte dintr-un chunk. Această clasă moștenește clasa MonoBehaviour din Unity.

Clasa CustomRenderer are două atribute RequireComponent care specifică faptul că necesită o componentă MeshFilter, o componentă MeshRenderer și o componentă MeshCollider.

Metoda Awake este apelată atunci când se încarcă instanța de script. Aceasta obține componentele MeshFilter și MeshCollider atașate la același obiect de joc ca și acest script și le atribuie câmpurilor corespunzătoare. De asemenea, atribuie plasa filtrului de plasă la câmpul mesh.

Metoda RenderMesh primește ca argument un obiect MeshData. Aceasta actualizează plasa cu datele din obiectul MeshData. Aceasta face acest lucru prin ștergerea modelului 3D anterior și prin stabilirea vârfurilor, triunghiurilor și a valorilor UV la valorile corespunzătoare din datele din obiectul MeshData. Apoi recalculează normalele ochiului de plasă. În cazul în care există vârfuri în datele de plasă, se creează, de asemenea, o nouă plasă de coliziune și setează verticile și triunghiurile sale la valorile corespunzătoare din datele de plasă și se recalculează normalele meșei de coliziune.

4.3.4 Descrierea clasei ChunkRenderer

Clasa ChunkRenderer este folosită pentru a desena modelele 3D din care este alcătuit un chunk. Această clasă moștenește clasa MonoBehaviour din Unity.

Clasa ChunkRenderer conține mai multe câmpuri și proprietăți publice:

- solidRenderer, de tip CustomRenderer, folosit pentru a desena blocurile solide aflate pe uscat
- underWaterRenderer, de tip CustomRenderer, folosit pentru a desena blocurile solide aflate pe sub apă
- liquidRenderer, de tip CustomRenderer, folosit pentru a desena blocurile lichide
- transparentRenderer, de tip CustomRenderer, folosit pentru a desena blocurile transparente
- ChunkData, care reține datele pentru blocurile din acel chunk
- modifiedByPlayer, un boolean care indică faptul că acel chunk a fost modificat de jucător

De asemenea, clasa definește mai multe metode folosite pentru a desena modelele 3D:

- Metoda InitialiseChunk primește ca argument un obiect ChunkData. Aceasta stabilește valoarea proprietății ChunkData la datele date.
- Metoda RenderMesh primește ca argument un obiect MeshData. Aceasta desenează diferitele tipuri de modele 3D conținute în datele privind modelele 3D, utilizând sistemele de redare corespunzătoare.
- Există două supraîncărcări ale metodei UpdateChunk.
    - Prima supraîncărcare nu acceptă niciun argument și actualizează chunk-ul prin apelarea metodei GetChunkMeshData pe datele chunk și prin transmiterea datelor de plasă rezultate către metoda RenderMesh.
    - A doua supraîncărcare primește ca argument un obiect MeshData și actualizează chunk-ul trecând datele MeshData respective la metoda RenderMesh.

4.4 Descrierea sistemului de gestionare a lumii

4.4.1 Descrierea clasei WorldDataHelper

Clasa WorldDataHelper ajută în gestionarea datelor create, această clasă are rolul de a calcula ce date trebuie create, ce date trebuie modificate și ce date trebuie să fie șterse.

Pentru a obține comportamentul dorit sunt folosite următoarele metode:

- Metoda ChunkPositionFromVoxelCoords primește ca argumente un obiect World și o poziție Vector3Int și returnează poziția chunk corespunzătoare coordonatelor blocului dat. Aceasta face acest lucru prin împărțirea coordonatelor voxelului la dimensiunea chunk-ului în fiecare dimensiune, apoi se ia partea întreagă a împărțirii. Rezultatul este apoi înmulțit cu dimensiunea chunk-ului.

- Metoda GetChunkPositionsAroundPlayer primește ca argumente un obiect World și o poziție a jucătorului Vector3Int și returnează o listă de poziții chunk în jurul jucătorului. Aceasta calculează pozițiile de început și de sfârșit ale intervalului de chunk-uri care trebuie create pe baza poziției jucătorului, a intervalului de desenare a chunk-urilor și a dimensiunii chunk-urilor. Apoi, se iterează peste acest interval și se adaugă pozițiile chunk-urilor la o listă folosind metoda ChunkPositionFromVoxelCoords. Intervalul este alcătuit este calculat folosind câmpul chunkDrawingRange din setarile de generare a lumii. Câmpul chunkDrawingRange determina câte chunk-uri se vor genera în jurul jucătorului pe toate cele 3 axe. De exemplu, putem desena mai multe chunk-uri pe orizontala și în adâncime decât desenăm pe înalțime.

- Metoda GetDataPositionsAroundPlayer face același lucru precum metoda GetChunkPositionsAroundPlayer, doar că intervalul de desenare este mărit cu o

unitate. Acest lucru este făcut deoarece, pentru a genera corect modele 3D este necesar ca datele pentru chunk-urile aflate la marginea lumii sa fie calculate. Astfel evitam recalcularea modelelor 3D deja existente atunci când noi chunk-uri sunt calculate.

- Metoda SelectPositonsToCreate primește ca argumente un obiect World.WorldData, o listă a tuturor pozițiilor de chunk-uri necesare și o poziție a jucătorului Vector3Int. Aceasta returnează o listă de poziții chunk care trebuie să fie create. Aceasta face acest lucru prin filtrarea listei tuturor pozițiilor chunk necesare pentru a include doar pozițiile care nu se află deja în dicționarul chunkDictionary, care reține toate chunk-urile care au un model 3D și datele legate de blocurile componente. Apoi ordonează lista rezultată în funcție de distanța față de poziția jucătorului.

- Metoda SelectDataPositonsToCreate este similară metodei SelectPositonsToCreate, dar returnează pozițiile datelor chunk-urilor (ChunkData) în loc de pozițiile chunk. Aceasta primește ca argumente un obiect World.WorldData, o listă cu toate pozițiile de date necesare și o poziție a jucătorului Vector3Int. Aceasta filtrează lista tuturor pozițiilor de date necesare pentru a include doar pozițiile care nu se află deja în dicționarul de date al chunk-urilor (chunkDataDictionary) și ordonează lista rezultată în funcție de distanța față de poziția jucătorului.

- Metoda GetUnnededDataprimește ca argumente un obiect World.WorldData și o listă a tuturor pozițiilor chunk necesare. Aceasta returnează o listă de poziții de date care nu mai sunt necesare. Aceasta face acest lucru prin filtrarea cheilor din dicționarul de date chunk pentru a include doar pozițiile care nu se află în lista tuturor pozițiilor chunk necesare.

- Metoda GetUnnededChunks este similară metodei GetUnnededData, dar returnează pozițiile chunk-urilor în loc de pozițiile datelor. Aceasta primește ca argumente un obiect World.WorldData și o listă a tuturor pozițiilor chunk necesare. Aceasta filtrează cheile dicționarului de chunk-uri pentru a include numai pozițiile care nu se află în lista tuturor pozițiilor de chunk-uri necesare.

- Metoda RemoveChunk care primește ca argumente un obiect World și o poziție Vector3Int. Aceasta elimină din lume chunk-ul aflat în poziția dată, dacă acesta există. În acest scop, verifică dacă dicționarul chunk conține o valoare pentru poziția dată, utilizând metoda TryGetValue. În caz afirmativ, se elimină chunk-ul din sistemul de randare a lumii și din dicționarul de chunk-uri folosind metodele RemoveChunk și, respectiv, TryRemove.

- Metoda RemoveChunkData primește ca argumente un obiect World și o poziție Vector3Int. Aceasta elimină din dictionarul chunkDataDictionary datele de tip chunk de la poziția dată, dacă acestea există. Pentru a face acest lucru, se apelează metoda TryRemove pe dicționarul de date chunkDataDictionary cu poziția dată.

- Metoda SetVoxelFromWorldCoordinates primește ca argumente un obiect World, o poziție în lume Vector3Int și un VoxelType. Aceasta stabilește blocul din poziția lumii dată la tipul de voxel dat. Pentru a face acest lucru, se obțin datele chunk pentru poziția dată a lumii folosind metoda GetChunkDataFromWorldCoordinates. În cazul în care datele chunk nu sunt nule, se calculează poziția locală a voxelului în cadrul chunk-ului cu ajutorul metodei GetVoxelInChunkCoordinates ș i setează blocul din acea poziție locală la tipul de bloc dat cu ajutorul metodei SetVoxelFromChunkCoordinates din clasa Chunk.

- Metoda GetVoxelFromWorldCoorinates primește ca argumente un obiect World și o poziție în lume Vector3Int. Aceasta returnează tipul de bloc la poziția lumii dată. Pentru a face acest lucru, se obțin datele de tip "chunk" pentru poziția dată a lumii folosind metoda GetChunkDataFromWorldCoordinates. În cazul în care datele chunk nu sunt nule, se calculează poziția locală a voxelului în cadrul chunk-ului prin metoda GetVoxelInChunkCoordinates și se returnează tipul de bloc la acea poziție locală prin metoda GetVoxelFromChunkCoordinates din clasa Chunk.

- Metoda GetChunkDataFromWorldCoordinates primește ca argumente un obiect World și o poziție în lume Vector3Int. Aceasta returnează datele de tip "chunk" pentru poziția lumii dată. Pentru aceasta, se calculează poziția chunk pentru poziția lumii dată folosind metoda ChunkPositionFromVoxelCoords și apoi se verifică dacă există o valoare pentru acea poziție chunk în dicționarul de date chunkDataDictionary folosind metoda TryGetValue.

- Metoda GetChunk primește ca argumente un obiect World și o poziție Vector3Int. Aceasta returnează chunk-ul din poziția respectivă dacă există sau, în caz contrar, este nul. Pentru aceasta, verifică dacă există o valoare pentru poziția respectivă în dicționarul chunkDictionary folosind metoda ContainsKey.

4.4.2 Descrierea clasei WorldSettings

Clasa moștenește clasa ScriptableObject din Unity si poate fi creat ca și un asset.

Clasa conține mai multe câmpuri publice care definesc:

- poziția de pornire

- dimensiunea unui chunk
- dimensiunea unui bloc
- dimensiunile harții în chunk-uri
- decalajul global folosit pentru generarea de zgomot neted
- pragul de apă
- dimensiunile hărții în blocuri.

4.4.3 Descrierea clasei WorldRenderer

Clasa WorldRenderer este folosită pentru a gestiona și pentru a desena chunk-urile din spațiul de joc. Clasa WorldRenderer moștenește din MonoBehaviour.

Această clasă se folosește de tehnica Object Pooling. Object Pooling este o modalitate de a optimiza proiecte și de a reduce overhead-ul de pe CPU atunci când trebuie create și distruse obiecte din spațiul de joc în mod repetat [18].

Clasa WorldRenderer conține mai multe câmpuri publice:

- chunkPrefab este de tip GameObject și reprezintă prefabricatul utilizat pentru a crea noi Chunk-uri.
- chunkPool este de tip Queue<ChunkRenderer> și reprezintă un grup de redare a chunk-urilor care pot fi reutilizate.
- RenderedChunks este de tip GameObject și reprezintă obiectul părinte pentru toate bucățile redate.

Metoda RenderChunk primește ca argumente un obiect ChunkData, o poziție Vector3Int și un obiect MeshData. Aceasta returnează un obiect ChunkRenderer pentru datele chunk date la poziția dată și cu datele mesh date.

- În acest scop, verifică dacă există vreun obiect ChunkRenderer disponibil în coada chunkPool. În cazul în care există unul, se scoate din coadă și setează poziția acestuia la poziția scalată cu dimensiunea blocurilor. În cazul în care nu există nici un obiect ChunkRenderer în coada, se instanțiază un nou ChunkRenderer din prefabricatul bucăților de date la poziția redimensionată cu dimensiunea blocurilor și se obține componenta ChunkRenderer a acestuia.
- Metoda apelează apoi metoda "InitialiseChunk" pe obiectul ChunkRenderer nou sau pe cel reutilizat, având ca argument datele de redare a chunk-ului dat. Apoi apelează metoda "UpdateChunk" a obiectului ChunkRenderer cu datele MeshData ca argument. Aceasta stabilește starea activă a obiectului de joc al chunk renderer-ului la true și îi stabilește numele la reprezentarea șirului de caractere al poziției.

Metoda "RemoveChunk" primește ca argument un obiect ChunkRenderer. Aceasta elimină obiectul ChunkRenderer dat prin stabilirea stării active a acestuia la false și prin reintroducerea lui în coada chunkPool pentru a fi reutilizat.

