Contents

[1. Prezentarea Sistemului Informatic 3](#_Toc218628789)

[1.1 Descrierea Generală a Sistemului 3](#_Toc218628790)

[Arhitectura Modulară și Extensibilă 3](#_Toc218628791)

[Algoritmi Incluși (Exemple Demonstrative) 3](#_Toc218628792)

[Caracteristici Principale 4](#_Toc218628793)

[Stack Tehnologic 4](#_Toc218628794)

[1.2. Specificarea Cerințelor 5](#_Toc218628795)

[1.2.1 Diagrama Cazurilor de Utilizare (Diagram\_1\_UseCases) 5](#_Toc218628796)

[1.2.2 Descrierea Textuală a Cazurilor de Utilizare 6](#_Toc218628797)

[2. Analiza Sistemului Informatic 9](#_Toc218628798)

[2.1 Diagrame de Activitate 9](#_Toc218628799)

[Diagrama 2.1: Fluxul Generării Procedurale (Diagram\_2\_1\_Activity\_GenFlow) 9](#_Toc218628800)

[Diagrama 2.2: Configurarea Modulului WFC (Diagram\_2\_2\_Activity\_WFC) 10](#_Toc218628801)

[Diagrama 2.3: Exportul Scenei (Diagram\_2\_3\_Activity\_Export) 10](#_Toc218628802)

[2.2 Diagrama de Clase (Diagram\_2\_2\_Class) 11](#_Toc218628803)

[2.3 Diagrame de Stare 12](#_Toc218628804)

[Diagrama 2.3.1: Ciclul de Viață al Scenei (Diagram\_2\_3\_State\_Scene) 12](#_Toc218628805)

[Diagrama 2.3.2: Starea unui Algoritm de Generare (Diagram\_2\_3\_State\_Generator) 15](#_Toc218628806)

[2.4 Diagrame de Interacțiune 17](#_Toc218628807)

[Diagrama 2.4.1: Secvență - Adăugarea unui Algoritm WFC (Diagram\_2\_4\_Sequence\_AddAlgo) 17](#_Toc218628808)

[Diagrama 2.4.2: Secvență - Execuția Pipeline-ului (Diagram\_2\_4\_Communication\_Exec) 17](#_Toc218628809)

[Diagrama 2.4.3: Secvență - Monitorizare Performanță (Diagram\_2\_4\_Timing\_Perf) 18](#_Toc218628810)

[2.5 Diagrame BPMN 19](#_Toc218628811)

[Diagrama de Proces BPMN (Proces\_BPMN.png) 19](#_Toc218628812)

[Diagrama de Colaborare BPMN (Colaborare\_BPMN.png) 20](#_Toc218628813)

[3. Proiectarea Sistemului Informatic 22](#_Toc218628814)

[3.1 Diagrama de Clase Detaliată (Diagram\_3\_2\_Detailed\_Class) 22](#_Toc218628815)

[3.2 Structura Bazei de Date (Diagram\_3\_2\_ERD) 24](#_Toc218628816)

[3.3 Proiectarea Interfețelor Utilizator 25](#_Toc218628817)

[3.4 Diagrama de Componente (Diagram\_3\_4\_Component) 30](#_Toc218628818)

[3.5 Diagrama de desfăşurare (Diagram\_3\_5\_Deployment) 32](#_Toc218628819)

[Anexe 33](#_Toc218628820)

[Lista Completă a Diagramelor 33](#_Toc218628821)

# 1. Prezentarea Sistemului Informatic

## 1.1 Descrierea Generală a Sistemului

RAMY Procedural Generation Studio este un framework avansat și modular destinat generării procedurale de conținut tridimensional. Aplicația oferă o soluție completă pentru artiști tehnici și dezvoltatori de jocuri, permițând proiectarea și iterarea rapidă a scenelor 3D complexe fără a necesita modelare manuală tradițională.

### Arhitectura Modulară și Extensibilă

Punctul forte al sistemului este arhitectura sa **complet modulară și extensibilă**. Framework-ul nu este limitat la un set fix de algoritmi - orice dezvoltator poate implementa și integra propriile module de generare prin simpla implementare a interfeței IGenerator. Această abordare "plugin-based" permite:

* **Extensibilitate nelimitată**: Adăugarea de noi algoritmi fără modificarea codului sursă al motorului principal
* **Personalizare totală**: Fiecare studio sau dezvoltator poate crea generatoare specializate pentru nevoile proiectului
* **Combinare liberă**: Pipeline-ul permite stacking-ul oricâtor generatoare, în orice ordine, creând rezultate unice prin combinare
* **Hot-swapping**: Modulele pot fi încărcate și descărcate dinamic la runtime

### Algoritmi Incluși (Exemple Demonstrative)

Sistemul vine pre-configurat cu câteva implementări demonstrative ale interfeței IGenerator:

* **Perlin Noise Generator** - generare de terenuri și suprafețe organice folosind noise coherent
* **Wave Function Collapse (WFC)** - construcție de structuri coerente bazate pe reguli de adiacență
* **Shape Grammars** - generare de arhitectură procedurală folosind reguli gramaticale formale
* **Scatter Generator** - distribuție pseudo-aleatoare a vegetației și obiectelor decorative

**IMPORTANT**: Acestea sunt doar exemple incluse pentru demonstrație. Datorită arhitecturii modulare, utilizatorii pot dezvolta și integra oricând algoritmi noi precum:

* L-Systems pentru vegetație și structuri fractale
* Voronoi Diagrams pentru distribuție spațială
* Marching Cubes pentru mesh-uri volumetrice
* Diamond-Square pentru heightmaps
* Dungeon Generators bazate pe BSP trees
* Cellular Automata pentru simulări
* Orice altă metodă de generare procedurală imaginabilă

### Caracteristici Principale

**Pipeline de Procesare Modular:**

Sistemul folosește o stivă (stack) de generatoare care se execută secvențial. Utilizatorul poate adăuga, șterge sau reordona pașii prin interfața drag-and-drop. Fiecare pas primește datele de la pasul anterior, permițând transformări incrementale (ex: Generează Teren → Aplică Eroziune → Adaugă Vegetație → Plasează Clădiri).

**Configurare Dinamică:**

Fiecare generator expune propriii parametri configurabili prin interfața comună GetParameters(). Panoul de inspecție se adaptează automat pentru a afișa controalele relevante.

**Previzualizare în Timp Real:**

Viewport-ul 3D este actualizat imediat după fiecare modificare, permițând iterație rapidă și feedback instantaneu.

**Export Versatil:**

Rezultatele pot fi exportate în formate industriale standard (OBJ, FBX), compatibile cu Unity și Unreal Engine.

### Stack Tehnologic

* **C++ (Core Engine)**: Performanță nativă pentru procesarea milioanelor de vertecși
* **OpenGL/Vulkan (Randare)**: Randare hardware-accelerated și compute shaders
* **SQLite (Persistență)**: Stocare eficientă a presetărilor și configurațiilor
* **JSON (Configurări)**: Format ușor de citit pentru serializarea parametrilor

## 1.2. Specificarea Cerințelor

### 1.2.1 Diagrama Cazurilor de Utilizare (Diagram\_1\_UseCases)

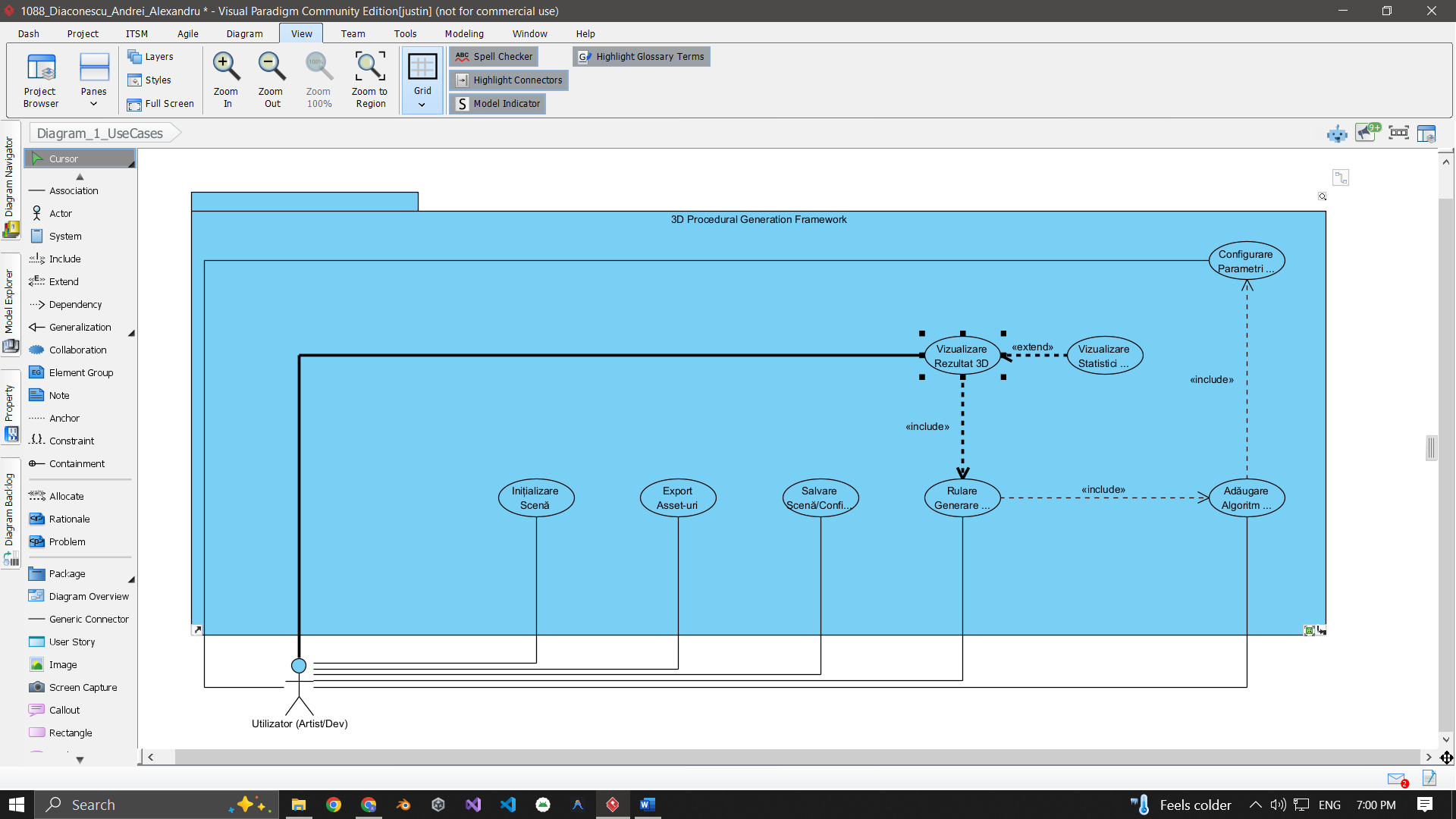
Diagrama prezintă interacțiunile dintre actorul principal "Utilizator (Artist/Dev)" și sistemul "3D Procedural Generation Framework".

**Cazurile de utilizare identificate:**

* **UC1 - Inițializare Scenă**: Pornirea unei noi scene cu parametri impliciti
* **UC2 - Configurare Parametri Globali**: Setarea seed-ului, dimensiunilor și altor setări generale
* **UC3 - Adăugare Algoritm Generare**: Selectarea și adăugarea unui algoritm (Noise, WFC etc.) în stiva de procesare
* **UC4 - Rulare Generare (Execute Pipeline)**: Declanșarea execuției tuturor algoritmilor din stivă
* **UC5 - Vizualizare Rezultat 3D**: Navigarea și inspectarea scenei generate
* **UC6 - Salvare Scenă/Configurație**: Persistarea proiectului curent
* **UC7 - Export Asset-uri**: Exportul geometriei în format OBJ sau FBX
* **UC8 - Vizualizare Statistici (FPS/Poly)**: Afișarea informațiilor de performanță

**Relațiile dintre cazuri:**

* UC3 "include" UC2: Adăugarea unui algoritm necesită obligatoriu configurarea parametrilor globali mai întâi
* UC4 "include" UC3: Rularea pipeline-ului necesită cel puțin un algoritm în stivă
* UC5 "include" UC4: Vizualizarea rezultatului este posibilă doar după generare
* UC8 "extend" UC5: Statisticile sunt opționale și extind funcția de vizualizare



### 1.2.2 Descrierea Textuală a Cazurilor de Utilizare

În această etapă, a fost realizată o **analiză textuală** a cerințelor funcționale pentru a identifica actorii, cazurile de utilizare și clasele conceptuale ale sistemului RAMY Procedural Generation Studio. Această metodă permite extragerea riguroasă a elementelor care vor constitui arhitectura logică a aplicației.

#### A. Identificarea Actorilor

* **Utilizator (Artist/Developer):** Reprezintă actorul principal al sistemului, având rolul de a configura parametrii și de a gestiona pipeline-ul de generare procedurală.

#### B. Identificarea Claselor Conceptuale (Entități)

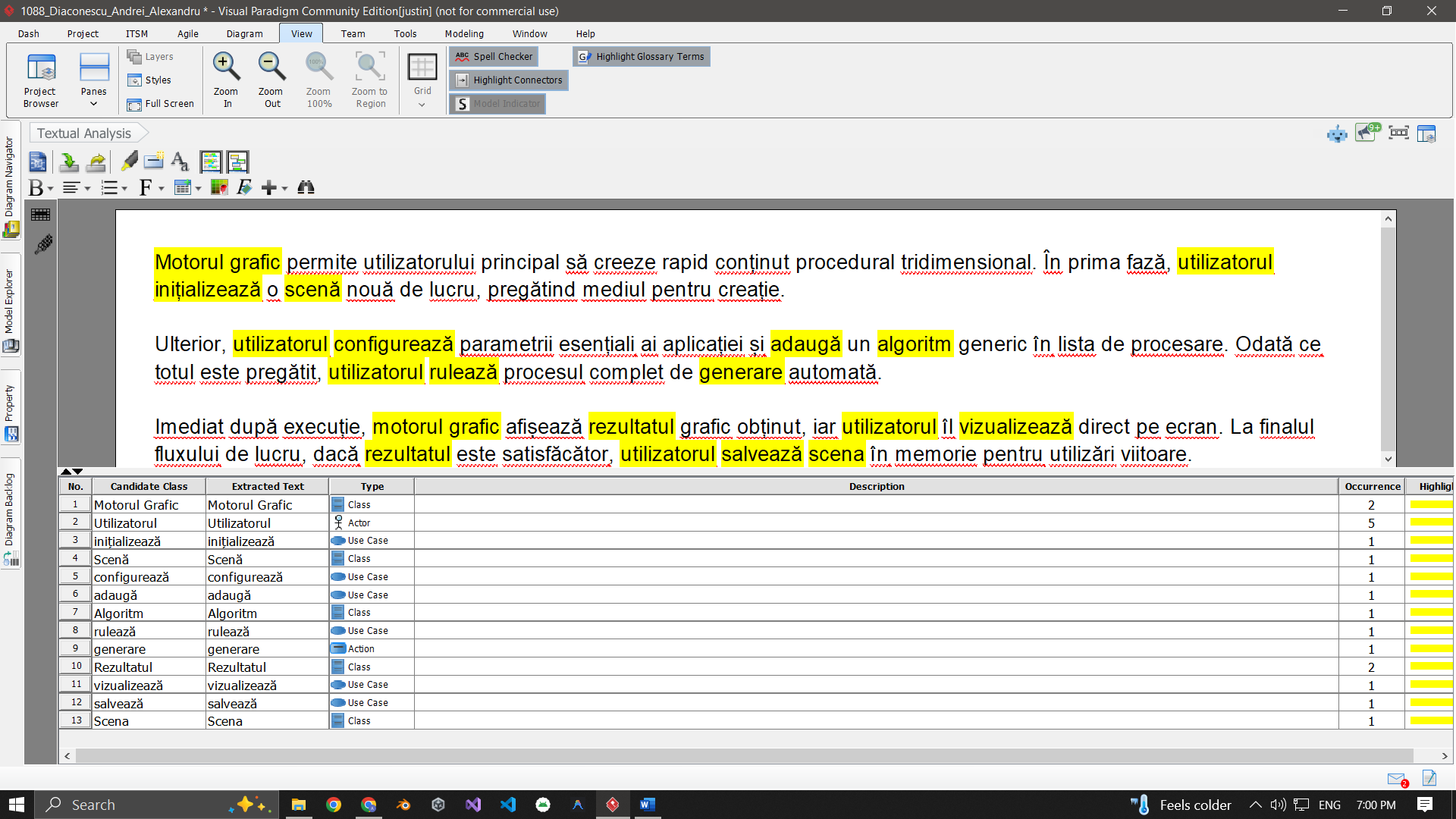
Aceste elemente reprezintă obiectele de bază gestionate de sistem în timpul execuției:

* **Motorul Grafic:** Componenta centrală responsabilă pentru procesarea instrucțiunilor și coordonarea resurselor hardware.
* **Scenă:** Spațiul virtual care stochează obiectele 3D, configurațiile globale și rezultatul generării.
* **Algoritm:** Modulul logic (ex: Perlin Noise, WFC) care definește regulile de construcție a geometriei.
* **Rezultatul:** Produsul final al pipeline-ului, reprezentat prin mesh-uri 3D și date vizuale.

#### C. Descrierea Textuală a Cazurilor de Utilizare (6 UC-uri principale)

Următoarele cazuri de utilizare au fost extrase din textul de specificații, reprezentând funcționalitățile cheie oferite utilizatorului:

1. **UC1: Inițializează Scenă**
   * **Descriere:** Permite utilizatorului pregătirea mediului de lucru prin crearea unei instanțe noi de proiect.
   * **Rezultat:** O scenă vidă, gata pentru configurare.
2. **UC2: Configurează Parametri**
   * **Descriere:** Utilizatorul definește atributele globale (seed, dimensiuni, world scale) necesare procesului de calcul.
   * **Rezultat:** Parametrii sistemului sunt validați și stocați.
3. **UC3: Adaugă Algoritm**
   * **Descriere:** Inserarea unui modul de generare generic în stiva de procesare a aplicației.
   * **Rezultat:** Pipeline-ul este actualizat cu un nou pas de execuție.
4. **UC4: Rulează Procesul**
   * **Descriere:** Declanșarea algoritmilor de generare automată în ordinea stabilită în stivă.
   * **Rezultat:** Transformarea datelor numerice în geometrie tridimensională.
5. **UC5: Vizualizează Rezultatul**
   * **Descriere:** Redarea grafică instantanee în viewport-ul 3D pentru inspecția vizuală a scenei generate.
   * **Rezultat:** Feedback vizual pentru utilizator.
6. **UC6: Salvează Scena**
   * **Descriere:** Persistarea configurațiilor și a geometriei în memorie sau în baza de date pentru acces ulterior.
   * **Rezultat:** Starea curentă a scenei este salvată cu succes.



# 2. Analiza Sistemului Informatic

## 2.1 Diagrame de Activitate

### Diagrama 2.1: Fluxul Generării Procedurale (Diagram\_2\_1\_Activity\_GenFlow)

Această diagramă ilustrează procesul complet de generare procedurală, de la acțiunea utilizatorului până la randarea finală.

**Elementele diagramei:**

* **Nod de start**: Procesul începe când utilizatorul apasă "Generează"
* **Romb de decizie "Stiva de Generare e goală?"**: Punctul de bifurcație al fluxului
* Ramura DA: Conduce la afișarea erorii și oprirea procesului
* Ramura NU: Continuă cu procesarea normală
* **Partiție "Buclă Procesare"**: Iterarea prin toți generatorii din stivă
* While loop: "Mai sunt generatori în stivă?"
* Activități în buclă: Extragere generator → Preluare parametri → Executare algoritm → Modificare date GameObject
* **Post-procesare**: Recalculare normale, actualizare buffere OpenGL, deblocare interfață
* **Nod final**: Randare scenă finală

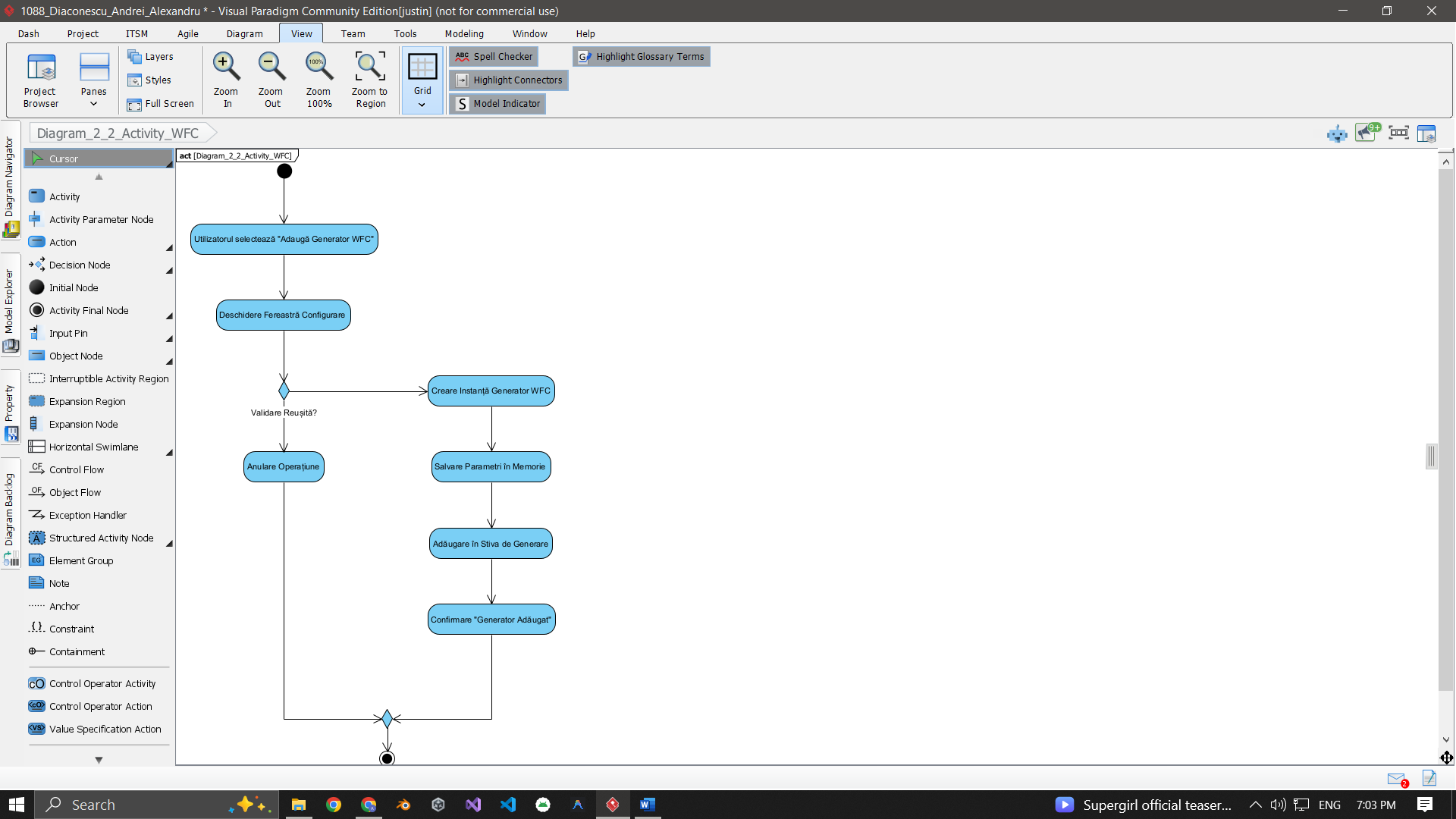


### Diagrama 2.2: Configurarea Modulului WFC (Diagram\_2\_2\_Activity\_WFC)

Prezintă procesul de configurare pentru algoritmul Wave Function Collapse.