4.4.4 Descrierea clasei World

Clasa World este are rolul de a gestiona crearea datelor generate procedural din jurul jucătorului. De asemenea această clasă are rolul de a gestiona modificarea acestor date de către jucător (de a începe procesul de modificare a datelor din jurul jucătorului). Clasa World moștenește clasa MonoBehaviour din Unity și este atașată unui obiect în scenă.

Pentru a îndeplini acest rol, clasa World are definite următoarele câmpuri si proprietați:

- worldSettings, o structură de date care reține setarile folosite pentru generarea lumii,
- worldRenderer, o clasă folosită pentru a desena modelele 3D din datele generate
- terrainGenerator, o clasă responsabila pentru a genera datele pentru blocurile dintr-un Chunk
- OnWorldCreated, un UnityEvent care este invocat atunci când lumea este generată pentru prima dată
- OnNewChunksGenerated, un UnityEvent care este invocat atunci când sunt generate noi chunk-uri
- worldData, o structură care reține dicționarele folosite pentru a gestiona datele create (chunkDataDictionary) și datele pentru modelele 3D (chunkDictionary)
- IsWorldCreated, un boolean care ne indică dacă lumea a fost generată pentru prima dată sau nu.
- stringSeed este de tip string și reprezintă o valoare de semințe introdusă de utilizator.
- intSeed este de tip int și reprezintă valoarea întreagă a seminței derivată din valoarea inițială a șirului.
- inputField este de tip TMP_InputField și reprezintă un câmp de intrare în care utilizatorul poate introduce valoarea pentu stringSeed.

De asemenea, definește două structuri imbricate: WorldGenerationData și WorldData.

Structura WorldGenerationData conține patru câmpuri: chunkPositionsToCreate, chunkDataPositionsToCreate, chunkPositionsToRemove și chunkDataToRemove. Aceste câmpuri reprezintă liste de poziții chunk de creat, poziții chunk data de creat, poziții chunk de eliminat și chunk data de eliminat.

Structura WorldData conține trei câmpuri: chunkDataDictionary, chunkDictionary și worldSettings. Câmpul chunkDataDictionary este de tip ConcurrentDictionary<Vector3Int, ChunkData> și reprezintă un dicționar care face corespondența între pozițiile chunk și datele chunk. Câmpul chunkDictionary este de tip ConcurrentDictionary<Vector3Int, ChunkRenderer> și reprezintă un dicționar care face legătura între pozițiile chunk și dispozitivele de redare a chunk. Câmpul worldSettings este de tip WorldSettings și reprezintă setările pentru lume.

De asemenea, clasa World conține definiții pentru mai multe metode care ajută în gestionarea lumii:

- Metodă GetWorldGenerationDataAroundPlayer. Această metodă primește ca argument poziția jucătorului și returnează o nouă instanță a structurii WorldGenerationData cu câmpurile sale setate în funcție de poziția jucătorului. Aceasta face acest lucru prin apelarea mai multor metode din clasa WorldDataHelper pentru a obține toate pozițiile chunk necesare, toate pozițiile de date chunk necesare, pozițiile chunk de creat, pozițiile de date chunk de creat, pozițiile chunk de eliminat și datele chunk de eliminat.
- Metoda GetSeed nu acceptă niciun argument și atribuie valoarea textului din câmpul de intrare, inputField, în câmpul stringSeed.
- Metoda "SetSeed" nu acceptă argumente și setează valoarea seed-ului pentru lume. Pentru aceasta, se verifică dacă câmpul stringSeed este un șir gol. În caz afirmativ, aceasta generează un seed aleatoriu folosind o nouă instanță a clasei System.Random. În cazul în care nu este un șir gol, metoda calculează codul hash al lui stringSeed și îl atribuie câmpului intSeed. Apoi creează o nouă instanță a clasei System.Random folosind seed-ul intSeed și generează decalaje aleatorii pentru dimensiunile x, y și z. Atribuie aceste decalaje atât proprietății mapSeedOffset a obiectului world settings.

- Metoda Awake inițializează proprietatea worldData cu o nouă instanță a structurii WorldData. Aceasta setează câmpul worldSettings al noii instanțe WorldData la valoarea câmpului worldSettings al acestui obiect World. De asemenea, inițializează câmpurile chunkDataDictionary și chunkDictionary ale noii instanțe WorldData cu noi instanțe ale claselor ConcurrentDictionary<Vector3Int, ChunkData> și, respectiv, ConcurrentDictionary<Vector3Int, ChunkRenderer>. De asemenea, valoarea proprietății IsWorldCreated este setată la false. Apoi, aceasta apelează metoda CalculateVoxelSizeInverse, care calculează vectorul 1/voxelSize și stabilește valorile proprietăților minMapDimensions și maxMapDimensions ale obiectului World Settings prin scalarea dimensiunilor hărții corespunzătoare (unde se considera că blocurile sunt de dimensiune de 1x1x1) cu dimensiunea unui bloc.

- Metoda GenerateWorld este o metodă asincronă care nu acceptă argumente și returnează void. Aceasta stabilește valoarea proprietății IsWorldCreated la false, apelează metodele GetSeed și SetSeed pentru a obține și a seta seed-ul pentru lume și apoi apelează metoda GenerateWorld cu proprietatea startingPosition a obiectului de worldSettings ca argument.

- Metoda GenerateWorld este o metodă asincronă care primește ca argument o poziție Vector3Int și returnează un Task. Aceasta generează lumea din jurul poziției date.

- Metoda începe prin apelarea metodei GetWorldGenerationDataAroundPlayer cu poziția dată ca argument și așteaptă rezultatul.
- Apoi trece în revistă pozițiile chunk-urilor care trebuie eliminate și datele chunk-urilor care trebuie eliminate din datele de generare a lumii returnate și apelează metodele RemoveChunk și RemoveChunkData din clasa WorldDataHelper pentru a elimina acele chunk-uri și date chunk.
- Aceasta declară o variabilă locală de tip ConcurrentDictionary<Vector3Int, ChunkData> pentru a păstra datele calculate ale bucăților. Apoi apelează metoda CalculateWorldChunkData cu pozițiile de date ale bucăților de creat din datele de generare a lumii ca argument și așteaptă rezultatul. În cazul în care se produce o excepție, se înregistrează un mesaj și se returnează.
- Metoda iterează apoi peste datele chunk calculate și le adaugă la dicționarul de date chunk al obiectului de date globale folosind metoda TryAdd.
- Apoi sunt parcurse toate valorile din chunkDataDictionary pentru a adăuga în chunk-urile potrivite blocurile care s-au generat în afara limitelor unui chunk (astfel aparținând de fapt altui chunk vecin). Această operatie este efectuată de metoda AddOutOfChunkBoundsVoxel
- Se creează o nouă variabilă locală de tip ConcurrentDictionary<Vector3Int, MeshData> pentru a păstra datele MeshData calculate. Obține o listă de date pe bucăți pentru a le reda prin filtrarea și selectarea valorilor din dicționarul de date pe bucăți în funcție de faptul dacă cheile lor se află sau nu în lista de poziții pe bucăți care trebuie create din datele de generare a lumii.
- Metoda apelează apoi metoda CreateMeshData cu lista de date de redare ca argument și așteaptă rezultatul. În cazul în care se aruncă o excepție, se înregistrează un mesaj și se returnează.
- În cele din urmă, se pornește o corutină folosind metoda ChunkCreationCoroutine cu datele de plasă calculate ca argument pentru a desena chunk-urile în spațiul de joc.

- Metoda ChunkCreationCoroutine este o corutină care primește ca argument un ConcurrentDictionary<Vector3Int, MeshData> și returnează un IEnumerator. Aceasta creează obiectele 3D din datele aflate în dictionarul dat ca argument.
  - Metoda ChunkCreationCoroutine începe prin parcurgerea perechilor cheie-valoare din dicționarul dat. Pentru fiecare pereche cheie-valoare, se obțin poziția și MeshData din cheie și, respectiv, din valoare. Metoda apelează apoi metoda RenderChunk a obiectului de redare a lumii (WorldRenderer), având ca argumente datele chunk pentru poziția respectivă, poziția și datele de meșă. Obiectul ChunkRenderer returnat de această metoda este apoi stocat în dictionarul de chunk-uri global.
  - Metoda cedează apoi controlul înapoi către Unity prin returnarea unei noi instanțe a clasei WaitForEndOfFrame. Acest lucru permite Unity să redea un cadru înainte de a relua execuția acestei corutine. Acest lucru este

implementat în acest mod pentru a nu bloca thread-ul principal cu desenarea de chunk-uri noi.
    - După ce toate iterațiile sunt finalizate se verifică dacă valoarea proprietății IsWorldCreated este falsă. În caz afirmativ, o stabilește la true și invocă evenimentul OnWorldCreated dacă acesta nu este nul.
- Metoda AddOutOfChunkBoundsVoxel primește ca argument un obiect ChunkData și adaugă în lume blocurile care se află în afara limitelor chunk-ului în chunk-ul potrivit.
    - Metoda începe prin crearea unei noi variabile locale de tip Dictionary<Vector3Int, VoxelType> pentru a păstra voxelele plasate. Apoi, ea trece prin perechile cheie-valoare din outOfChunkBoundsVoxelDictionary al datelor de tip chunk date. Pentru fiecare pereche cheie-valoare, se obține poziția lumii și tipul de voxel din cheie și, respectiv, din valoare.
    - Metoda apelează apoi metoda SetVoxelFromWorldCoordinates pentru acest obiect World, având ca argumente poziția lumii și tipul de voxel. Dacă această metodă returnează true, se adaugă poziția lumii și tipul de voxel la dicționarul de blocuri plasate.
    - După ce toate iterațiile sunt finalizate, metoda stabilește valoarea outOfChunkBoundsVoxelDictionary a datelor de bucată date într-un nou dicționar care conține numai perechi cheie-valoare din dicționarul original ale căror chei nu se află în dicționarul blocuri plasate, astfel blocurile pot fi plasate la un alt apel al funcției GenerateWorld dacă nu putut fi plasați.

- Metoda CalculateWorldChunkData primește ca argument o listă de poziții Vector3Int de creare a datelor de tip chunk și returnează un Task care reprezintă operațiunea asincronă de calculare a datelor de tip chunk pentru pozițiile respective.
    - Metoda începe prin crearea unei noi variabile locale de tip ConcurrentDictionary<Vector3Int, ChunkData> pentru a păstra datele chunk calculate. Apoi returnează un nou Task care este creat prin apelarea metodei Task.Run cu un delegat și taskTokenSource.Token ca argumente.
    - Delegatul transmis metodei Task.Run apelează metoda iterează peste pozițiile date de date ale chunk-urilor. Pentru fiecare poziție, se verifică dacă tokenul de anulare se află în starea de anulare. În caz afirmativ, se aruncă o excepție OperationCancedledException. Apoi creează un nou obiect ChunkData pentru poziția respectivă și apelează metoda GenerateChunkData pe obiectul generator de teren cu aceste date chunk ca argument. După generarea datelor se apelează metoda TryAdd a dicționarului cu poziția și noile date de tip chunk ca argumente.
    - După ce toate iterațiile sunt finalizate, delegatul returnează dicționarul.

- Metoda CreateMeshData primește ca argument o listă de obiecte ChunkData de desenat și returnează o sarcină care reprezintă operațiunea asincronă de creare modelelor 3D pentru acele obiecte de date ChunkData.
    - Metoda începe prin crearea unei noi variabile locale de tip ConcurrentDictionary<Vector3Int, MeshData> pentru a păstra datele folosite pentru a crea datele 3D calculate. Apoi returnează o nouă sarcină care este creată prin apelarea metodei Task.Run cu un delegat și taskTokenSource.Token ca argumente.
    - Delegatul transmis metodei Task.Run parcurge obiectele de date ale chunk-urilor date. Pentru fiecare obiect de date de tip chunk, se verifică dacă tokenul de anulare se află în starea de anulare. În caz afirmativ, se aruncă o excepție OperationCancedledException. Apoi, se apelează metoda GetChunkMeshData a clasei Chunk cu acel obiect de date de tip chunk ca argument. După generarea datelor se apelează metoda TryAdd a dicționarului cu poziția și noile date de tip MeshData ca argumente.
    - După ce toate iterațiile sunt finalizate, delegatul returnează dicționarul de ochiuri.
- Metoda LoadAdditionalChunksRequest este o metodă asincronă care primește ca argument o poziție Vector3 și returnează void. Aceasta generează chunk-uri suplimentare în jurul poziției date.
    - Metoda LoadAdditionalChunksRequest începe prin a apelea metoda GenerateWorld cu poziția dată rotunjită ca argument și așteaptă rezultatul. După finalizarea metodei GenerateWorld, aceasta invocă evenimentul OnNewChunksGenerated, în cazul în care acesta nu este nul.
- Metoda SetVoxel primește ca argumente un obiect RaycastHit și un VoxelType și returnează un bool. Aceasta stabilește voxelul din poziția lovită de raycast la tipul de voxel dat.
    - Metoda SetVoxel începe prin obținerea componentei ChunkRenderer a părintelui transformării obiectului de joc lovit de raycast. Dacă această componentă este nulă, se returnează false.
    - Apoi, metoda apelează metoda GetBlockPosition cu ajutorul argumentului dat de raycast hit. În cazul în care tipul de voxel dat este "air", se trece în revistă toate direcțiile din tabloul VoxelHelper.directions. Pentru fiecare direcție, se calculează coordonatele voxelului vecin din direcția respectivă și se obține tipul de voxel cu ajutorul metodei GetVoxelFromWorldCoorinates din clasa WorldDataHelper. Dacă acest tip de voxel este apă, se stabilește tipul de voxel la apă. Astfel, dacă un bloc vecin este de tipul apă, în loc de aer, blocul va deveni apă.
    - Metoda apelează apoi metoda SetVoxelFromWorldCoordinates din clasa WorldDataHelper cu acest obiect lume, poziția lumii și tipul de voxel ca argumente. Aceasta stabilește valoarea proprietății modifiedByPlayer a

dispozitivului de redare a bucăților la true. Apoi, se apelează metoda CheckEdges cu datele chunk din ChunkRenderer și poziția lumii ca argumente. Metoda CheckEdges verifică daca trebuie actualizate și modelele 3D ale chunk-urilor vecine. Aceste chunk-uri ar trebui actualizate atunci când un bloc de la marginile unui chunk sunt modificate.

  - Metoda apelează apoi metoda "UpdateChunk" a obiectului ChunkRenderer. Apoi este invocat evenimentul OnChunkUpdate dacă acesta nu este nul. În cele din urmă, se returnează true.

- Prima metodă CheckVoxel primește un obiect RaycastHit și doi parametri de ieșire de tip Vector3Int și, respectiv, ChunkRenderer. Aceasta returnează un VoxelType. Aceasta verifică voxelul din poziția lovită de raycast și returnează tipul său de voxel.
  - Metoda începe prin a obține componenta ChunkRenderer a părintelui transformării obiectului de joc lovit de rază. Aceasta atribuie această componentă parametrului "chunk out". Dacă această componentă este nulă, setează valoarea parametrului pos out la un nou Vector3Int cu toate componentele setate la -1 și returnează VoxelType.Nothing.
  - Metoda apelează apoi metoda GetBlockPosition cu ajutorul unui argument dat ca rezultat al raycast-ului și atribuie poziția lumii returnată unei variabile locale. Ea stabilește valoarea parametrului pos out la această poziție globală. Apoi apelează metoda GetVoxelFromWorldCoorinates a clasei WorldDataHelper cu acest obiect lume și poziția lumii ca argumente și returnează rezultatul.
- A doua metodă CheckVoxel primește ca argument o poziție Vector3 și returnează un VoxelType. Aceasta verifică voxelul din poziția dată și returnează tipul său de voxel.
  - Metoda începe prin apelarea metodei GetBlockPosition cu poziția dată ca argument și prin atribuirea poziției mondiale returnate la o variabilă locală. Apoi apelează metoda GetVoxelFromWorldCoorinates a clasei WorldDataHelper cu acest obiect lume și poziția lumii ca argumente și returnează rezultatul.
- Metoda CheckEdges primește ca argumente un obiect ChunkData și o poziție mondială Vector3Int.
  - Aceasta verifică dacă voxelul de la poziția mondială dată se află pe o margine a datelor de tip chunk date. În caz contrar, se returnează imediat. În caz afirmativ, se obține o listă de obiecte de date Chunk vecine folosind metoda GetEdgeNeighbourChunk a clasei Chunk cu datele Chunk și poziția lumii date ca argumente.
  - Metoda trece apoi prin această listă de obiecte de date ale chunk-urilor vecine. Pentru fiecare obiect de date chunk vecin, se verifică dacă acesta este nul. Dacă nu este nul, se apelează metoda GetChunk a clasei WorldDataHelper, având ca argumente acest obiect world și poziția în lume a datelor chunk vecine.

Metoda GetChunk întoarce un obiect ChunkRenderer. În cazul în care acest chunk renderer nu este nul, se apelează metoda UpdateChunk a acestuia.

- Prima metodă GetBlockPosition primește ca argument un obiect RaycastHit și returnează un Vector3Int. Aceasta calculează poziția blocului blocului lovit de raycast.
    - Metoda începe prin scalarea punctului și a normalei obiectului lovit de raycast dat cu dimensiunea inversă a blocului și, respectiv, cu dimensiunea voxelului din setările acestei lumi. Apoi creează un nou Vector3 cu componentele sale setate la rezultatul apelării metodei GetRealPosition cu componentele corespunzătoare ale poziției și normalei loviturii ca argumente. Se returnează versiunea întreagă rotunjită a acestui vector.
- Cea de-a doua metodă GetBlockPosition primește ca argument o poziție Vector3 și returnează un Vector3Int. Aceasta calculează poziția blocului din poziția dată.
    - Metoda începe prin scalarea poziției date cu dimensiunea inversă a voxelului din setările acestei lumi. Apoi returnează versiunea întreagă a acestei poziții scalate.
- Metoda GetRealPosition primește două argumente float, pos și normal, și returnează un float. Aceasta calculează poziția reală a unui blocului pe baza poziției sale de lovire și a normalului de lovire.
    - Metoda verifică dacă valoarea absolută a restului împărțirii lui pos la 1 este egală cu 1,5. În caz afirmativ, metoda returnează pos minus jumătate din normal. În caz contrar, se returnează pos.
- Metoda CalculateVoxelSizeInverse nu acceptă argumente și este de tip void. Aceasta calculează dimensiunea inversă a voxelului pentru setările acestei lumi.
    - Metoda stabilește valoarea proprietății inverseVoxelSize din setările acestei lumi la un nou Vector3 cu componentele sale stabilite la 1 împărțit la componentele corespunzătoare ale dimensiunii blocului acestei lumi.
- Metoda OnDisable este o metoda moștenită din clasă MonoBehaviour, care este suprascrisă în această clasă. Ea este apelată atunci când scriptul este dezactivat.
    - Metoda OnDisable apelează metoda Cancel pe câmpul taskTokenSource. Aceasta anulează toate sarcinile care utilizează tokenul de anulare din această sursă de token.

4.5 Descrierea sistemului de generare procedurală a terenului

4.5.1 Descrierea clasei NoiseSettings

Clasa NoiseSettings care moștenește ScriptableObject. Clasa NoiseSettings este decorată cu atributul CreateAssetMenu, care specifică numele fișierului și numele meniului care trebuie utilizate la crearea unui activ de acest tip.

Clasa NoiseSettings conține mai multe câmpuri publice care reprezintă diverse setări pentru generarea zgomotului.

Aceste câmpuri includ:

- noiseZoom, acesta este de tip Vector3 și reprezintă nivelul de zoom al zgomotului, aceste valori sunt pozitive. Un nivel scăzut de zoom va genera valori similare pentru o suprafată mare, iar pentru un nivel ridicat de zoom opusul este valabil. Cum nivelul de zoom este independent pe cele 3 axe, pot fi definite valori diferite de zoom pe cele 3 axe pentru a determina, de exemplu, un aspect alungit pentru a genera dunele într-un desert. Un al exemplu ar fi generarea de peșteri care pot fi mai lungi și mai late decât înalte, d.p.d.v al zoom-ului acest lucru se traduce printr-un zoom mai mare pentru axa y, și un zoom mai mic pe axele x și z.
- localOffset, acesta este de tip Vector3Int și reprezintă decalajul local al zgomotului
- worldOffset, acesta este, de asemenea, de tip Vector3Int și reprezintă decalajul global al zgomotului.
- amplitudeMultiplyer, acesta este de tip float și reprezintă multiplicatorul de amplitudine pentru zgomot. De obicei, în lucrări de specialitate această variabila este numita persistență [19].
- frequencyMultyplier, acesta este de tip float și reprezintă multiplicatorul de frecvență al zgomotului. De obicei, în lucrări de specialitate această variabila este numita lacunaritate [19].
- octaves, este de tip "uint" și reprezintă numărul de octave care trebuie utilizate la generarea zgomotului
- exponent, este de tip float și reprezintă exponentul care trebuie utilizat la generarea zgomotului, daca zgomotul este redistribuit.
- redistributionModifier, este, de tip float și reprezintă modificatorul de redistribuire pentru zgomot. Acesta este folosit pentru a ajusta marimea zgomotului, fară a modifica nivelul de zoom.
- minDimension este de tip Vector3Int și reprezintă dimensiunile minime pentru cartografierea zgomotului.
- maxDimension este, de asemenea, de tip Vector3Int și reprezintă dimensiunile maxime pentru cartografierea zgomotului.

Aceste setări sunt folosite pentru a genera:

- Harta de înălțime a terenului
- Pozițiile structurilor pe hartă
- Zgomotul folosit pentru a genera peșterile
- Zgomotul folosit pentru a genera „filoane” de blocuri în interiorul peșterilor

4.5.2 Descrierea clasei statice CustomNoise

Clasa statică CustomNoise are rolul de a genera zgomot neted și de a oferi uneltele necesare pentru a modela zgomotul pentru a ajunge la un rezultat dezirabil care poate fi folosit pentru a genera diverse aspecte ale lumii generate procedural.

Aceasta clasa poate genera zgomot neted folosind fie funcția de zgomot Perlin din clasa Mathf [20], fie folosind funcția de zgomot Simplex din clasa FastNoiseLiteStatic a cărei implementare a fost preluată din pachetul de funcții de zgomot FastNoise Lite [21], dezvoltată de către Jordan Peck [22]. Această implementare poate fi găsită pe GitHub și poate fi folosită sub licență MIT. Acest tip de licență are permisiuni pentru uz personal, distribuție, modificare și uz comercial.

Metodele clase se folosesc de câmpul useSimplex pentru a determina dacă pentru generarea de zgomot va fi folosit zgomot de tip Perlin sau dacă va fi folosit zgomot de tip Simplex.

Clasa CustomNoise conține mai multe metode statice publice:

- Noise2D
- Noise3D
- FractalNoise2D
- FractalNoise3D
- ResistributeNoise
- MapFloatValue
- MapIntValue
- MapNormalizedValue
- MapNormalizedValueToInt
- Redistribution

Metodele au următoarele definiții:

- Metoda Noise2D primește ca argumente două valori de tip float, x si z, care reprezintă coordonate. Această metodă întoarce o valoare aflată în intervalul [1, 1] folosind fie funcția Mathf.PerlinNoise dacă valoarea , fie funcția FastNoiseLiteStatic.GetNoise, folosind cele două coordonate x și z.
    - Funcția Mathf.PerlinNoise întoarce o valoare aflată în intervalul [1, 1], cu amendmentul că pot fi întoarse valori cu o eroare neglijabila la capetele intervalului, deci se pot întoarce valori de putin mai mici decât 1, sau puțin mai mari decât 1. Acest aspect poate fi ameliorat folosind funcția Clamp din clasa Mathf pentru a limita valorile în intervalul [1, 1].
    - Funcția FastNoiseLiteStatic.GetNoise întoarce o valoare aflată în intervalul [-1, 1]. Pentru a aduce valorile în intervalul dorit de [1, 1] este folosită metoda

MapFloatValue, care mapează o valoare dată dintr-un interval dat, în alt interval dat.

- Metoda Noise3D este similară cu metoda Noise2D, doar că primește ca argumente trei valori de tip float, x, y si z, care reprezintă coordonate. Această metodă întoarce o valoare aflată în intervalul [1, 1] folosind fie funcția Mathf.PerlinNoise dacă valoarea , fie funcția FastNoiseLiteStatic.GetNoise, folosind cele două coordonate x, y și z
    - Cum clasa Mathf nu are o funcție care poate genera zgomot neted pentru trei valori, este folosită următoarea metodă
        - Sunt calculate șase valori de zgomot pentru toate permutările posibile pentru cele trei coordonate luate două câte două, folosind funcția Mathf.PerlinNoise
        - Rezultatul va fi media celor șase valori
        - Funcția care este costisitoare d.p.d.v. al complexității, deoarece funcția apelează de șase ori funcția Mathf.PerlinNoise
    - Funcția FastNoiseLiteStatic.GetNoise poate genera direct o valoare de zgomot neted pentru trei coordonate. Aceasta întoarce o valoare aflată în intervalul [-1, 1]. Pentru a aduce valorile în intervalul dorit de [1, 1] este folosită metoda MapFloatValue, care mapează o valoare dată dintr-un interval dat, în alt interval dat.