**Elementele diagramei:**

* **Acțiune inițială**: Utilizatorul selectează "Adaugă Generator WFC" și se deschide fereastra de configurare
* **Buclă repeat-until**: Utilizatorul definește setul de reguli (Tiles) și dimensiunea grid-ului
* **Romb de decizie "Erori de Adiacență?"**: Sistemul verifică consistența regulilor
* DA: Afișare avertisment "Reguli Invalide" cu sugestie de corectare, bucla continuă
* NU: Se iese din buclă
* **Romb "Validare Reușită?"**:
* DA: Creare instanță WFC, salvare parametri, adăugare în stivă, confirmare
* NU: Anulare operațiune

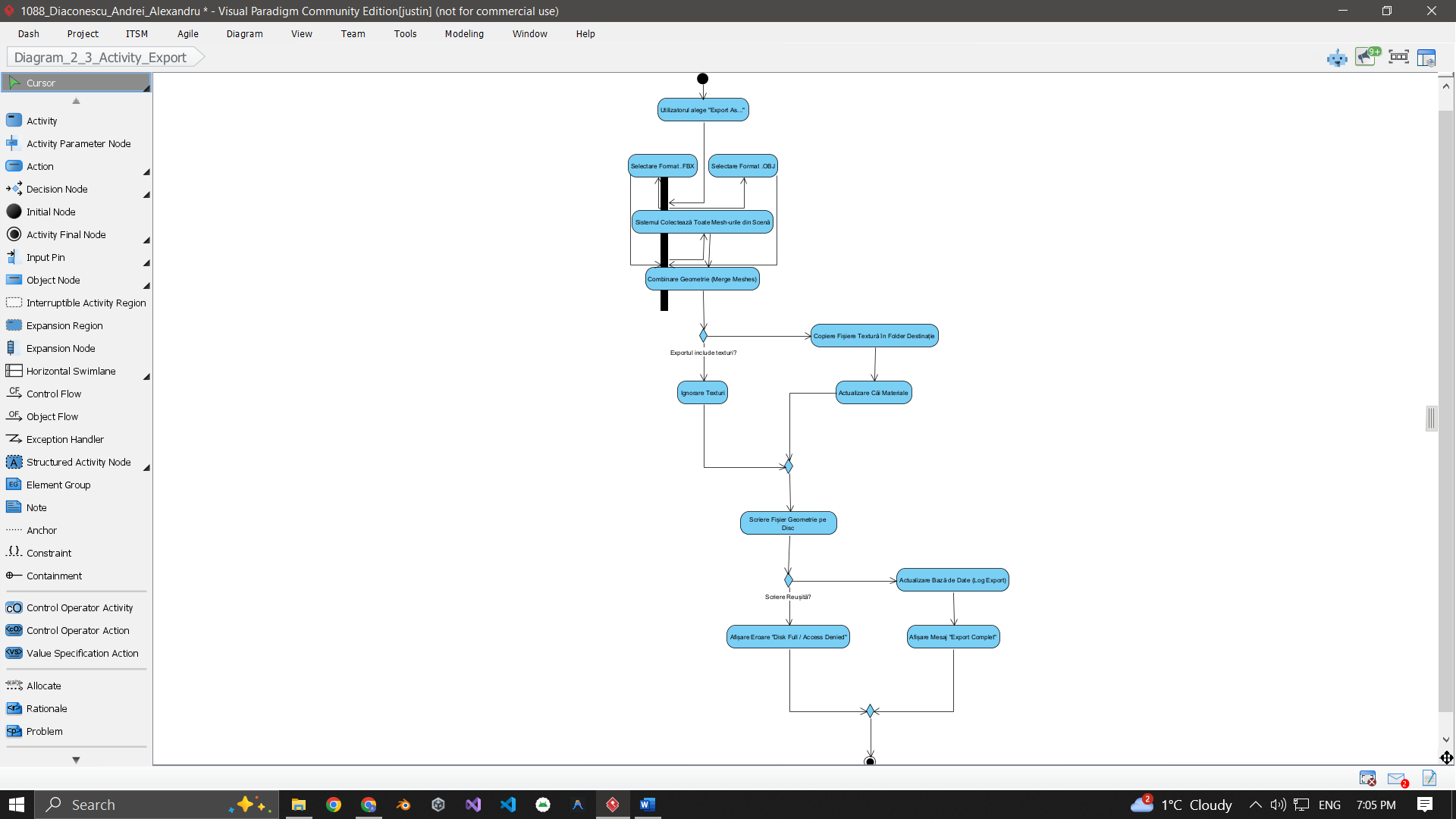


### Diagrama 2.3: Exportul Scenei (Diagram\_2\_3\_Activity\_Export)

Descrie procesul de export al geometriei 3D.

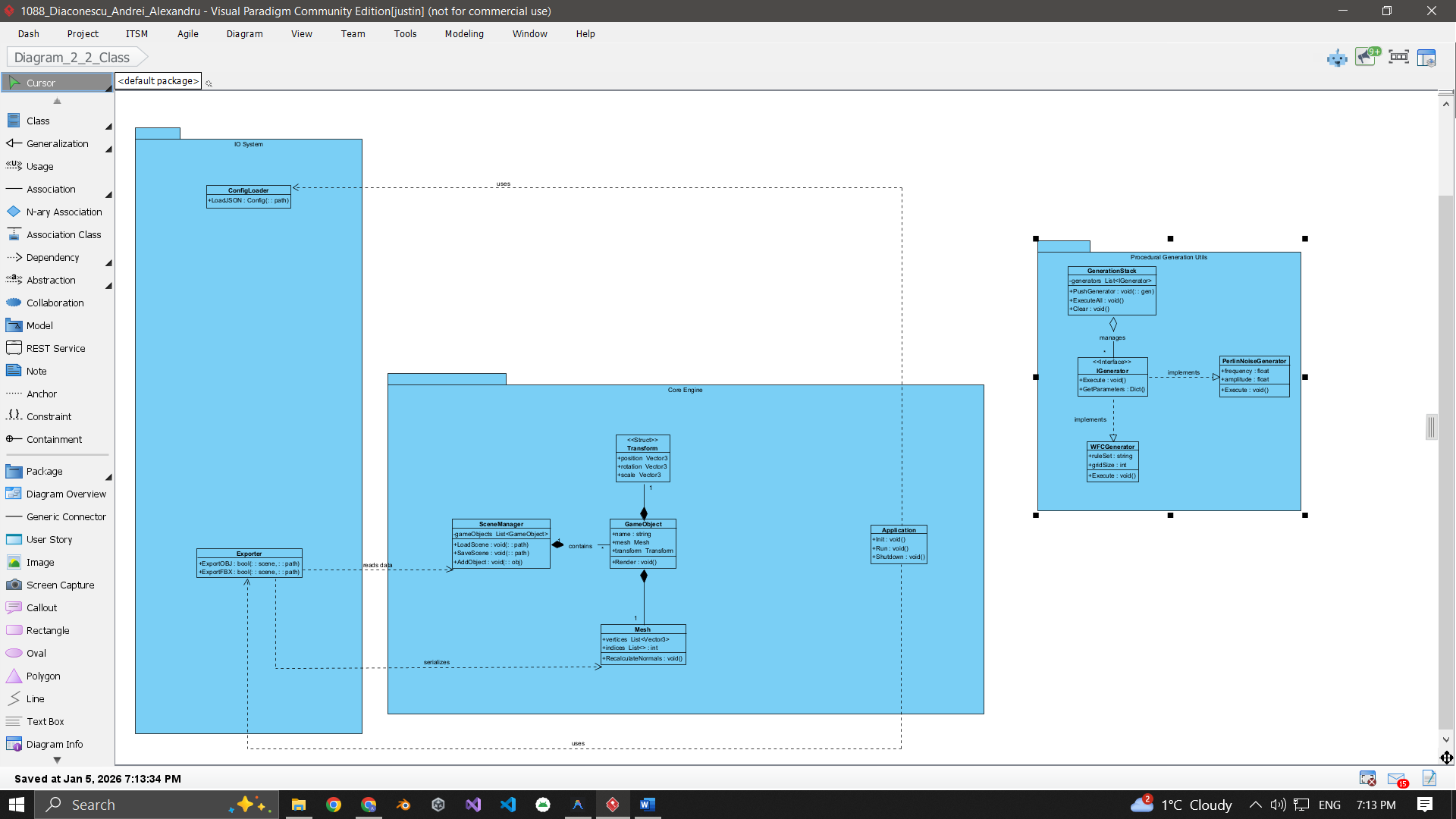
**Elementele diagramei:**

* **Fork/Join (procesare paralelă)**: Sistemul permite selectarea formatului .OBJ sau .FBX în paralel
* **Colectare date**: Sistemul colectează toate mesh-urile din scenă și le combină
* **Romb "Exportul include texturi?"**:
* DA: Copiere fișiere textură, actualizare căi materiale
* NU: Ignorare texturi
* **Romb "Scriere Reușită?"**:
* DA: Actualizare bază de date (log export), afișare mesaj "Export Complet"
* NU: Afișare eroare "Disk Full / Access Denied"



## 2.2 Diagrama de Clase (Diagram\_2\_2\_Class)

* Diagrama prezintă structura statică a sistemului organizată în trei pachete principale.
* **Pachetul "Core Engine":**
* **Application**: Clasa principală cu metodele Init(), Run(), Shutdown()
* **SceneManager**: Gestionează lista de GameObject-uri cu metode LoadScene(), SaveScene(), AddObject()
* **GameObject**: Entitatea de bază cu atribute name, mesh, transform și metoda Render()
* **Transform**: Struct cu position, rotation, scale (Vector3)
* **Mesh**: Conține vertices, indices și metoda RecalculateNormals()
* **Pachetul "Procedural Generation Utils":**
* **GenerationStack**: Stiva care gestionează generatorii cu metode PushGenerator(), ExecuteAll(), Clear()
* **IGenerator<>**: Interfața comună cu metode Execute() și GetParameters()
* **PerlinNoiseGenerator**: Implementare cu atribute frequency, amplitude
* **WFCGenerator**: Implementare cu atribute ruleSet, gridSize
* **Pachetul "IO System":**
* **Exporter**: Clasă responsabilă de serializarea scenei în formate externe (OBJ, FBX). Citește datele din SceneManager și serializează obiectele Mesh.
* **ConfigLoader**: Încărcare configurații JSON
* **Relații:**
* Application → SceneManager și GenerationStack (asociere)
* Application -.-> ConfigLoader : uses (dependență)
* Application -.-> Exporter : uses (dependență)
* SceneManager ◆ GameObject: compoziție 1 la multe
* GameObject ◆ Transform: compoziție
* GameObject ◆ Mesh: compoziție
* GenerationStack ○ IGenerator: agregare (manages)
* PerlinNoiseGenerator și WFCGenerator implementează IGenerator
* Exporter -.-> SceneManager : reads data (dependență)
* Exporter -.-> Mesh : serializes (dependență)