- Metoda FractalNoise2D primește ca parametrii o poziție 2D Vector2Int și un obiect de tip NoiseSettings. Aceasta calculează o valoare folosind tehnici de generare a zgomotului roz pentru o poziție 2D (pos) folosind setările furnizate în obiectul de tip NoiseSettings. Obiectul "settings" conține diverse proprietăți care controlează generarea zgomotului, cum ar fi nivelul de zoom, numărul de octave și multiplicatorii de amplitudine și frecvență. Zgomotul roz este generat prin combinarea mai multor straturi (sau octave) de zgomot la diferite frecvențe și amplitudini. Rezultatul final este un model de zgomot mai complex și mai natural decât cel care ar putea fi obținut cu un singur strat de zgomot.

    - Metoda calculează mai întâi coordonatele x și z prin înmulțirea componentelor x și y ale vectorului pos cu componentele corespunzătoare ale proprietății noiseZoom din obiectul "settings". Coordonatele x și z sunt apoi incrementate cu componentele corespunzătoare ale proprietății noiseZoom. Această adunare este necesară în cazul în care pentru generarea zgomotului este folosit zgomot de tip Perlin pentru a evita unele probleme care apar la generarea de zgomot Perlin dacă sunt folosite valori întregi.
    - Valorile de deplasament global și local sunt folosite pentru a genera valori diferite de zgomot pentru aceleași poziții
    - Frecvența este inițializată cu 1
    - Amplitudinea este inițializată cu 1

- Metoda apoi iterează peste intervalul întreg de la 1 la numărul de octave din obiectul NoiseSettings. La fiecare iterație, sunt calculate coordonatele pentru care se generează zgomot neted folosind ecuația (1). Valorile pentru deplasamentul local și global sunt extrase din obiectul NoiseSettings.

$$noiseCoord = (coord + local\_offset + global\_offset) * frecvență \quad (1)$$

- Folosind coordonatele calculate anterior se generează zgomot de tip neted folosind funcția Noise2D. Această valoare este adăugată la rezultatul final folosind ecuația (2)

$$finalResult \mathrel{+}= amplitude * noise\_result \quad (2)$$

- Valorile pentru amplitudine și frecvență sunt modificate folosind multiplicatorii de amplitudine, respectiv frecvență prin înmulțire.
- Deoarece rezultatul final este o medie ponderată, este reținută suma tuturor amplitudinilor
- După ce toate iterațiile sunt finalizate, metoda returnează rezultatul final împărțit la amplitudinea totală.

- Metoda FractalNoise3D primește ca parametrii o poziție 3D Vector3Int și un obiect de tip NoiseSettings. Această metodă este similară cu metoda FractalNoise2D, doar că este generat zgomot neted pentru o poziție tridimensionala. Toate operațiile aplicate coordonatelor x și z se vor aplica și coordonatei y.

- Metoda Redistribution primește două argumente: un float numit value și un obiect NoiseSettings numit settings. Metoda returnează o valoare flotantă. Metoda calculează o nouă valoare prin aplicarea unei funcții de redistribuire la valoarea de intrare folosind setările furnizate. Obiectul settings conține diverse proprietăți care controlează redistribuirea zgomotului, cum ar fi modificatorul de redistribuire și exponentul.
  - Metoda înmulțește mai întâi valoarea zgomotului de intrare cu proprietatea redistributionModifier a obiectului settings. Rezultatul este apoi ridicat la puterea proprietății exponent a obiectului settings folosind funcția Mathf.Pow încorporată în Unity. Rezultatul final este apoi returnat.
  - Aceasta poate fi utilizată pentru a modifica distribuția valorilor de zgomot pentru a obține efectul dorit fără a modifica alte setări de zgomot.

- Metoda ResistributeNoise primește două argumente: un float numit noise și un obiect NoiseSettings numit settings. Metoda returnează o valoare flotantă. Metoda

redistribuie valoarea zgomotului de intrare utilizând setările furnizate. Obiectul settings conține diverse proprietăți care controlează redistribuirea zgomotului.

- Metoda apelează mai întâi o altă metodă numită Redistribution, având ca argumente valoarea zgomotului de intrare și obiectul settings. Rezultatul acestui apel este stocat înapoi în variabila zgomot.
- Cum redistribuirea de mai sus poate avea ca rezultat o valoare care nu se află în intervalul [1, 1], trebuie sa redistribuim valorile 1, respectiv 1 cu aceleași setări pentru a obține noile capete ale intervalului în care se află valoare redistribuită. Astfel, se inițializează două variabile, minRedistribution și maxRedistribution, pentru a stoca valorile minime și maxime posibile ale zgomotului redistribuit. Valoarea minimă este setată la 1, în timp ce valoarea maximă este calculată prin apelarea metodei Redistribution cu o valoare de intrare de 1 și cu obiectul settings ca argumente.
- Metoda apelează apoi o altă metodă numită MapFloatValue, având ca argumente valoarea zgomotului redistribuit, valorile minimă și maximă de redistribuire și intervalul de ieșire dorit de la 1 la 1. Rezultatul acestei operații este returnat.

- Clasa implementează, de asemenea, diferite funcții de mapare, care mapează o valoare dată aflată într-un interval dat, în alt interval dat. Această mapare este efectuată folosind ecuația (3)

$$result = outMinRange + \frac{(input - inMinRange) * (outMaxRange - outMinRange)}{(inMaxRange - inMinRange)} \quad (3)$$

4.5.3 Descrierea clasei abstracte VoxelLayerHandler

Clasa VoxelLayerHandler moștenește din clasa MonoBehaviour furnizată de motorul de joc Unity. Clasa este destinată a fi utilizată ca o componentă atașată unui obiect de joc într-o scenă Unity. Această clasă implementează o versiune modificată a design pattern-ul chain of responsability. Acest design pattern, folosit de obicei in aplicații server-client, este folosit pentru a permite modelarea unui sistem de generare procedurală ușor de modificat, fiecare strat fiind responsabil pentru o parte relativ mică din logica de generare procedurală prin care se determina tipul unui bloc de la o poziție dată.

Acestă clasă de bază abstractă este folosită pentru implementarea gestionarilor de straturi blocuri într-un joc Unity. O clasă care moștenește VoxelLayerHandler este responsabilă pentru modificarea datelor de tip VoxelType dintr-un chunk în funcție de anumite condiții prin implementarea metodei abstracte TryHandling.

Clasa conține mai multe câmpuri serializate care pot fi setate în inspectorul Unity. Acestea includ o referință la un alt obiect VoxelLayerHandler numit Next și două liste de obiecte VoxelType numite canReplace și cannotReplace.

Clasa conține, de asemenea, două câmpuri neserializate de tip HashSet<VoxelType> denumite canReplaceHashSet și cannotReplaceHashSet. Aceste câmpuri sunt utilizate pentru a stoca conținutul listelor canReplace și cannotReplace ca seturi hash pentru o căutare mai rapidă.

Clasa suprascrie metoda Awake furnizată de clasa MonoBehaviour. În această metodă, conținutul listelor canReplace și cannotReplace este copiat în seturile hash corespunzătoare.

Metoda Handle care primește mai multe argumente: un obiect ChunkData numit data, un Vector3Int numit pos, un intreg numit surfaceHeight și o referință la un VoxelType numit currentVoxel. Metoda returnează o valoare bool.

- Metoda apelează mai întâi o metodă abstractă numită TryHandling cu aceleași argumente. Dacă această metodă returnează true, atunci și metoda Handlehandle returnează true.
- În cazul în care apelul la TryHandling returnează false și câmpul Next nu este nul, atunci metoda apelează metoda Handle pe obiectul la care face referire câmpul Next cu aceleași argumente. Rezultatul acestui apel este apoi returnat.
- În cazul în care ambele condiții sunt false, metoda returnează "false".

Clasa definește, de asemenea, o metodă abstractă numită TryHandling care primește aceleași argumente ca și metoda Handle și returnează o valoare bool. Această metodă trebuie să fie implementată de orice subclasă concretă a clasei VoxelLayerHandler.

În cele din urmă, clasa suprascrie metoda OnValidate, folosită doar atunci când codul este rulat în editorul Unity, furnizată de clasa MonoBehaviour. În această metodă, conținutul listelor canReplace și cannotReplace este copiat în seturile hash corespunzătoare.

4.5.4 Subclasele concrete ale clasei abstracte VoxelLayerHandler

Subclasele concrete ale clasei abstracte VoxelLayerHandler sunt:

- BedrockLayerHandler, care amplasează un strat de tipul de voxel Bedrock în partea inferioară a lumii astfel încât jucătorul să nu cada prin lume acolo unde există goluri de aer.
- WaterLayerHandler care amplasează un bloc de VoxelType Water dacă poziția globală y se află între nivelul de apă aflat în setările de generare a lumii și înălțimea hărții pentru coloana x, z plus un deplasament. Dacă poziția globală y se află între

deplasament și înălțimea hărții pentru coloana x, z, atunci este amplasat un bloc cu VoxelType-ul Sand.

- AirLayer care amplasează VoxelType-ul de Air pentru acele blocuri care au poziția globală y mai mare decât înălțimea hărții pentru coloana x, z
- SurfaceLayer care a amplasează VoxelType-ul SurfaceVoxelType (setat în editor) pentru acele blocuri care au poziția globală y egală cu înălțimea hărții pentru coloana x, z
- UndergroundLayer care amplasează VoxelType-ul nearGroundVoxelType (setat în editor) pentru acele blocuri care au poziția globală y se află între înălțimea hărții pentru coloana x, z și un deplasament setat în editor, dacă poziția y se află sub acel deplasament se amplasează VoxelType-ul underGroundVoxelType (setat în editor).
- CustomVoxelLayer care se folosește de o valoare de zgomot neted tridimensională și o valoare de threshold pentru a amplasa un bloc de VoxelType setat în editor. Această clasă are posibilitatea de a inversa acel thershold pentru a avea o flexibilitate mai mare în modul în care se generează lumea. Acest VoxelLayer este folosit pentru a genera peșteri și filoane de blocuri. Folosind această implementare, o peșteră este doar un filon de blocuri pentru care VoxelType-ul setat în editor este de tip Air.

4.5.5 Descrierea clasei StructureSO

Clasa StructureSO reține o listă de StructureVoxel. Aceasta este decorată cu atributul CreateAssetMenu, care specifică numele fișierului și numele meniului care trebuie utilizate la crearea unui activ de acest tip. Această clasă reține toate blocurile care determină o structură, precum și pozițiile lor.

Clasa StructureVoxel are două câmpuri, unul care reține o poziție de tip Vector3Int, iar celălalt care reține un VoxelType. Această clasă este folosită pentru a determina poziția la care trebuie amplasat un bloc într-o structură.

4.5.6 Descrierea clasei abstracte StructureLayerHandler

Clasa StructureLayerHandler moștenește din clasa MonoBehaviour furnizată de motorul de joc Unity. Clasa este destinată a fi utilizată ca o componentă atașată unui obiect de joc într-o scenă Unity. Această clasă este implementată în același fel ca și clasa VoxelLayerHandler, doar că nu verifică ce blocuri poate înlocui, sau ce blocuri nu poate înlocui, iar poziția dată reprezintă poziția unde va fi amplasat primul bloc al structurii.

4.5.7 Subclasele concrete ale clasei abstracte StructureLayerHandler

- TreeLayerHandler, care amplasează o structură de tip copac reținută în câmpul de tip StructureSO la înălțimea hărții pentru coloana x, z dată. Structura poate fi amplasată doar pe blocuri de tip Grass_Dirt. Înainte ca structura sa fie amplasată, aceasta este rotită folosind un unghi ales aleator din lista angleChangeList a clasei

TreeLayerHandler. Structura este amplasată iterând lista de blocuri din structura reținută în câmpul de tip StructureSO. Pentru fiecare iterație este amplasat blocul curent din listă folosind metoda SetVoxelFromChunkCoordinates a clasei Chunk.

- CactusLayerHandler este asemănătoare cu clasa TreeLayerHandler, doar că este amplasată o structură de tip cactus care poate fi amplasată doar pe blocuri de tip Sand. De asemenea, coloana din mijloc a structurii de tip cactus poate varia în înălțime de la cactus la cactus.
- CoralLayerHandler este asemănătoare cu clasa TreeLayerHandler, doar că este amplasată o structură de tip coral care poate fi amplasată doar pe blocuri de tip Sand și dacă blocurile amplasate se află la o anumită adâncime setată în editor. Astfel putem avea mai multe tipuri diferite de coral, dar care se pot afla doar de la anumite adâncimi în jos diferite.