## 2.3 Diagrame de Stare

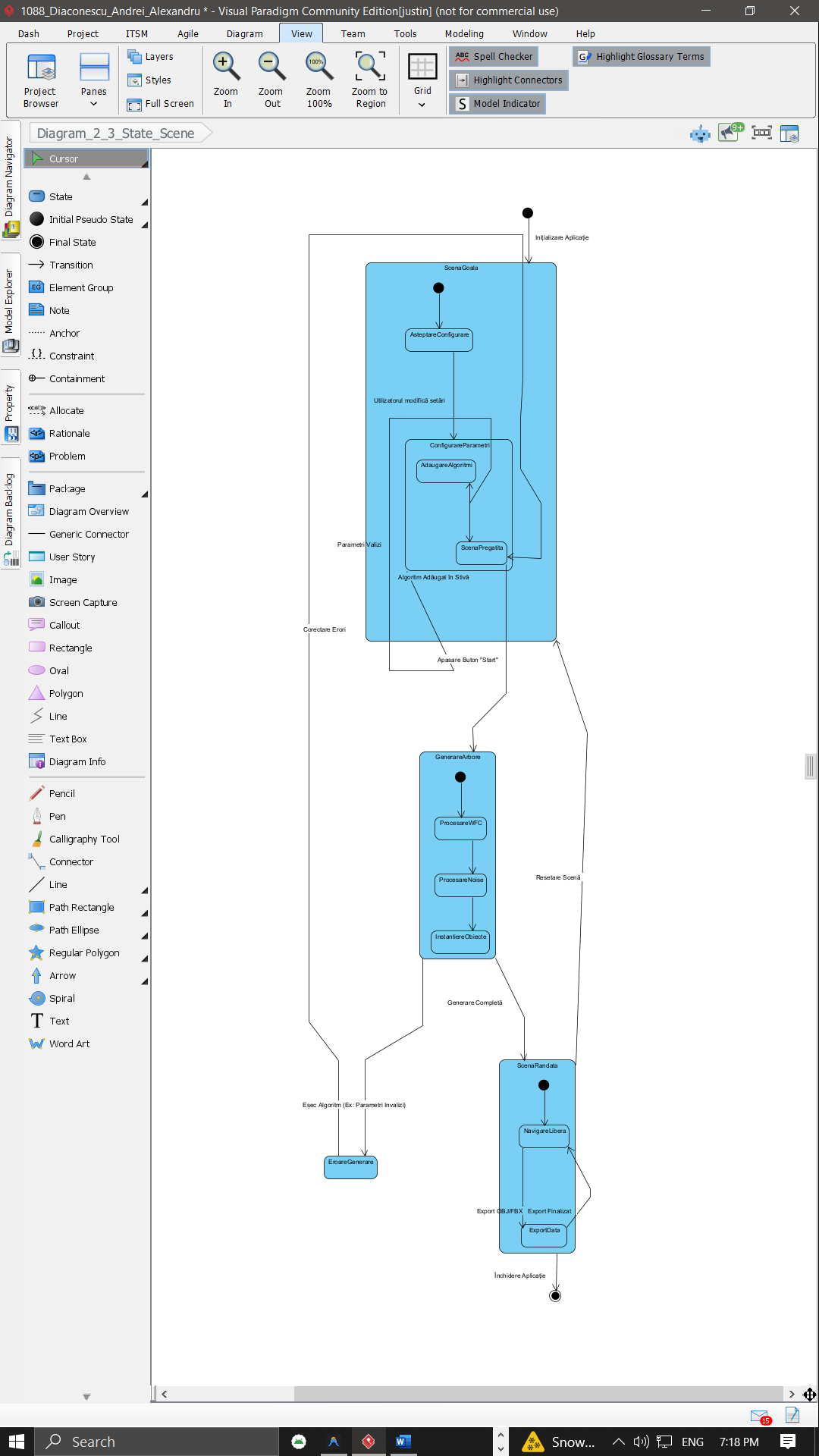
### Diagrama 2.3.1: Ciclul de Viață al Scenei (Diagram\_2\_3\_State\_Scene)

**Stările principale:**

* **ScenaGoala**: Starea inițială după pornirea aplicației
* Substare internă: AsteptareConfigurare → ConfigurareParametri
* **ConfigurareParametri**: Utilizatorul setează parametrii globali
* Tranziție către AdaugareAlgoritmi când parametrii sunt valizi
* **ScenaPregatita**: Stiva conține cel puțin un algoritm, gata de execuție
* **GenerareArbore**: Procesul de generare este în curs
* Substări: ProcesareWFC → ProcesareNoise → InstantiereObiecte
* **EroareGenerare**: Stare de eroare (ex: parametri invalizi)
* Permite revenire la ScenaPregatita după corectare
* **ScenaRandata**: Generarea s-a finalizat cu succes
* Substări: NavigareLibera ↔ ExportData

**Tranziții importante:**

* ScenaPregatita → GenerareArbore: Apăsare buton "Start"
* GenerareArbore → EroareGenerare: Eșec algoritm
* ScenaRandata → ScenaGoala: Resetare scenă
* ScenaRandata → [Final]: Închidere aplicație



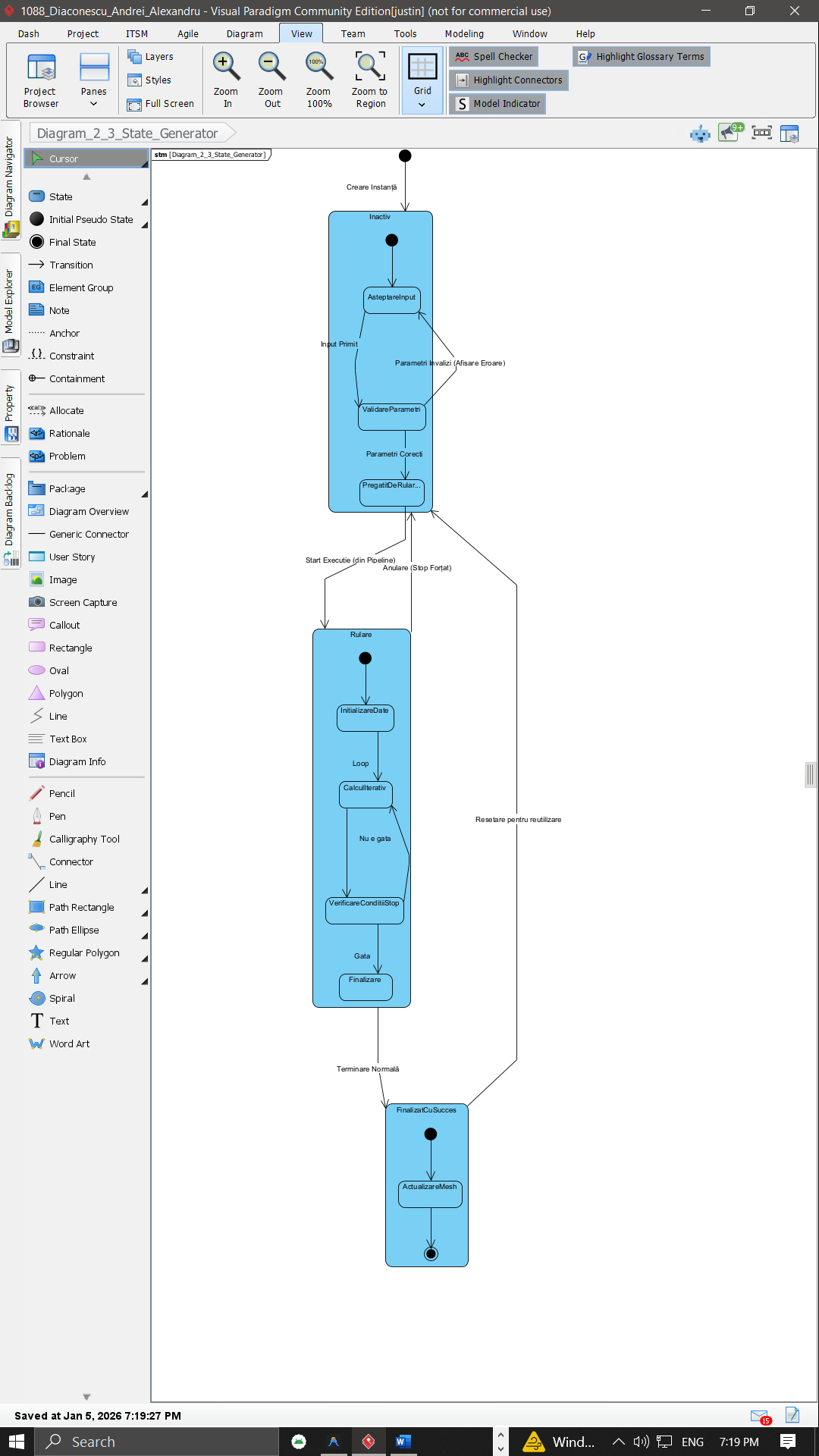
### Diagrama 2.3.2: Starea unui Algoritm de Generare (Diagram\_2\_3\_State\_Generator)

**Stările principale:**

* **Inactiv**: Algoritmul este creat dar nu rulează
* Substări: AsteptareInput → ValidareParametri → PregatitDeRulare
* Buclă la ValidareParametri dacă parametrii sunt invalizi
* **Rulare**: Algoritmul execută calculele
* Substări: InitializareDate → CalculIterativ → VerificareConditiiStop → Finalizare
* Bucla CalculIterativ ↔ VerificareConditiiStop continuă până când condiția de stop e îndeplinită
* **FinalizatCuSucces**: Algoritmul a terminat cu succes
* Substare: ActualizareMesh - mesh-ul rezultat este aplicat

**Tranziții:**

* Rulare → Inactiv: Anulare (stop forțat)
* FinalizatCuSucces → Inactiv: Resetare pentru reutilizare



## 2.4 Diagrame de Interacțiune

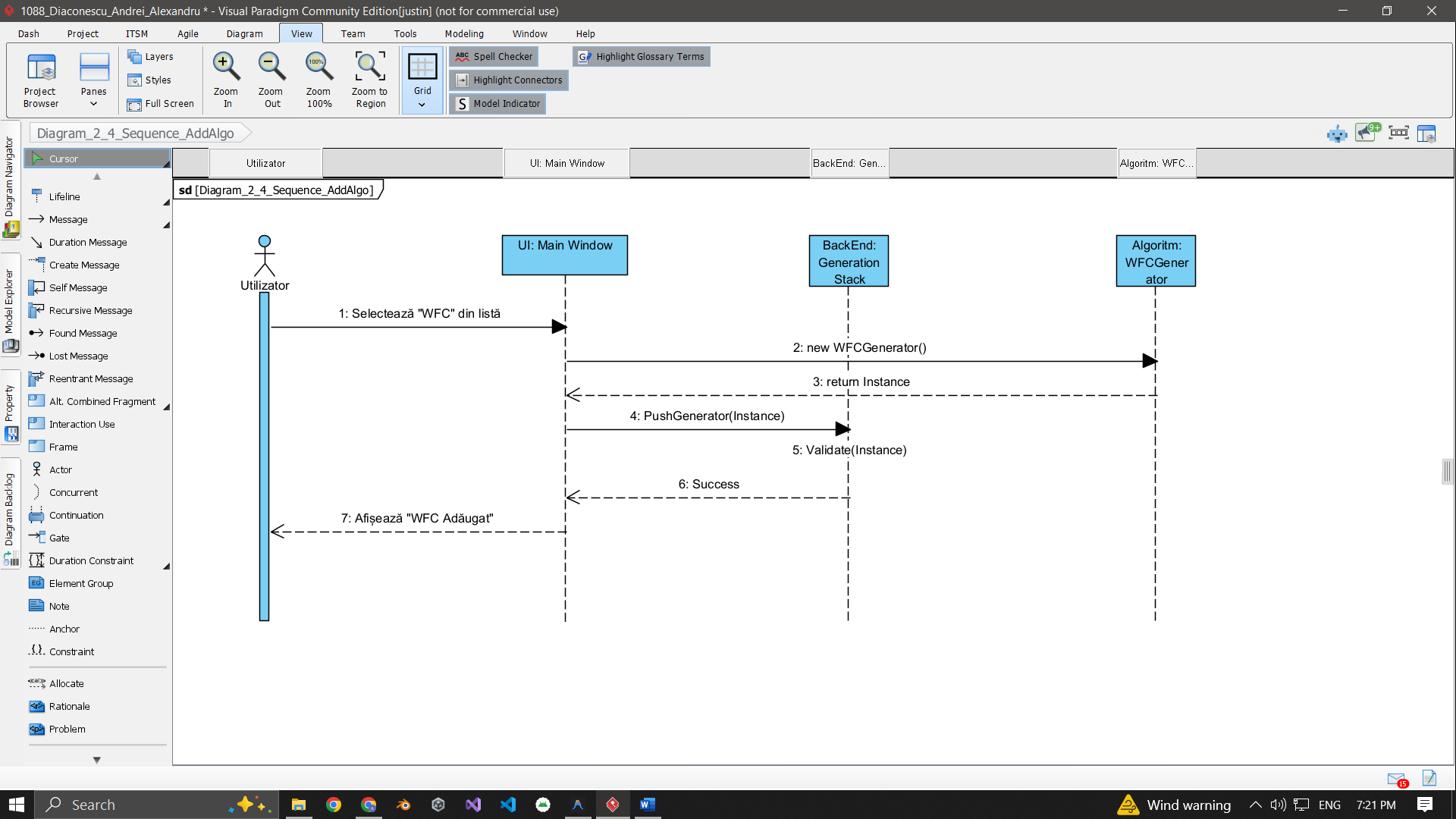
### Diagrama 2.4.1: Secvență - Adăugarea unui Algoritm WFC (Diagram\_2\_4\_Sequence\_AddAlgo)

**Participanți:**

* Utilizator (actor)
* UI: Main Window
* BackEnd: GenerationStack
* Algoritm: WFCGenerator

**Fluxul mesajelor:**

1. Utilizator → UI: Selectează "WFC" din listă
2. UI → WFC: new WFCGenerator() - creează instanță nouă
3. WFC → UI: return Instance
4. UI → Stack: PushGenerator(Instance) - adaugă în stivă
5. Stack → Stack: Validate(Instance) - validare internă
6. Stack → UI: Success
7. UI → Utilizator: Afișează "WFC Adăugat"



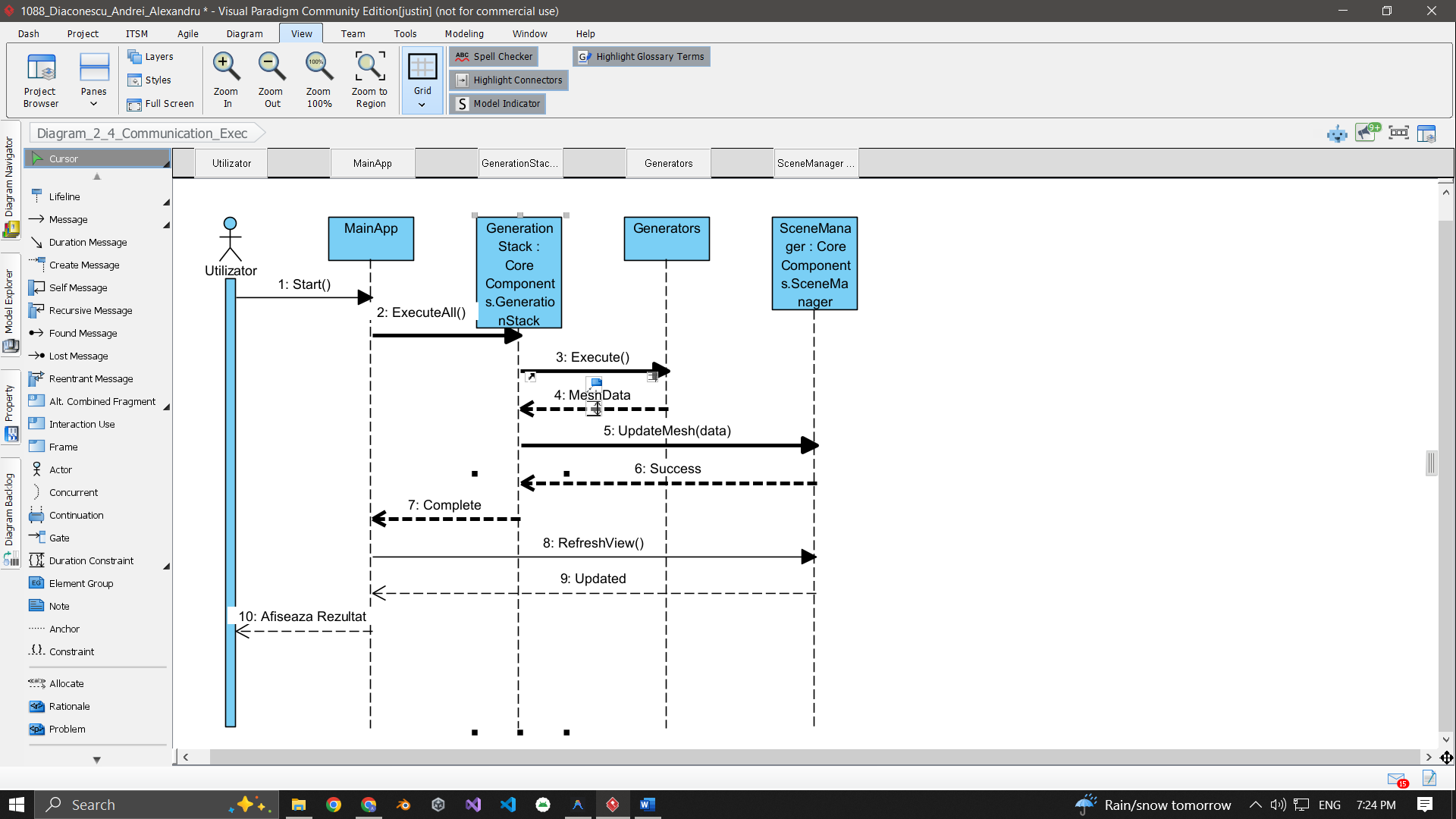
### Diagrama 2.4.2: Secvență - Execuția Pipeline-ului (Diagram\_2\_4\_Communication\_Exec)

**Participanți:**

* Utilizator
* MainApp
* GenerationStack
* Generators
* SceneManager

**Fluxul mesajelor:**

1. Utilizator → App: Start()
2. App → Stack: ExecuteAll()
3. Stack → Gens: Execute() - pentru fiecare generator
4. Gens → Stack: MeshData - rezultatul execuției
5. Stack → Scene: UpdateMesh(data)
6. Scene → Stack: Success
7. Stack → App: Complete
8. App → Scene: RefreshView()
9. Scene → App: Updated
10. App → Utilizator: Afișează Rezultat



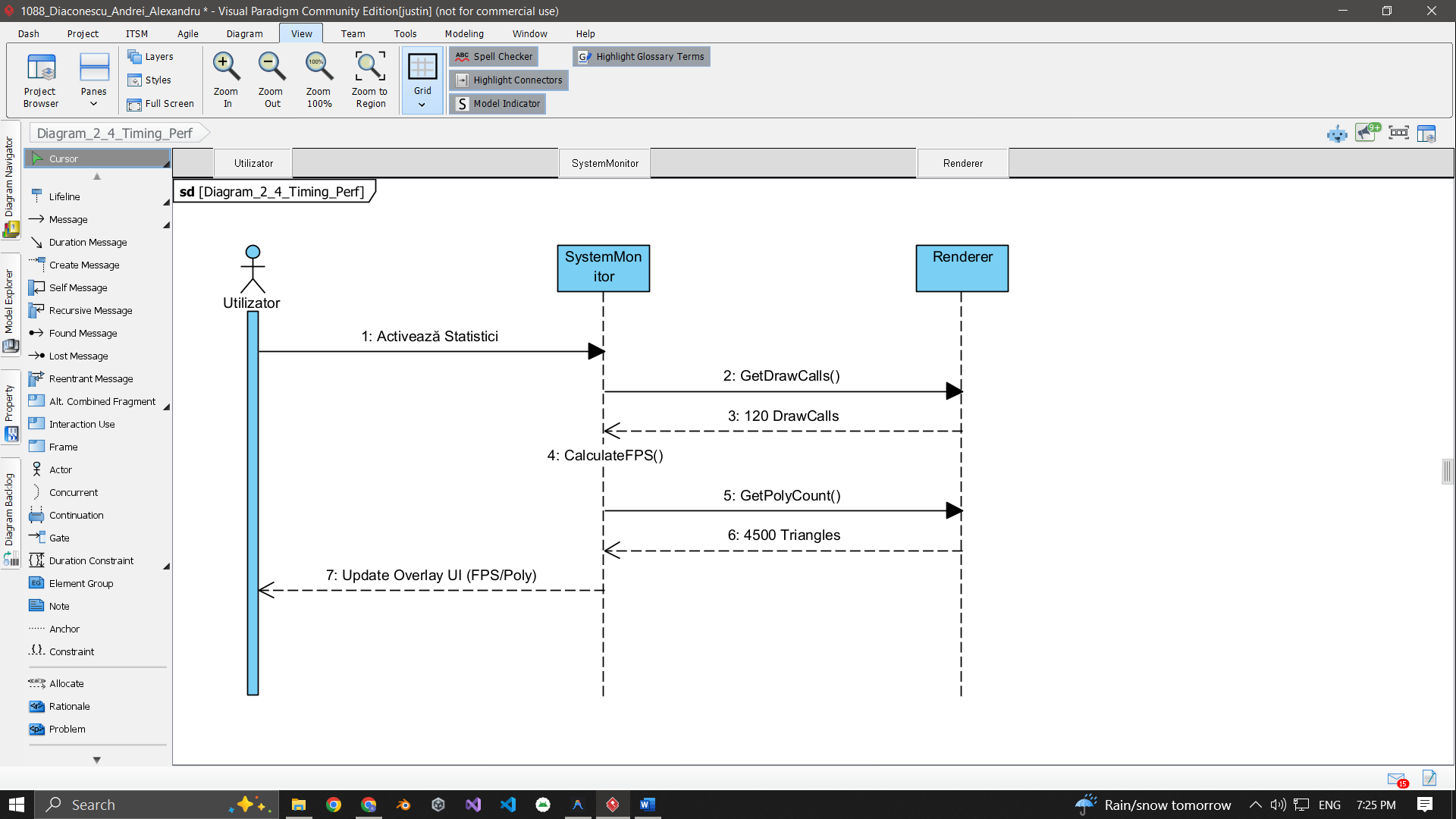
### Diagrama 2.4.3: Secvență - Monitorizare Performanță (Diagram\_2\_4\_Timing\_Perf)

**Participanți:**

* Utilizator
* SystemMonitor
* Renderer

**Fluxul mesajelor (în buclă la 60 FPS):**

1. Sys → Render: GetDrawCalls() - obține numărul de apeluri de desenare
2. Render → Sys: 120 DrawCalls (exemplu)
3. Sys → Sys: CalculateFPS() - calcul intern
4. Sys → Render: GetPolyCount() - obține numărul de poligoane
5. Render → Sys: 4500 Triangles (exemplu)
6. Sys → User: Update Overlay UI (FPS/Poly) - actualizare statistici afișate



## 2.5 Diagrame BPMN

### Diagrama de Proces BPMN (Proces\_BPMN.png)

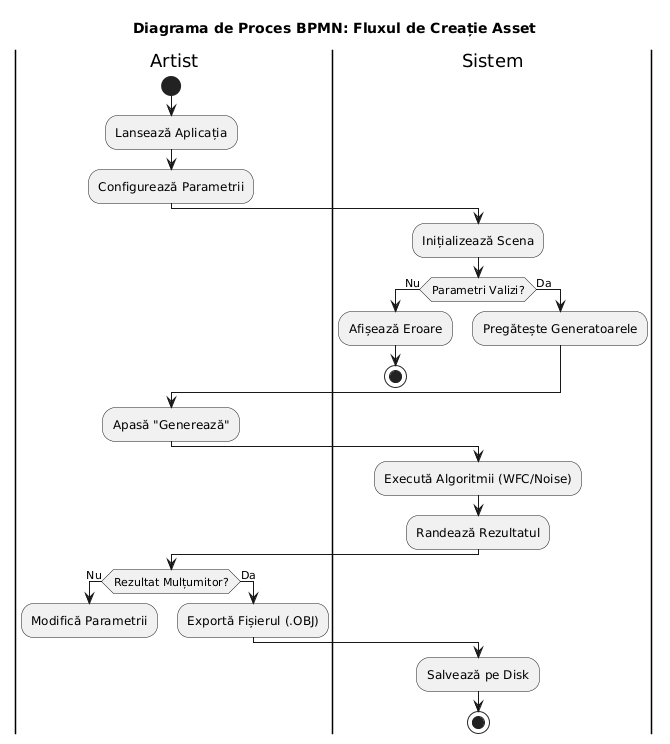
Diagrama modelează fluxul de business pentru crearea unui asset 3D procedural.