4.5.8 Descrierea clasei TerrainGenerator

Clasa TerrainGenerator moștenește din MonoBehaviour. Clasa TerrainGenerator conține un câmp public numit biomeGenerator de tip BiomeGenerator. De asemenea, definește o metodă numită GenerateChunkData care primește ca argument un obiect ChunkData și returnează un obiect ChunkData.

Metoda GenerateChunkData generează date de tip chunk pentru obiectul de date de tip chunk dat. Aceasta începe prin a calcula numărul de blocuri din chunk prin înmulțirea componentelor dimensiunii chunk-ului. Apoi apelează metoda parcurge toate pozițiile din acel chunk.

- Pentru fiecare indice de bloc, metoda calculează coordonatele x, y și z ale blocului folosind operații de divizare și modulo de numere întregi împreună cu dimensiunile chunk-ului pe cele trei axe. Aceste operații sunt descrise în ecuațiile (4)-(6). Apoi declară o variabilă locală pentru a reține poziția la sol și apelează metoda Get2DTerrainY pe obiectul BiomeGenerator cu coordonatele x și z și cu datele de fragmentare ca argumente. Se atribuie valoarea returnată variabilei "poziția la sol".

$$x = index \,\%\, chunkSize.x \quad (4)$$

$$y = (index \,/\, chunkSize.x) \,\%\, chunkSize.y \quad (5)$$

$$z = index \,/\, (chunkSize.x * chunkSize.y) \quad (6)$$

- Metoda apelează apoi metoda ProcessVoxel pe obiectul BiomeGenerator, având ca argumente datele chunk, un nou Vector3Int cu componentele sale setate la coordonatele blocului și poziția la sol. Aceasta atribuie datele de tip chunk returnate variabilei data.

- După ce toate iterațiile sunt finalizate, metoda apelează metoda ProcessStructures pe obiectul BiomeGenerator având ca argument datele chunk. Aceasta atribuie datele de tip "chunk" returnate variabilei "data". În cele din urmă, metoda returnează aceste date actualizate.

4.5.9 Descrierea clasei BiomeGenerator

Clasă BiomeGenerator moștenește din MonoBehaviour.

Clasa BiomeGenerator conține următoarele câmpuri publice:

- biomeNoiseSettings, care este de tipul "NoiseSettings" și reprezintă setările de zgomot utilizate pentru a genera biomul
- firstLayer, este de tip VoxelLayerHandler și reprezintă primul gestionar de strat utilizat pentru a determina tipul blocului aflat la o poziție în lume.
- additionalLayerHandlres, este de tip List<VoxelLayerHandler> și reprezintă straturile suplimentare utilizate pentru a determina tipul blocului aflat la o poziție în lume.
- structureGenerators, este de tip List<StructureGenerator> și reprezintă generatorii de structuri utilizați pentru a genera structuri în biom.

De asemenea, sunt definite urmatoarele metode: metode:

- ProcessVoxel,
- ProcessStructures
- Get2DTerrainY.

Metoda ProcessVoxel care primește ca argumente un obiect ChunkData, o poziție Vector3Int și o poziție la sol int. Aceasta returnează un obiect ChunkData. Aceasta procesează un voxel la poziția dată în datele Chunk date.

- Metoda începe prin setarea valorii proprietății world offset din setările de zgomot din biome la proprietatea map seed offset din setările acestei lumi. Setează valoarea proprietății "noise settings" a datelor de tip "chunk data" la valoarea câmpului "biome noise settings". Apoi, se obține voxelul din poziția dată în datele chunk date, utilizând metoda GetVoxelFromChunkCoordinates a clasei Chunk.

- Apoi, metoda apelează metoda Handle din cadrul primului operator de strat, având ca argumente datele chunk date, poziția, poziția la sol și voxelul curent. Apoi trece prin toate straturile suplimentare din lista de gestionari de straturi suplimentare. Pentru fiecare gestionar de strat, se apelează metoda Handle cu aceleași argumente.

- După ce toate iterațiile sunt finalizate, se returnează datele actualizate ale chunk-ului.

Metoda ProcessStructures primește ca argument un obiect ChunkData și returnează un obiect ChunkData. Aceasta procesează structurile din datele chunk date.

- Metoda începe prin iterația peste toți generatorii de structuri din lista de generatori de structuri. Pentru fiecare generator de structuri, aceasta apelează metoda GenerateStructureData cu datele de tip chunk date ca argument. Aceasta atribuie lista returnată de poziții locale unei variabile locale.
- Apoi, metoda parcurge această listă de poziții locale. Pentru fiecare poziție locală, aceasta apelează metoda Handle a primului gestionar de strat de structură al acelui generator de structuri, având ca argumente datele chunk-ului, poziția locală și acest generator de biome.
- După ce toate iterațiile sunt finalizate pentru acel generator de structură, se trece peste toate straturile de structură suplimentare din lista sa de gestionari de straturi de structură suplimentare. Pentru fiecare gestionar de strat de structură suplimentar, se trece din nou peste toate pozițiile locale. Pentru fiecare poziție locală, se apelează metoda Handle cu aceleași argumente.
- După ce toate iterațiile sunt finalizate pentru toți generatorii de structuri, se returnează datele actualizate ale bucăților.

Metoda Get2DTerrainY primește două argumente int x și z și un obiect ChunkData ca argumente. Aceasta returnează un int care reprezintă coordonatele y ale terenului în poziția x-z din datele de tip chunk.

- Metoda începe prin a calcula un index în matricea de hartă a înălțimii din datele acelui fragment prin înmulțirea lui x cu dimensiunea sa z și adăugarea lui z. Verifică dacă acest element din matricea de hartă a înălțimii nu este egal cu 1. Dacă nu este egal cu 1, returnează valoarea sa.
- Metoda calculează apoi coordonatele mondiale x și z prin adăugarea x și z la componentele lor corespunzătoare din poziția mondială a datelor din acel chunk. Se declară o variabilă locală pentru a păstra zgomotul terenului.
- Se apelează metoda OctaveNoise2D din clasa statică CustomNoise cu un nou Vector2Int cu componentele sale setate la worldPosX și, respectiv, worldPosZ și biomeNoiseSettings ca argumente. Se atribuie valoarea returnată la terrainNoise.
- În urma apelului la generatorul de zgomot neted se va obține o valoare normalizată. Acea valoare este apoi mapată la intervalul [biomeNoiseSettings.minDimension.y, biomeNoiseSettings.maxDimension.y] pentru a obține o care poate fi folosită drept înălțimea coloanei de blocuri cu pozițiile x, z date. Această valoare este apoi returnată.

4.5.10 Descrierea clasei DataProcessing

Clasa DataProcessing are rolul de a determina maximele locale dintr-o matrice 2D dată.

În acest scop sunt implementate urmatoarele metode:

- Metoda CheckNeighbours primește ca argumente o matrice bidimensională, un Vector2Int care reprezintă o poziție și o funcție delegat de tip Func<float, bool> numit succesCondition. Această metoda întoarce o valoare de tip bool. Acească metodă verifică dacă funcția delegată este adevărată pentru toți vecinii poziției date.
    - Metoda iterează peste o listă de directii cardinale (Nord, Nord-Est, Est, Sud-Est, Sud, Sud-Vest, Vest, Nord-Vest), aceste direcții sunt reprezentate de vectori normalizați de tip Vector2Int
    - Într-o iterație într-o variabila locală se stochează suma dinte poziția dată și direcția curentă. Daca noua poziție se află la una din marginile matricii date, atunci se continuă la următoarea iterație. Dacă noua poziție nu se află la una din marginile matricii date, atunci se verifică dacă rezultatul funcții delegat, în cazul în care rezultatul este fals funcția returnează fals.
    - După ce se termină toate iterațiile se returnează true

- Metoda FindLocalMaxima2D primește ca argumente o matrice bidimensională și întoarce o listă de Vector3Int pentru care se determină doar coordonatele x si z ale fiecărui element din listă. Această metodă are rolul de a determina maximele locale dintr-o matrice bidimensionala. Maximele locale sunt definite ca acele puncte cu valori strict mai mari decât valorile vecinilor din pozițiile din Nord, Nord-Est, Est, Sud-Est, Sud, Sud-Vest, Vest, Nord-Vest.
    - Metoda iterează peste toate valorile din matricea dată și verifică folosind metoda CheckNeighbours dacă un punct este maxim local. În caz pozitiv, această poziție este adăugată listei de maxime locale.
    - În final, lista este returnată

4.5.11 Descrierea clasei abstracte StrucutreGenerator

Clasa abstractă StructureGenerator moștenește din clasa MonoBehaviour furnizată de motorul de joc Unity. Clasa este destinată a fi utilizată ca o componentă atașată unui obiect de joc într-o scenă Unity. Un generator de structuri este responsabil pentru generarea de date pentru structurile dintr-un chunk pe baza anumitor condiții. Comportamentul exact al fiecărui generator este determinat de implementarea sa a metodei abstracte GenerateStructureData.

Clasa conține două câmpuri publice: o referință la un obiect StructreLayerHandler numit firstStructureLayerHandler și o listă de obiecte StructreLayerHandler numită additionalStructureLayerHandler.

Clasa definește, de asemenea, o metodă abstractă numită GenerateStructureData care primește un singur argument: un obiect ChunkData numit chunkData. Metoda returnează o listă de obiecte Vector3Int. Această metodă trebuie să fie implementată de orice subclasă concretă a clasei StructureGenerator. Comportamentul exact al metodei este determinat de implementarea acesteia în subclasa respectivă.

4.5.12 Subclasele concrete ale clasei abstracte StrucutreGenerator

- TreeGenerator
- CactusGenerator
- CoralGenerator

Toate subclasele implementează o metoda numită GenerateStructureNoise2D care primește ca argumente un obiect de tip ChunkData și un obiect de tip NoiseSettings. Această funcție întoarce o matrice bidimensională ce conține valori de zgomot generate folosind funcția FractalNoise2D din clasa CustomNoise.

- Metoda iterează peste toate pozițiile de pe axele x și z ale obiectului ChunkData.
- Pentru fiecare poziție se calculează valoarea zgomotului și se stochează în matricea valorilor de zgomot la indecșii aferenți poziției

De asemenea, toate subclasele implementează metoda GenerateStructureData definită în clasa abstractă StrucutreGenerator. Implementarea acestei funcții este identică pentru toate clasele dupa cum urmează:

- Este calculată matricea cu valorile de zgomot folosind metoda GenerateStructureNoise2D
- Este returnată o listă de poziții ale structurilor folosind funcția FindLocalMaxima2D din clasa DataProcessing

Deoarece zgomotul este generat folosind funcția FractalNoise2D din clasa CustomNoise avem un grad foarte ridicat de control asupra pozițiilor unde se vor genera structuri. Pentru a controla modelul zgomotului trebuie modificate setările folosite pentru a genera zgomotul. Astfel, de exemplu, pentru a modifica densitatea structurilor putem să modificam setarea de zoom pentru generarea zgomotului.

4.6 Descrierea implementării managerilor de joc

Pentru a gestiona multiplele aspecte de joc, sunt folosiți mai mulți manageri de joc. Aceștia controlează logica de:

- Generare a lumii
- Generare a meșei de navigare a inamicilor
- Spawnare a jucătorului
- Spawnare a inamicilor
- Generare a bordurilor din spațiul de joc
- Tranziție între starea normală de joc și starea de pauză a jocului

4.6.1 Descrierea implementării logicii de generare a lumii

Clasa GameManager moștenește clasa MonoBehaviour furnizată de motorul de joc Unity. Clasa este destinată a fi utilizată ca o componentă atașată unui obiect de joc într-o scenă Unity. Această clasă are rolul de a controla camera la începutul jocului, pană când aceasta va fi controlată de jucător și de a initializa ceilalți manageri cu datele relevante.

Metoda Awake pune camera într-o poziție aflată deasupra harții astfel încât generarea lumii să poată fi vizualizată, de asemenea modul camerei este setat la modul Orthographic. De asemenea toți managerii sunt stocați pentru a evita apeluri repetate către funcția GetComponentInChildren care este costisitoare d.p.d.v. al timpului de execuție.

Metoda SetUpManagers are rolul de a inițializa ceilalți manageri cu valorile potrivite după ce lumea a fost generată. Această funcție este apelată atunci când este invocat evenimentul OnWorldCreated din clasa World.