**Swimlanes (benzi):**

* Banda "Artist": Toate acțiunile care necesită intervenție umană
* Banda "Sistem": Procesele automatizate

**Gatewayuri (puncte de decizie):**

* Gateway exclusiv "Parametri valizi?": Determină dacă se poate continua
* Gateway exclusiv "Rezultat mulțumitor?": Permite iterare sau export



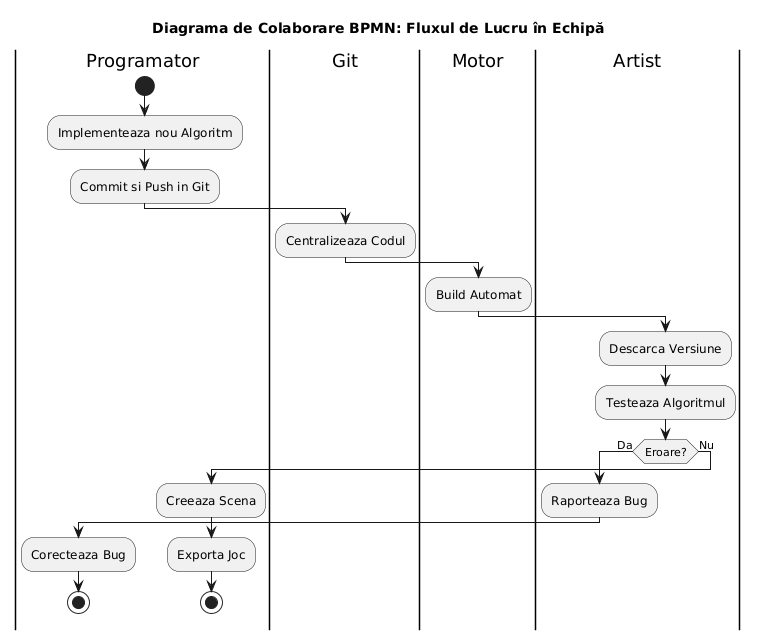
### Diagrama de Colaborare BPMN (Colaborare\_BPMN.png)

Modelează interacțiunea dintre echipa de dezvoltare.

**Pools:**

* Pool "Programator": Dezvoltă și livrează noi module
* Pool "Artist": Testează și oferă feedback

**Message Flows:** Schimb de informații între pools (raportare bug-uri, livrare DLL-uri noi)



# 3. Proiectarea Sistemului Informatic

## 3.1 Diagrama de Clase Detaliată (Diagram\_3\_2\_Detailed\_Class)

Extinde diagrama de analiză cu detalii de implementare și Design Patterns.

**Singleton Pattern:**

* **Application**: Instanță unică accesibilă global
* Constructor privat: -Application()
* Metodă statică: +GetInstance() : Application

**Factory Pattern:**

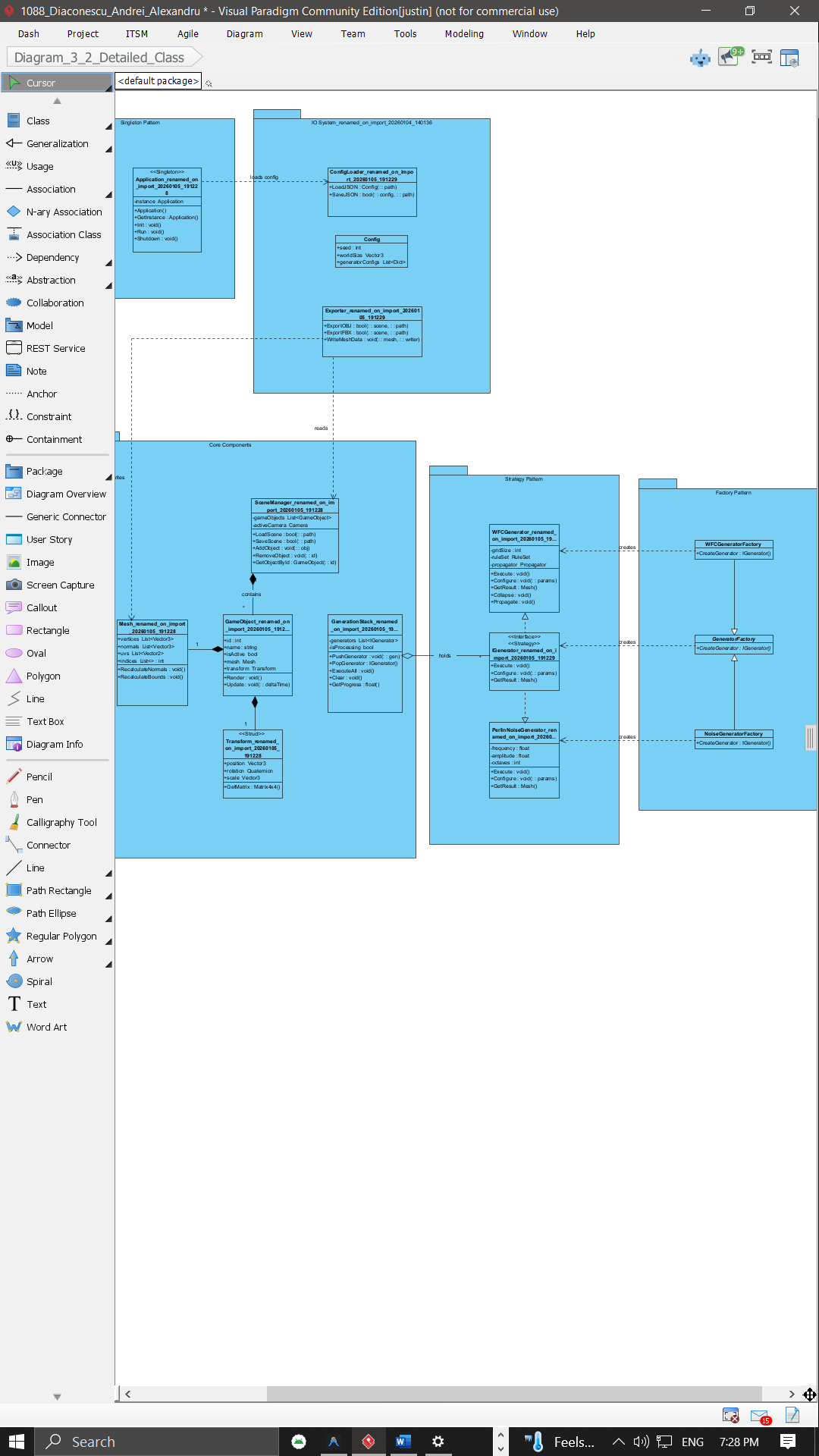
* **GeneratorFactory** (abstract): +CreateGenerator() : IGenerator
* **NoiseGeneratorFactory**: Creează instanțe PerlinNoiseGenerator
* **WFCGeneratorFactory**: Creează instanțe WFCGenerator

**Strategy Pattern:**

* **IGenerator<<Strategy>>**: Interfața abstractă
* +Execute() : void
* +Configure(params : Dict) : void
* +GetResult() : Mesh
* **PerlinNoiseGenerator**: Atribute private: frequency, amplitude, octaves
* **WFCGenerator**: Atribute private: gridSize, ruleSet, propagator
* Metode private: -Collapse(), -Propagate()

**Detalii de implementare:**

* Tipuri de date C++: List<GameObject>, Vector3, Quaternion, Matrix4x4
* Vizibilitate: + public, - private
* Metode suplimentare: Update(deltaTime), GetProgress(), RecalculateBounds()



## 3.2 Structura Bazei de Date (Diagram\_3\_2\_ERD)

Schema bazei de date SQLite pentru persistența aplicației.

**Tabela Scene:**

* id : INTEGER (Primary Key)
* name : VARCHAR(100)
* seed : INTEGER
* world\_size\_x, world\_size\_y, world\_size\_z : FLOAT
* created\_at : DATETIME

**Tabela Algorithm:**

* id : INTEGER (PK)
* name : VARCHAR(50)
* type : ENUM (Noise, WFC, Shape, Scatter)
* description : TEXT
* default\_params : JSON

**Tabela SceneAlgorithm (Junction Table - rezolvă relația M:N):**

* id : INTEGER (PK)
* scene\_id : INTEGER (FK → Scene)
* algorithm\_id : INTEGER (FK → Algorithm)
* execution\_order : INTEGER
* custom\_params : JSON

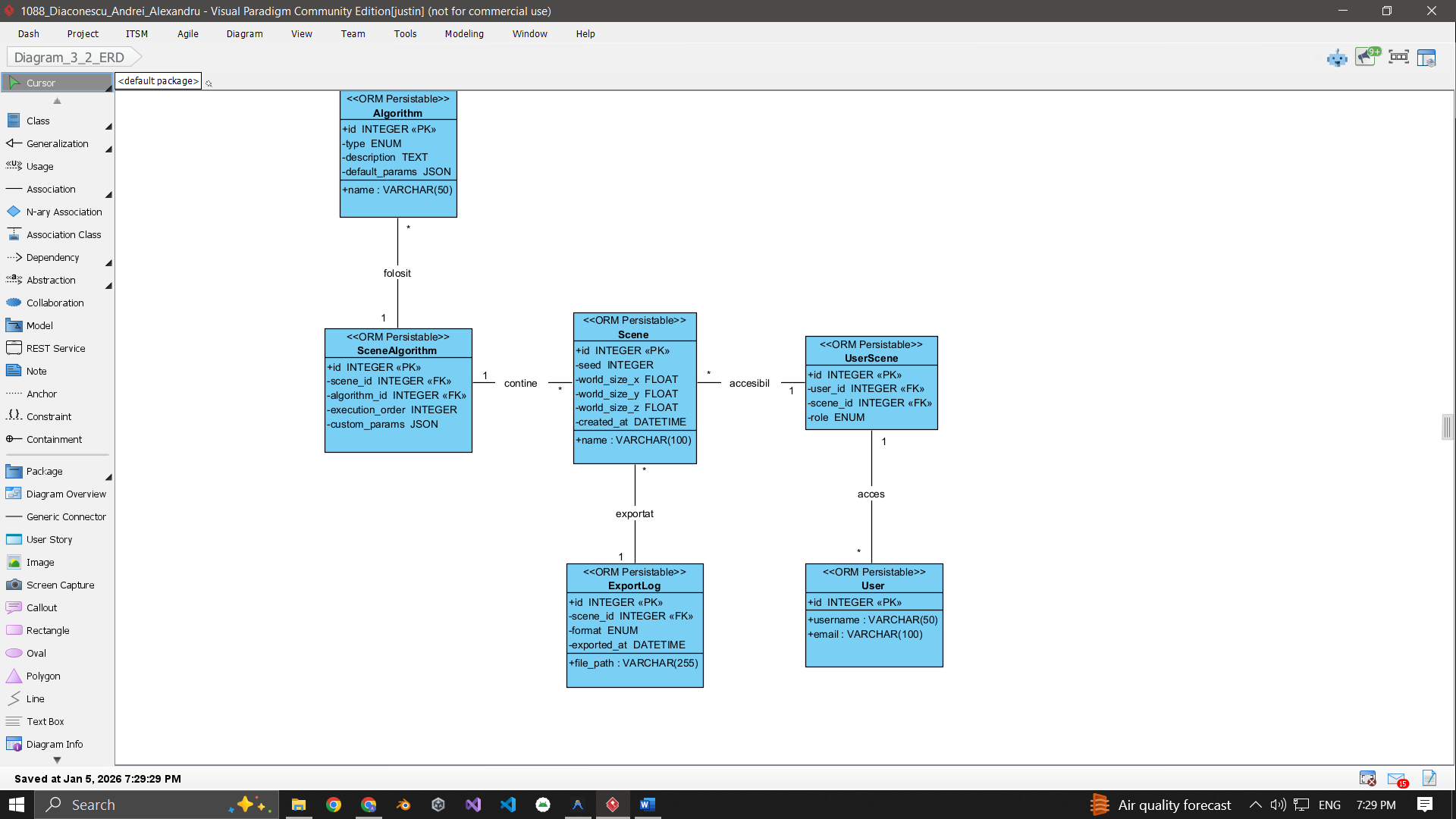
**Tabela ExportLog:**

* id : INTEGER (PK)
* scene\_id : INTEGER (FK → Scene)
* format : ENUM (OBJ, FBX)
* file\_path : VARCHAR(255)
* exported\_at : DATETIME