4.6.2 Descrierea implementării logicii de generare a meșei de navigare a inamicilor

Clasa GameManager moștenește clasa MonoBehaviour furnizată de motorul de joc Unity. Clasa este destinată a fi utilizată ca o componentă atașată unui obiect de joc într-o scenă Unity. Această clasă are rolul a genera meșa de navigare a inamicilor pe chunk-urile generate procedural.

Clasa are următoarele câmpuri:

- Un obiect GameObject denumit RenderedChunks, care este parintele tuturor chunk-urilor generate procedural, acest obiect trebuie sa aiba atașat componenta NavMeshSurface din pachetul Unity.AI.Navigation.
- Un obiect de tip NavMeshSurface denumit surface care este o referință către componenta NavMeshSurface a obiectului RenderedChunks.

Această componentă este folosită pentru a genera navmesh-ul pentru inamici

Această componentă are următoarele setări relevante:

- Agent Type: este setat la tipul Enemy
- Use Geometry: este setat la Render Meshes
- Collect Objects: setat la Current Object Hierearchy
- Include Layers: setat la Ground
- Buld HeightMesh: setat la true

Metoda SetupNavMeshManager primește ca argument un GameObject care va fi atribuit câmpului RenderedChunks, apoi surface va primi o referință către componenta NavMeshSurface a obiectului RenderedChunks folosind metoda GetComponent din clasa GameObject. După ce au fost atribuite valori pentru cele două câmpuri este apelată metoda CreateNavMesh.

Metoda CreateNavMesh are rolul de a genera meșa de navigare a inamicilor. Pentru a îndeplini această funcționalitate este apelată metoda BuildNavMesh a câmpului surface.

5 EVALUAREA REZULTATELOR

În capitolul următor, rezultatele acestui proiect vor fi evaluate pe baza a două criterii. În primul rând, performanța sistemului de generare a lumii va fi evaluată prin analiza eficienței componentelor sale. În al doilea rând, feedback-ul jucătorilor va fi examinat pentru a determina reacțiile acestora la diferite scene, preferințele lor în ceea ce privește aspectele grafice și generative ale jocului și intuitivitatea comenzilor, printre alți factori.

5.1 Analiza sistemului de generare a lumii

Pentru analiza sistemului de generare a lumii se vor urmări următoarele aspecte pentru fiecare scenă în parte:

- Timpul necesar pentru a genera datele blocurilor pentru toate chunk
- Timpul necesar pentru a genera datele legate de modelele 3D pentru toate chunk-urile
- Timpul necesar pentru a desena pe ecran toate modelele 3D pentru toate chunk-urile
- Timpul necesar pentru a genera meșă de navigare folosită de inamici

Fiecare scenă a fost generată cu seed-ul „UPB” de 10 ori, iar rezultatele intermediare au fost reținute manual. Setările de configurare a lumii for fi diferite de la scenă la scenă. Fiecare rezultat final în milisecunde reprezintă partea întreagă a mediei aritmetice a rezultatelor intermediare.

Pentru fiecare scenă, vor fi prezentate setările semnificative ale sistemului de generare a lumii care pot afecta substanțial performanța sistemului de generare a lumii.

5.1.1 Scena de câmpie

Pentru scena de câmpie, setările relevante care pot afecta în mod semnificativ performanța de timp a sistemului de generare sunt:

- Dimensiunea chunk-urilor. Pentru analiza efectuată au fost folosite următoarele dimensiuni: 16 unitați pentru axa x, 64 de unitați pentru axa y, 16 unitați pentru axa z. Astfel, fiecare chunk are o dimensiune de 16384 blocuri.
- Câte chunk-uri se desenează în jurul jucătorului. Pentru analiza efectuată se va genera un cub de dimensiune 25 x 1 x 25 de chunk-uri. Astfel sunt generate 625 de chunk-uri.
- Înmulțind cele două numere finale calculate anterior, obținem că este necesara procesarea a 10.240.000 de blocuri pentru a genera lumea

De asemenea, este important de menționat faptul că pentru această scenă este generat atât zgomot de tip bidimensional, cât și zgomot de tip tridimensional, iar o singură structură este generată.

Rezultatele pot fi observate în tabelul 1:

Tabel 1

Criteriu	**Timp (în milisecunde)**	**Timp (în secunde)**
Timp gen. blocuri	11749	11,749
Timp gen. date model 3D	878	1,878
Timp desenare	4114	4,114
Timp generare meșa navigare	4668	4,668
Total	21409	21,409

După cum se poate observa în tabelul 1, generarea datelor legate de blocuri durează cel mai mult, fiind necesar un număr ridicat de calcule pentru a genera zgomotul folosit pentru a genera lumea.

Deși pentru a genera datele pentru modelul 3D este necesară parcurgerea a șase fețe pentru toate cele 10.240.000 de blocuri, timpul asociat este unul neglijabil. Acest timp scăzut este cauzat de doi factori:

1. pentru marea majoritate a celor 10.240.000 de blocuri nu se va genera nici o față deoarece se află înconjurate de alte blocuri de același tip
2. pentru calcularea fețelor sunt utilizate calcule de o complexitate redusă

Timpul de desenare pare ridicat, dar trebuie reținut faptul că desenarea chunk-urilor este în mod voit încetinită pentru a nu desena mai mult de un chunk într-un singur cadru. Dacă această restricție ar fi eliminată, desenarea chunk-urilor s-ar produce mai rapid, dar ar exista un cost mai mare asociat cadrului în care se produc calculările necesare pentru desenarea chunk-urilor.

Timpul de generare a meșei de navigare este al doilea cel mai mare timp asociat sistemului de generare a lumii, acest timp ridicat este cauzat de faptul că trebuie sa genereze o meșă de navigare pentru toate suprafețele valide din chunk-urile desenate.

5.1.2 Scena de deșert

Pentru scena de deșert, setările relevante care pot afecta în mod semnificativ performanța de timp a sistemului de generare sunt:

- Dimensiunea chunk-urilor. Pentru analiza efectuată au fost folosite următoarele dimensiuni: 16 unitați pentru axa x, 64 de unitați pentru axa y, 16 unitați pentru axa z. Astfel, fiecare chunk are o dimensiune de 16384 blocuri.
- Câte chunk-uri se desenează în jurul jucătorului. Pentru analiza efectuată se va genera un cub de dimensiune 17 x 1 x 17 de chunk-uri. Astfel sunt generate 289 de chunk-uri.
- Înmulțind cele două numere finale calculate anterior, obținem că este necesara procesarea a 4.734.976 de blocuri pentru a genera lumea

De asemenea, este important de menționat faptul că pentru această scenă este generat doar zgomot de tip bidimensional și o singură structură este generată.

Rezultatele pot fi observate în tabelul 2:

Tabel 2

Criteriu	**Timp (în milisecunde)**	**Timp (în secunde)**
Timp gen. blocuri	5184	5,184
Timp gen. date model 3D	251	1,251
Timp desenare	1825	1,825
Timp generare meșa navigare	3439	3,439
Total	10699	10699

Aceleași observații făcute pentru scena de câmpie sunt valabile și pentru scena de deșert.

Se poate observa cum dimensiunea de desenare afectează timpul de executare al sistemului de generare a lumii, timpul total fiind aproximativ înjumătățit fată de scena anterioară.

5.1.3 Scena de insule

Pentru scena de insule, setările relevante care pot afecta în mod semnificativ performanța de timp a sistemului de generare sunt:

- Dimensiunea chunk-urilor. Pentru analiza efectuată au fost folosite următoarele dimensiuni: 16 unitați pentru axa x, 64 de unitați pentru axa y, 16 unitați pentru axa z. Astfel, fiecare chunk are o dimensiune de 16384 blocuri.
- Câte chunk-uri se desenează în jurul jucătorului. Pentru analiza efectuată se va genera un cub de dimensiune 25 x 1 x 25 de chunk-uri. Astfel sunt generate 625 de chunk-uri.
- Înmulțind cele două numere finale calculate anterior, obținem că este necesara procesarea a 10.240.000 de blocuri pentru a genera lumea
- Nivelul de apă care este setat să se genereze de la coordonata y = 25 în jos.

De asemenea, este important de menționat faptul că pentru această scenă este generat atât zgomot de tip bidimensional, cât și zgomot de tip tridimensional. În această scenă sunt generate patru tipuri diferite de structuri.

Rezultatele pot fi observate în tabelul 3:

Tabel 3

Criteriu	Timp (în milisecunde)	Timp (în secunde)
Timp gen. blocuri	11302	11,302
Timp gen. date model 3D	1256	1,256
Timp desenare	4240	4,240
Timp generare meșa navigare	1654	1,654
Total	18452	18,452

În această scenă rezultatele nu sunt similare cu datele analizate anterior, deși setările sunt similare. Această diferență este cauzată de introducerea unui nivel de apă în scenă.

Introducerea unui nivel de apă în scenă are următoarele efecte:

1. Se introduce un număr mai mare de verificări și de calcule necesare pentru generarea datelor pentru modelul 3D, acest lucru se poate observa în Tabelul 3, unde timpul de generare a modelului 3D este al doilea cel mai mare timp și este mai mare decât timpul analizat în scena de câmpie cu 35,42%.
2. Deoarece există mai puține suprafețe valide pe care se poate genera meșa de navigare (aceasta se poate genera doar pe geometria de tip solid), timpul de generare a acestui sistem este mai scăzut decât toate rezultatele din scenele anterioare pentru timpul de generare a meșei de navigare.

5.1.4 Scena de peșteri

Pentru scena de peșteri, setările relevante care pot afecta în mod semnificativ sunt performanța de timp a sistemului de generare sunt:

- Dimensiunea chunk-urilor. Pentru analiza efectuată au fost folosite următoarele dimensiuni: 16 unitați pentru axa x, 64 de unitați pentru axa y, 16 unitați pentru axa z. Astfel, fiecare chunk are o dimensiune de 16384 blocuri.
- Câte chunk-uri se desenează în jurul jucătorului. Pentru analiza efectuată se va genera un cub de dimensiune 17 x 3 x 17 de chunk-uri. Astfel sunt generate 867 de chunk-uri.
- Înmulțind cele două numere finale calculate anterior, obținem că este necesara procesarea a 14,204,928 de blocuri pentru a genera lumea

De asemenea, este important de menționat faptul că pentru această scenă este generat atât zgomot de tip bidimensional, cât și zgomot de tip tridimensional, iar în această scenă este generat un singur tip de structură.

Rezultatele pot fi observate în tabelul 4:

Tabel 4

Criteriu	Timp (în milisecunde)	Timp (în secunde)
Timp gen. blocuri	13460	13,460
Timp gen. date model 3D	1930	1,930
Timp desenare	8014	8,014
Timp generare mește navigare	7948	7,948
Total	31352	31,352

După cum se poate observa în tabelul 4, generarea datelor legate de blocuri durează cel mai mult, fiind necesar un număr ridicat de calcule pentru a genera zgomotul folosit pentru a genera lumea. Acest timp este mai ridicat decât cel calculat pentru scena de peșteri cu 13,57%.

Timpul pentru generarea de datele pentru modelul 3D este cu aprox. 75% mai mare decât timpul pentru generarea de datele pentru modelul 3D din scena de câmpie, deși numărul de blocuri este doar cu 32.44% mai mare decât numărul de blocuri din scena de câmpie. Această diferență de timp este cauzată de faptul că, introducând peșteri în mediul de joc, este mult mai probabil ca pentru un bloc să fie necesară calcularea mai multor fețe.

Timpul de generare a meșei de navigare este, de asemenea, disproporțional mai mare decât timpul de generare a meșei de navigare din scena de câmpie. Din nou, această diferență este cauză de peșterile generate în această scenă, deoarece există mai multe suprafețe pentru care să se genereze meșa de navigare.

5.2 Analiza feedback-ului oferit de către jucători

Pentru a evalua diferite aspecte ale jocului, a fost elaborat un formular de feedback, care a fost distribuit unui eșantion de persoane împreună cu jocul și a primit zece răspunsuri. După terminarea jocului, participanților anonimi li s-a cerut să completeze formularul răspunzând la 15 întrebări legate de calitatea sistemului de generare procedurală, de controlul jucătorului, de calitatea luptei cu inamicii și de sugestii pentru îmbunătățirea sistemului de generare procedurală.