**Tabela User și UserScene:** Pentru management multi-utilizator opțional

**Relații:**

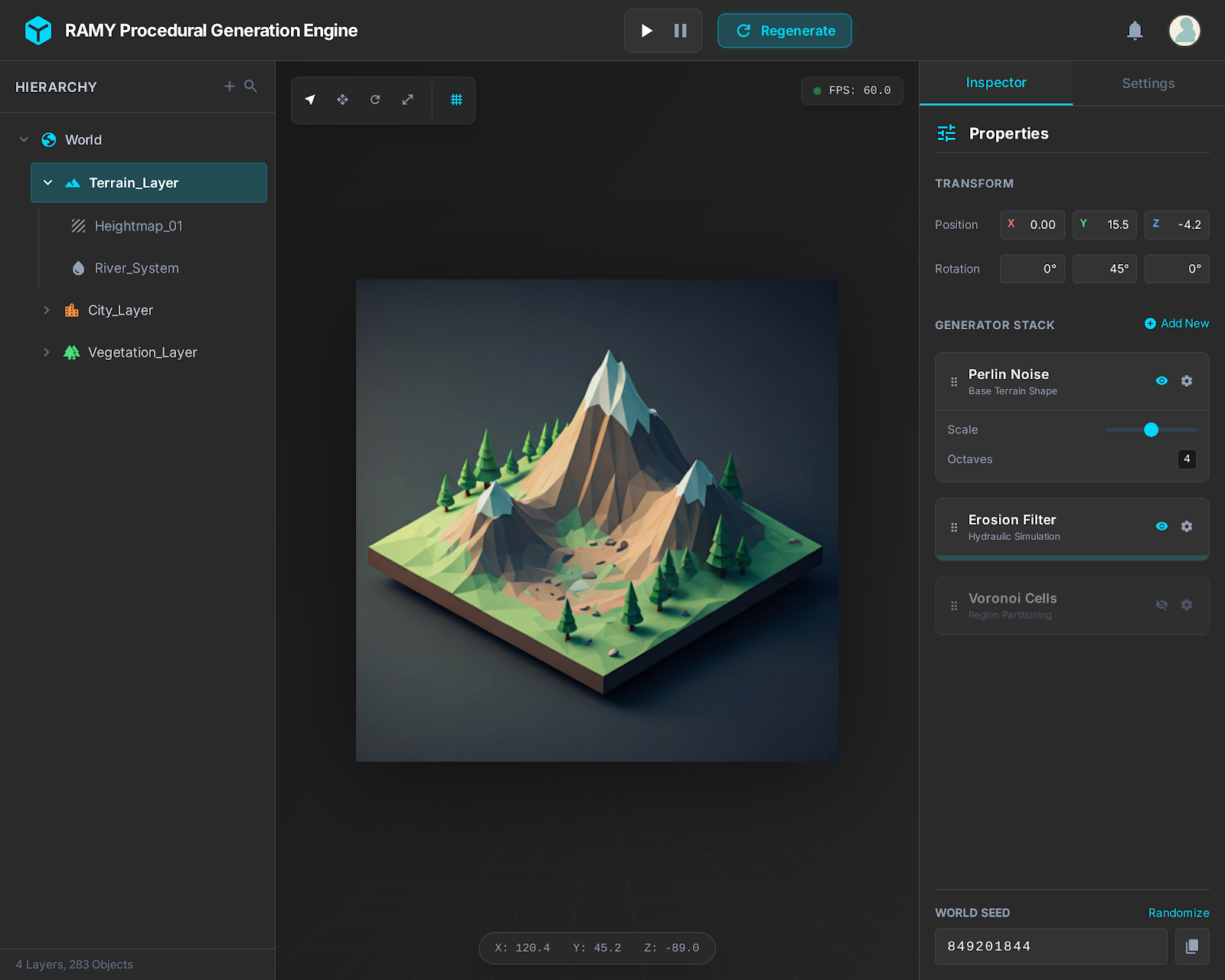
* Scene 1---\\* SceneAlgorithm (o scenă conține mai mulți algoritmi)
* Algorithm 1---\\* SceneAlgorithm (un algoritm poate fi folosit în mai multe scene)
* Scene 1---\\* ExportLog (o scenă poate avea mai multe exporturi)



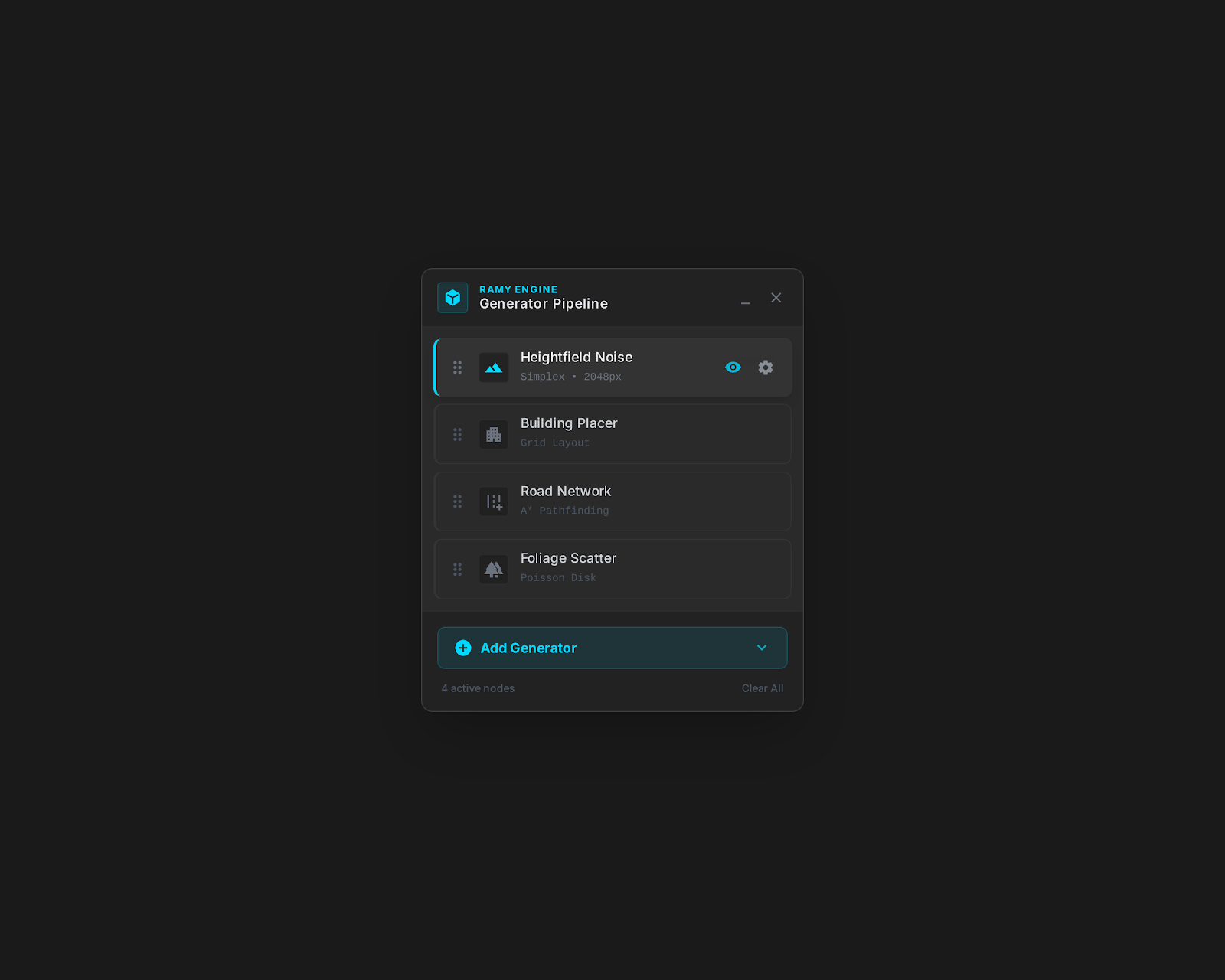
## 3.3 Proiectarea Interfețelor Utilizator

Interfața aplicației urmează principiile de design pentru unelte de productivitate, cu temă întunecată (Dark Mode) pentru a reduce oboseala vizuală în sesiuni lungi de lucru.

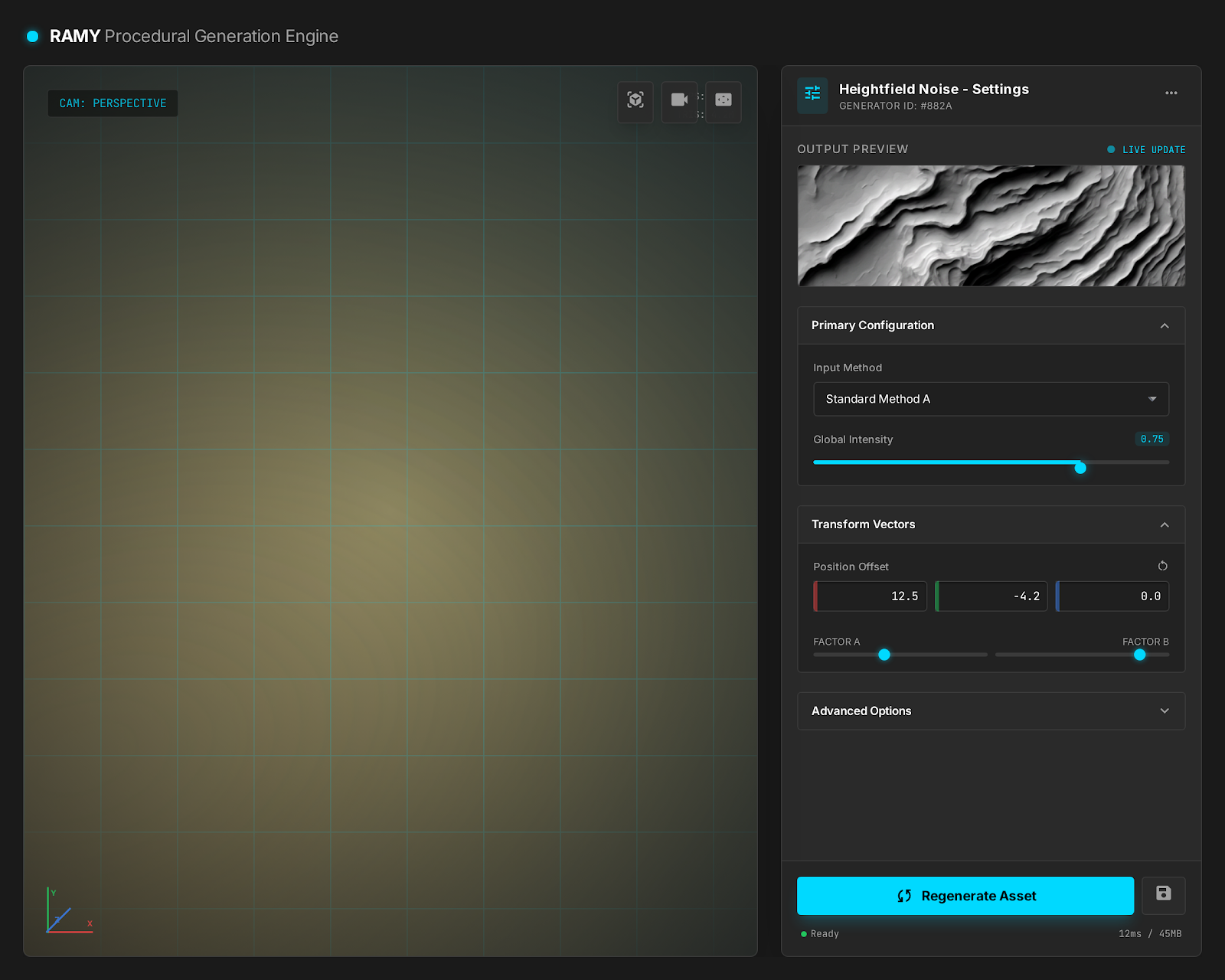
**1. Fereastra Principală (Main Editor)**

* **Viewport 3D (Centru)**: Zona principală de vizualizare a scenei generate. Include controale de navigare (Orbit, Pan, Zoom) și o grilă de referință pentru scară.
* **Toolbar Superior**: Acțiuni globale - New Project, Save, Undo/Redo, butoane Generate și Export.
* **Bara de Status (Footer)**: Afișează informații de performanță: numărul de poligoane (Polycount), FPS și utilizarea memoriei VRAM.
*  main\_editor\_workspace UI