În continuare, vom examina răspunsurile:

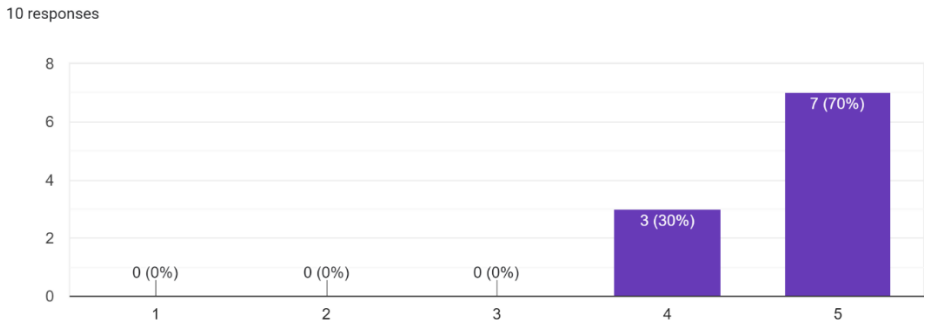
5.2.1 Întrebarea numărul 1

Cum considerați calitatea generării procedurale a terenului pentru biomul de câmpie (Plains)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, 3 au dat nota 4, iar 7 au dat nota 5, după cum se poate observa și în figura 15

Răspunsurile sugerează că generarea procedurală a biomului de câmpie este de înaltă calitate, variind de la bună la foarte bună.



Figură 15 – Răspunsurile pentru întrebarea numărul 1

5.2.2 Întrebarea numărul 2

Cum considerați calitatea graficii biomului de câmpie (Plains)

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, 4 au dat nota 4, iar 6 au dat nota 5, după cum se poate observa și în figura 16

Răspunsurile sugerează că grafica din biomul de câmpie este de înaltă calitate, variind de la bună la foarte bună.

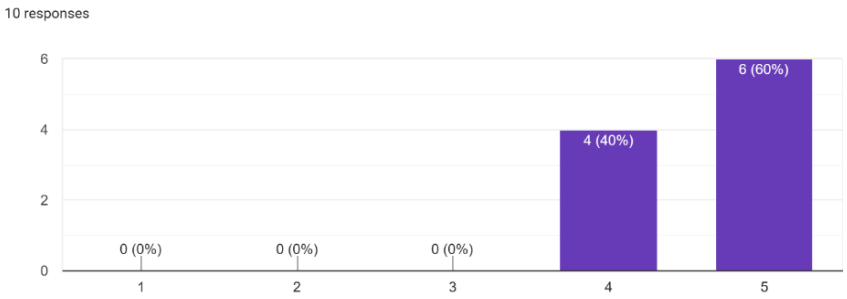


Figura 16 – Răspunsurile pentru întrebarea numărul 2

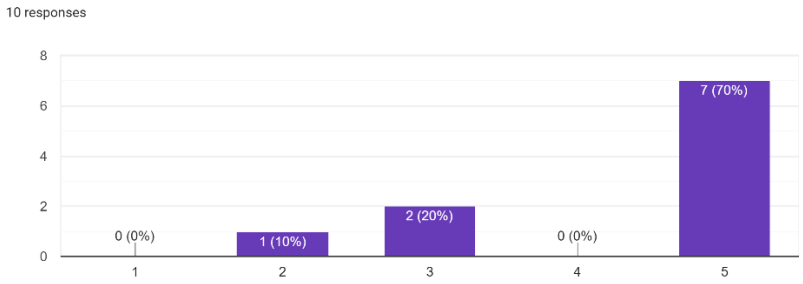
5.2.3 Întrebarea numărul 3

Cum considerați calitatea generării procedurale pentru biomul de deșert (Desert)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, unul a dat nota 1, doi au dat nota 3, iar 7 au dat nota 5, după cum se poate observa și în figura 17

Răspunsurile sugerează că generarea procedurală a biomului de deșert este de calitate bună, variind de la slabă la foarte bună.



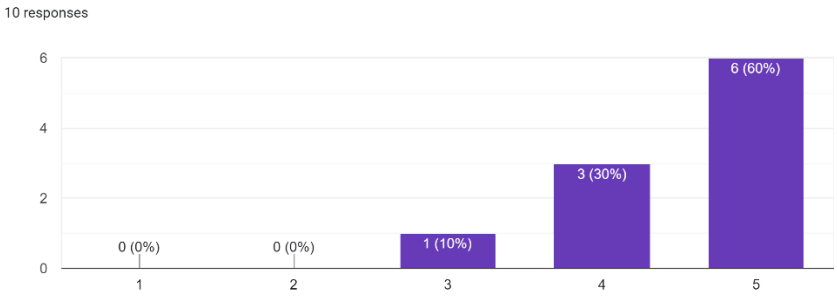
Figură 27 – Răspunsurile pentru întrebarea numărul 3

5.2.4 Întrebarea numărul 4

Cum considerați calitatea graficii biomului de deșert (Desert)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, 4 au dat nota 4, iar 6 au dat nota 5, după cum se poate observa și în figura 18



Figură 38 – Răspunsurile pentru întrebarea numărul 4

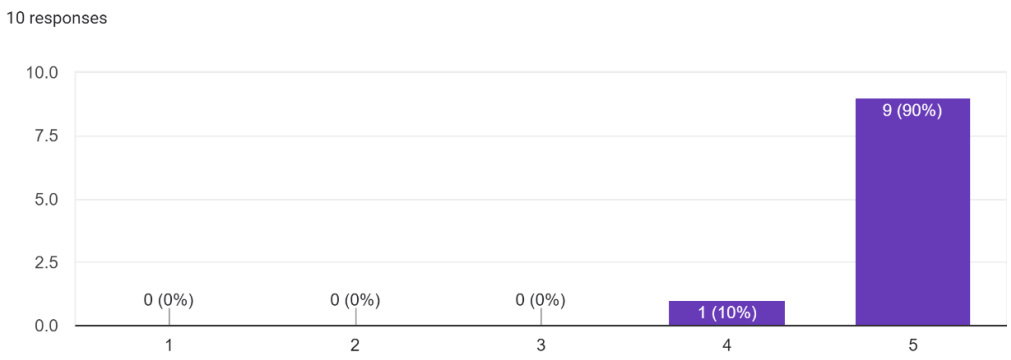
5.2.5 Întrebarea numărul 5

Cum considerați calitatea generării procedurale pentru biomul de insule (Islands)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, unul au dat nota 4, iar 9 au dat nota 5, după cum se poate observa și în figura 19

Răspunsurile sugerează că generarea procedurală a biomului de insule este de calitate foarte bună, variind de la bună la foarte bună.



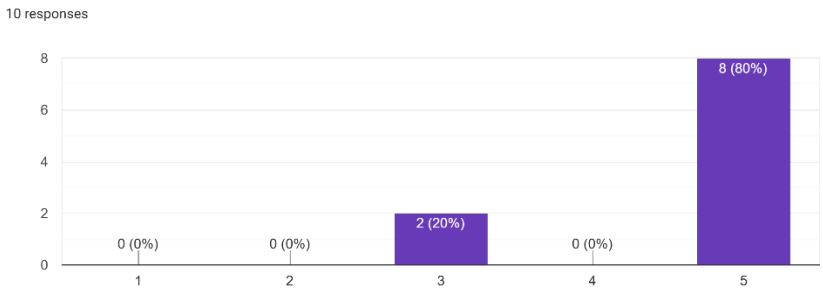
Figură 49 – Răspunsurile pentru întrebarea numărul 5

5.2.6 Întrebarea numărul 6

Cum considerați calitatea graficii biomului de insule (Islands)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, 2 au dat nota 3, iar 8 au dat nota 5, după cum se poate observa și în figura 20



Figură 20 – Răspunsurile pentru întrebarea numărul 6

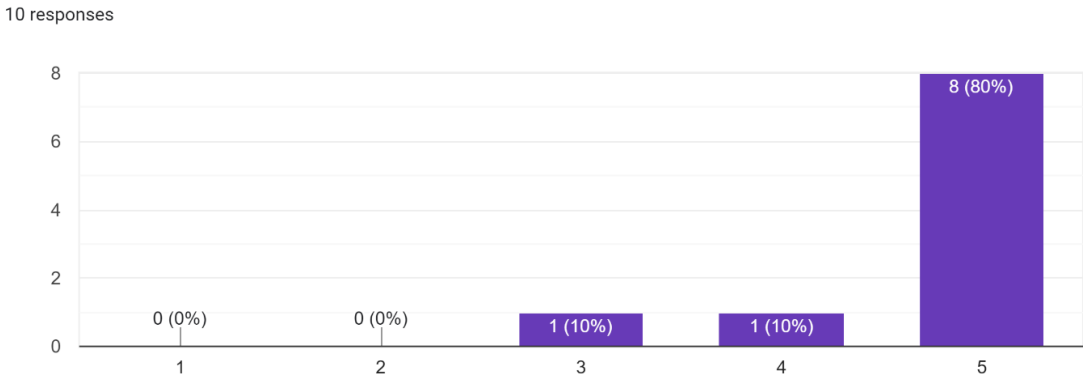
5.2.7 Întrebarea numărul 7

Cum considerați calitatea generării procedurale pentru biomul de peșteri (Caves)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, unul au dat nota 3, unul au dat nota 4, iar 8 au dat nota 5, după cum se poate observa și în figura 21

Răspunsurile sugerează că generarea procedurală a biomului de peșteri este de calitate foarte bună, variind de la moderată la foarte bună.



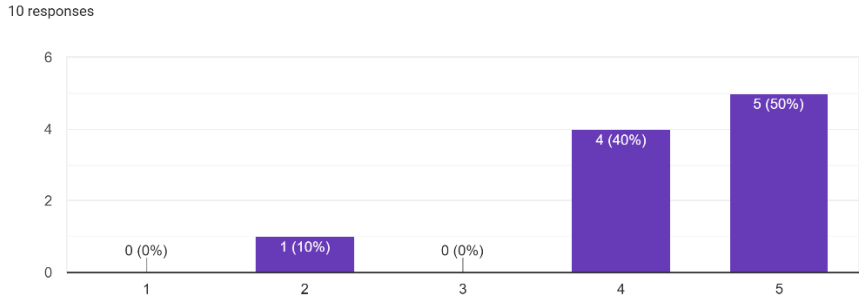
Figură 21 – Răspunsurile pentru întrebarea numărul 7

5.2.8 Întrebarea numărul 8

Cum considerați calitatea graficii biomului de peșteri (Caves)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, unul au dat nota 2, 5 au dat nota 4, iar 5 au dat nota 5, după cum se poate observa și în figura 22



Figură 22 – Răspunsurile pentru întrebarea numărul 8

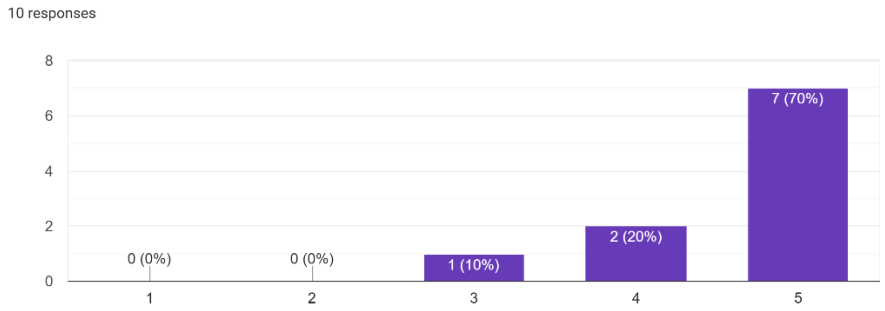
5.2.9 Întrebarea numărul 9

Cum apreciați calitatea d.p.d.v. al formei si al distribuției structurilor pe teren (copaci in biomul de câmpie, cactuși in biomul de deșert, palmieri și corali în biomul de insule)?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, unul au dat nota 3, doi au dat nota 4, iar 7 au dat nota 5, după cum se poate observa și în figura 23

Răspunsurile sugerează că forma si distribuția structurilor este de calitate foarte bună, variind de la moderată la foarte bună.