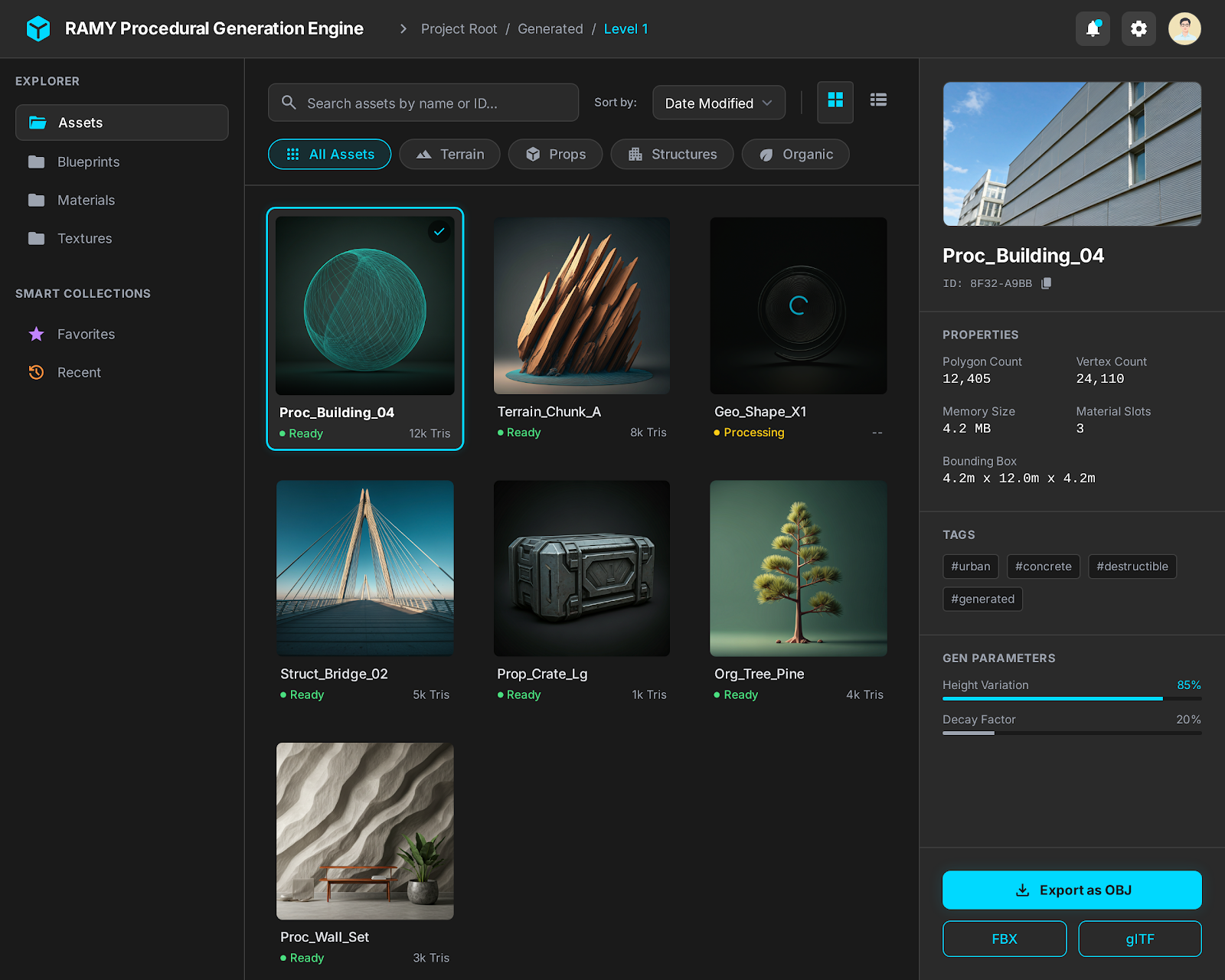
**2. Pipeline Manager (Panou Stânga)**

* Afișează stiva de generatoare sub formă de listă verticală
* Fiecare element este un "card" reprezentând un pas de generare
* Suportă drag-and-drop pentru reordonarea pașilor
* Controale per card: Checkbox (activare/dezactivare), buton Delete (X)
*  generator\_pipeline\_panel UI

**3. Inspector Panel (Panou Dreapta)**

* Panou context-sensitive care afișează parametrii generatorului selectat
* Se adaptează dinamic la tipul algoritmului (slidere pentru Noise, câmpuri text pentru reguli WFC)
* Include zonă de previzualizare (thumbnail) pentru efectul modificărilor
*  dynamic\_configuration\_panel UI

**4. Asset Explorer**

* Fereastră tip galerie pentru navigarea asset-urilor generate și salvate
* Afișare Grid View cu thumbnails
* Filtre: căutare după nume, dată, etichete
* Panou detalii: statistici (Vertices, Triangles), butoane Export și Import
*  asset\_explorer\_window UI

## 3.4 Diagrama de Componente (Diagram\_3\_4\_Component)

Arhitectura modulară organizată pe straturi.

**Presentation Layer:**

* Main Window: Interfața principală
* Inspector Panel: Configurare obiecte
* Pipeline Manager: Gestionare stivă generatoare
* Asset Explorer: Navigare fișiere

**Core Engine:**

* Scene Manager: Gestionare obiecte 3D
* Generation Stack: Execuție pipeline
* Renderer: Randare OpenGL

**Generators (plugin-uri):**

* IGenerator<<interface>>: Contract comun
* Noise Generator, WFC Generator, Shape Grammar, Scatter Generator: Implementări concrete

**IO System:**

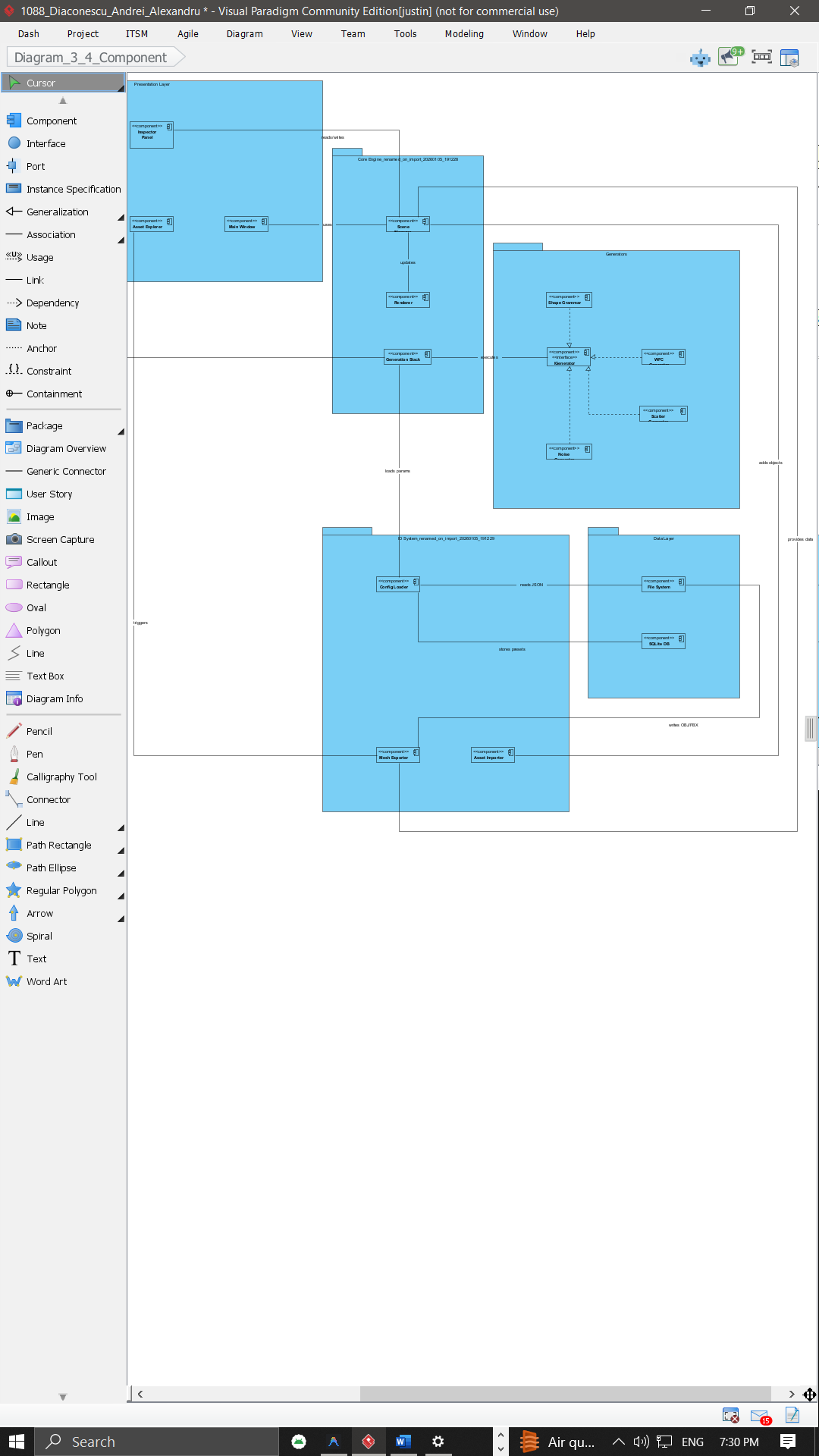
* Config Loader: Citire JSON
* Mesh Exporter: Scriere OBJ/FBX
* Asset Importer: Import resurse externe

**Data Layer:**

* SQLite DB: Preseturi și configurații
* File System: Fișiere proiect și exporturi

**Dependențe:**

* UI → Scene (uses)
* Pipeline → GenStack (manages)
* GenStack → IGenerator (executes)
* Config → FS (reads JSON)
* Export → FS (writes OBJ/FBX)



## 3.5 Diagrama de desfăşurare (Diagram\_3\_5\_Deployment)

Mediul fizic de rulare al aplicației.

**Nod "Workstation (PC Utilizator)":**

* Artefact: ProceduralGen.exe (executabilul principal)
* Artefact: GeneratorPlugins.dll (biblioteci de generatoare)
* Artefact: config.json (configurație)
* Componente: C++ Runtime (MSVC), OpenGL/Vulkan

**Nod "GPU (NVIDIA/AMD)":**

* Artefact: ComputeShaders.glsl (shadere pentru calcul paralel)
* Componentă: CUDA/OpenCL pentru accelerare

**Nod "Local Storage":**

* Folder ProjectFiles: scene.json, presets.db
* Folder Exports: \_.obj, \_.fbx

**Nod "Optional Cloud":**

* SharedAssets: Partajare între utilizatori
* BackupStorage: Copii de siguranță

**Conexiuni:**

* PC → GPU: Render Commands
* App → Storage: reads/writes
* App -.-> Cloud: syncs (optional)



# Anexe

## Lista Completă a Diagramelor

1. Diagram\_1\_UseCases - Cazuri de utilizare
2. Diagram\_2\_1\_Activity\_GenFlow - Activitate: Flux generare
3. Diagram\_2\_2\_Activity\_WFC - Activitate: Configurare WFC
4. Diagram\_2\_3\_Activity\_Export - Activitate: Export scenă
5. Diagram\_2\_2\_Class - Clase (analiză)
6. Diagram\_3\_2\_Detailed\_Class - Clase detaliate cu Design Patterns
7. Diagram\_2\_3\_State\_Scene - Stare: Ciclu scenă
8. Diagram\_2\_3\_State\_Generator - Stare: Algoritm generare
9. Diagram\_2\_4\_Sequence\_AddAlgo - Secvență: Adăugare algoritm
10. Diagram\_2\_4\_Communication\_Exec - Secvență: Execuție pipeline
11. Diagram\_2\_4\_Timing\_Perf - Secvență: Monitorizare performanță
12. Proces\_BPMN.png - BPMN Proces
13. Colaborare\_BPMN.png - BPMN Colaborare
14. Diagram\_3\_2\_ERD - Structură bază de date
15. Diagram\_3\_4\_Component - Componente sistem
16. Diagram\_3\_5\_Deployment - Deployment/Infrastructure
17. main\_editor\_workspace- Interfață: Fereastra Principală
18. generator\_pipeline\_panel- Interfață: Manager Pipeline
19. dynamic\_configuration\_panel- Interfață: Panou Configurare
20. asset\_explorer\_window- Interfață: Browser Asset-uri