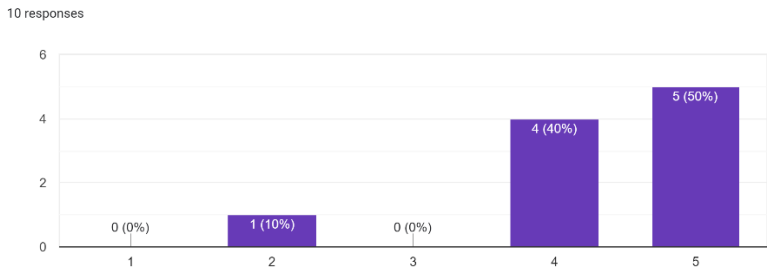
Figură 23 – Răspunsurile pentru întrebarea numărul

5.2.10 Întrebarea numărul 10

Cum considerați calitatea graficii structurilor amplasate pe teren (copaci in biomul de câmpie, cactuși in biomul de deșert, palmieri și corali în biomul de insule) ?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată

Din cei 10 respondenți, unul au dat nota 2, patru au dat nota 4, iar 5 au dat nota 5, după cum se poate observa și în figura 24



Figură 24 – Răspunsurile pentru întrebarea numărul 10

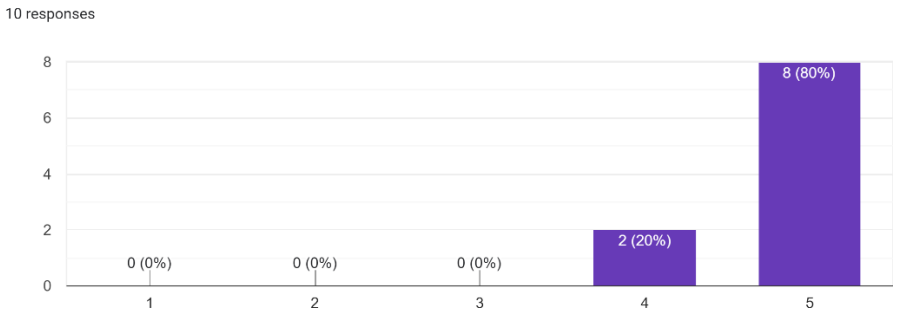
5.2.11 Întrebarea numărul 11

Toate biomurile au fost generate cu același mecanism de generare procedurală. Cum apreciați potențialul acestui sistem d.p.d.v. al diversității rezultatelor care pot fi obținute?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând un potențial foarte scăzut, iar 5 reprezentând o potențial foarte ridicat

Din cei 10 respondenți, doi au dat nota 4, iar 8 au dat nota 5, după cum se poate observa și în figura 25

Răspunsurile sugerează că mecanismul de generare procedurală are un potențial foarte bun, variind de la bun la foarte bună.



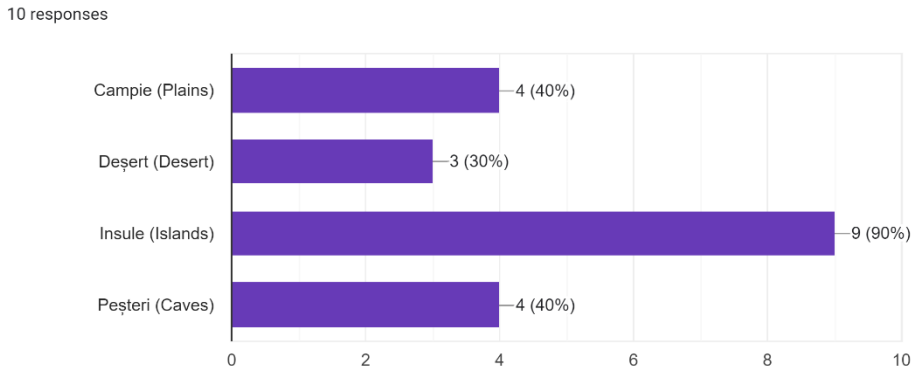
Figură 25 – Răspunsurile pentru întrebarea numărul 11

5.2.12 Întrebarea numărul 12

Care sunt biomurile dumneavoastră preferate?

La această întrebare respondenții puteau să aibă mai multe variante de răspuns.

Conform răspunsurilor din figura 26, biomul preferat este cel de insule, pe când pentru celelalte biomuri nu există o preferință demn de luată în seamă.



Figură 26 – Răspunsurile pentru întrebarea numărul 12

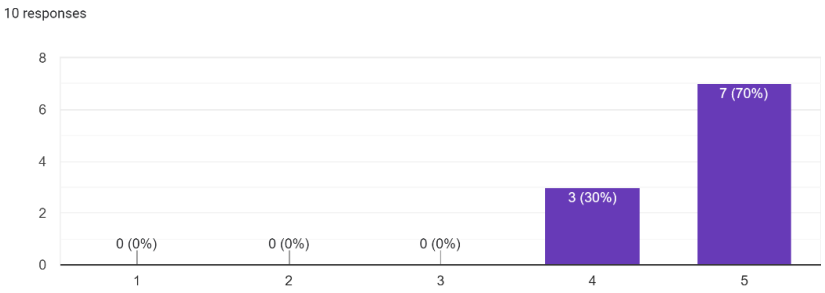
5.2.13 Întrebarea numărul 13

Cat de intuitiv considerați controlul personajului?

Scala răspunsurilor este de la 1 la 5, 1 reprezentând un control foarte puțin intuitiv, iar 5 reprezentând un control foarte intuitiv

Din cei 10 respondenți, trei au dat nota 4, iar 7 au dat nota 5, după cum se poate observa și în figura 27

Răspunsurile sugerează că mecanismul de controlul al personajului este foarte intuitiv, variind de la intuitiv la foarte intuitiv.



Figură 27 – Răspunsurile pentru întrebarea numărul 13

5.2.14 Întrebarea numărul 14

Cum considerați calitatea luptei cu inamicii?

Din cei 10 respondenți, doi au dat nota 2, unul a dat nota 1, patru au dat nota 4, iar cinci au dat nota cinci, după cum se poate observa și în figura 28

Scala răspunsurilor este de la 1 la 5, 1 reprezentând o calitate foarte scăzută, iar 5 reprezentând o calitate foarte ridicată.

Răspunsurile sugerează că luptei cu inamicii are o calitate moderată, variind de la scăzută la foarte bună.

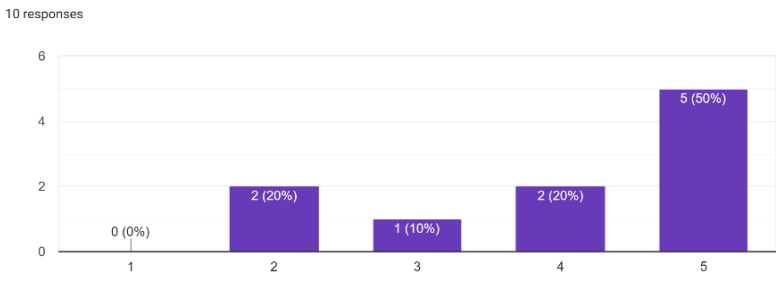


Figura 28 – Răspunsurile pentru întrebarea numărul 14

5.2.15 Întrebarea numărul 15

Ce îmbunătățiri considerați că ar putea fi aduse mecanismului de generare procedurală?

Pentru această întrebare cu răspuns liber am primit următoarele răspunsuri legate de generarea procedurală:

1. „Să se tina cont de marginile lumii, pentru ca copacii si cactușii la capetele lumii sa nu fie generate întrerupt”.
2. „O eventuală generare infinită a terenului, pe măsură ce jucătorul explorează lumea
3. „Generarea infinita de teren.”
4. „Peșterile pot avea mai multe puncte de interes înăuntrul lor”
5. „Implementarea mai multor bioame în aceeași scenă”

Din aceste răspunsuri putem extrage generarea unui spațiu infinit de joc, cu peșteri interesante și mai multe biome în aceeași scenă reprezintă o direcție bună pentru extinderea sistemului de generare procedurală în viitor.

De asemenea, au fost date următoarele răspunsuri pentru îmbunatațirea generală a jocului:

1. „Inamici mai rapizi”
2. „Upgrade la hitboxurile mobilor”

Din aceste răspunsuri putem extrage faptul că sistemul de inamici mai poate fi rafinat pentru a obține un comportament mai dezirabil pentru jucători.

6 CONCLUZII

În această lucrare, am prezentat un joc video de aventură 3D generat procedural care pune în evidență capacitățile și provocările generației procedurale în proiectarea jocurilor video. Jocul prezintă patru scene diferite: câmpie, deșert, insule tropicale și peșteri, fiecare cu un teren distinct.

Am oferit o descriere și o analiză detaliată a algoritmilor de generare procedurală folosiți în joc, cum ar fi generarea terenului. Am discutat, de asemenea, despre alegerile de proiectare și compromisurile implicate în crearea unui joc video generat procedural, cum ar fi alegerea unor setări corespunzătoare pentru a genera zgomotul folosit de sistemele de generare a terenului.

Am examinat sistemul de generare procedurală din punct de vedere al timpului necesar pentru a genera terenul din cele patru scene componente, unde am discutat motivele principale pentru care anumite componente ale sistemului de generare aveau un timp de generare diferit de la o scenă la alta. De asemenea, am analizat feedback-ul dat de jucători pentru a determina nivelul de calitate al scenelor atât d.p.d.v. al generării de teren, cât si al modelelor 3D generate. Mai mult, feedback-ul a fost folosit pentru a determina posibile aspecte ale jocului ce pot fi îmbunătățite, precum posibilitatea de genera lumea infinit într-un mod performant.

Prin prezentarea acestui joc video de aventură 3D generat procedural, am demonstrat potențialul și limitele unui sistem de generare procedurală.

7 BIBLIOGRAFIE

[1] „Minecraft," [Interactiv]. Available: https://www.minecraft.net/en-us/about-minecraft.

[2] Minecraft, „Creative vs Survival," [Interactiv]. Available: https://www.minecraft.net/en-us/article/creative-vs-survival-mode.

[3] J. Clement, „Statista," 2 August 2022. [Interactiv]. Available: https://www.statista.com/statistics/680124/minecraft-unit-sales-worldwide/.

[4] Hello Games, „Release Log," [Interactiv]. Available: https://www.nomanssky.com/release-log/?cli_action=1687791815.678.

[5] No Man's Sky Wiki, „Procedural generation," [Interactiv]. Available: https://nomanssky-archive.fandom.com/wiki/Procedural_generation.

[6] Iron Gate Studio, „Valheim," [Interactiv]. Available: https://www.valheimgame.com/#about.

[7] Refactoring Guru, „Chain of Responsibility," [Interactiv]. Available: https://refactoring.guru/design-patterns/chain-of-responsibility.

[8] Kenney, „Animated Characters 2," [Interactiv]. Available: https://kenney.nl/assets/animated-characters-2.

[9] K. Iglesias, „Unity Asset Store - Melee Warrior Animations FREE," [Interactiv]. Available: https://assetstore.unity.com/packages/3d/animations/melee-warrior-animations-free-165785.

[10] DancingEyebrows, „Unity Asset Store - Goblin01," [Interactiv]. Available: https://assetstore.unity.com/packages/3d/characters/goblin01-188119.

[11] Adobe, „Mixamo," [Interactiv]. Available: https://www.mixamo.com/#/.

[12] Polytope Studio, „Unity Asset Store - Lowpoly Medieval Skeleton," [Interactiv]. Available: https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/lowpoly-medieval-skeleton-free-medieval-fantasy-series-181883.

[13] SICS Games, „Unity Asset Store - Low Poly Weapons," [Interactiv]. Available: https://assetstore.unity.com/packages/3d/props/weapons/low-poly-weapons-71680.

[14] Polytope Studio, „Unity Asset Store - Lowpoly Medieval Plague Doctor,” [Interactiv]. Available: https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy/lowpoly-medieval-plague-doctor-free-medieval-fantasy-series-176809.

[15] Nvidia, „Improve Batching Using,” Nvidia, July 2004. [Interactiv]. Available: https://developer.download.nvidia.com/SDK/9.5/Samples/DEMOS/Direct3D9/src/BatchingViaTextureAtlases/AtlasCreationTool/Docs/Batching_Via_Texture_Atlases.pdf.

[16] Voxel Pack, [Interactiv]. Available: https://kenney.nl/assets/voxel-pack.

[17] Unity, „Unity Documentation,” 2021. [Interactiv]. Available: https://docs.unity3d.com/Manual/class-ScriptableObject.html. [Accesat June 2023].

[18] Unity Technologies, „Unity Learn,” 2019. [Interactiv]. Available: https://learn.unity.com/tutorial/introduction-to-object-pooling#.

[19] I. Parberry, „Designer Worlds: Procedural Generation of Infinite Terrain from Real-Wolrd Data,” *Journal of Computer Graphics Techniques,* vol. 3, nr. 1, pp. 74-85, 2014.

[20] Unity Tehnologies, „Mathf,” Unity Tehnologies, 2021. [Interactiv]. Available: https://docs.unity3d.com/ScriptReference/Mathf.html.

[21] A. J. Peck, „Github FastNoiseLite,” [Interactiv]. Available: https://github.com/Auburn/FastNoiseLite.

[22] A. J. Peck. [Interactiv]. Available: https://github.com/Auburn.