Generarea Procedurală a Dungeon-urilor în Unity

Cordunianu Radu Legian Andrei Tamaș Iulia
Mujdar Milan Modreanu Maria Ibadula Adel

1 Introducere

Generarea procedurală reprezintă crearea algoritmică a conținutului (niveluri, hărți, obiecte) în mod aleator sau pseudo-aleator, evitând proiectarea manuală a fiecărui element.[1] În contextul jocurilor video, un dungeon se referă la un labirint cu camere și coridoare populate de inamici, comori și capcane. Un dungeon generat procedural este de obicei un mediu labirintic interconectat, cu provocări și recompense plasate aleator, permițând explorare liberă dar menținând un progres controlat de dificultate. [2]

Scopul acestui proiect este să implementeze mai mulți algoritmi de generare a dungeon-urilor 2D în Unity. Proiectul demonstrează trei abordări principale:

- **Simple Random Walk** – un algoritm simplu de „plimbare aleatorie" a unui agent prin spațiu, care generează un layout de tip peșteră (cu coridoare întortocheate și spații deschise)
- **Random Walk Corridor (Corridor-First)** – întâi se generează coridoare liniare prin random walk, apoi se adaugă camere la capetele sau de-a lungul acestor coridoare
- **Binary Space Partitioning (Room-First)** – se subîmparte recursiv spațiul în secțiuni (camere) și apoi camerele sunt conectate cu coridoare, asigurând un layout structurat, fără suprapuneri

Toate cele trei generatoare sunt implementate ca clase MonoBehaviour în Unity, derivând dintr-o clasă abstractă comună. Fiecare generator procedural suprascrie metoda abstractă de rulare a algoritmului, oferind propria strategie de construire a dungeon-ului. Rezultatul este redat grafic pe un Tilemap Unity, folosind un tileset pixel-art pentru podele, pereți și obiecte de decor.

Figura 1: Tileset-ul folosit în cadrul proiectului

2 Simple Random Walk

Algoritmul **Simple Random Walk** (cunoscut și ca 'drunkard's walk") folosește o celulă de start care se deplasează aleator (sus, jos, stânga, dreapta), marcând podeaua dungeon-ului pe măsură ce avansează. Rezultatul este o rețea de celule conectate, cu aspect neregulat, similar cu o peșteră. Se poate rula de mai multe ori din puncte diferite pentru a crea zone extinse. [3]

Implementare în Unity:

- Clasa *SimpleRandomWalkDungeonGenerator* implementează algoritmul.
- Parametrii precum numărul de iterații și lungimea plimbării sunt setați într-un *ScriptableObject*.
- "Celula" pornește dintr-o poziție și creează o mulțime de coordonate *(HashSet¡Vector2Int¿)* pentru podea.
- Punctul de start pentru fiecare iterație este ales aleator din zonele deja explorate.

Listing 1: Implementarea algoritmului Simple Random Walk din cadrul proiectului.

```
protected HashSet<Vector2Int> RunRandomWalk(SimpleRandomWalkSO parameters
    , Vector2Int startPosition)
{
    var currentPosition = startPosition;
    HashSet<Vector2Int> floorPositions = new HashSet<Vector2Int>();

    for (int i = 0; i < parameters.iterations; i++)
    {

        var path = ProceduralGenerationAlgorithms.SimpleRandomWalk(
            currentPosition, parameters.walkLength);


        floorPositions.UnionWith(path);

        if (parameters.startRandomlyEachIteration)
        currentPosition = floorPositions.ElementAt(Random.Range(0,
            floorPositions.Count));
    }

    return floorPositions;
}
```

Integrare în Unity

După generare:

- Se curăță tilemap-ul:

```
tilemapVisualizer.Clear();
```

- Se picteaza podeaua:

```
tilemapVisualizer.PaintFloorTiles(floorPositions);
```

- Se generează pereții:

```
WallGenerator.CreateWalls(floorPositions, tilemapVisualizer);
```

Rezultate și caracteristici:

- Structură neregulată și haotică, cu aspect natural.
- Zonele pot include camere mari și coridoare înguste.
- Flexibil și simplu de implementat.
- Lipsa unei structuri logice de camere/coridoare – potrivit pentru medii de tip cavernă.

3 Random Walk Corridor (Corridor-First)

Algoritmul **Random Walk Corridor** este o variație simplificată, specializată pe generarea de coridoare drepte. În loc să schimbe direcția la fiecare pas, acest algoritm alege o direcție fixă la început și marchează în linie o serie de celule pe acea direcție. Cu alte cuvinte, se generează un coridor de lungime predefinită orientat aleatoriu (nord, sud, est sau vest). [4] Structura rezultată are un traseu clar, iar camerele oferă spații pentru obiecte și lupte.

Implementare în Unity

Clasa folosită este CorridorFirstDungeonGenerator, care extinde SimpleRandomWalkDungeonGenerator. Are parametri specifici:

- corridorLength, corridorCount – controlează numărul și lungimea segmentelor.
- roomPercent – procent din capetele coridoarelor care vor genera camere.
- ItemPlacementHelper – plasează inamici și obiecte.

Listing 2: Implementarea algoritmului Simple Random Walk din cadrul proiectului.

```
private List<List<Vector2Int>> CreateCorridors(HashSet<Vector2Int>
    floorPositions, HashSet<Vector2Int> potentialRoomPositions)
{
    var currentPosition = startPosition;
    potentialRoomPositions.Add(currentPosition);

    List<List<Vector2Int>> corridors = new List<List<Vector2Int>>();

    for (int i = 0; i < corridorCount; i++)
    {
        var corridor = ProceduralGenerationAlgorithms.RandomWalkCorridor(
            currentPosition, corridorLength);
        corridors.Add(corridor);
        currentPosition = corridor[corridor.Count - 1];
        potentialRoomPositions.Add(currentPosition);
        floorPositions.UnionWith(corridor);
    }

    corridorPositions = new HashSet<Vector2Int>(floorPositions);
    return corridors;
}
```

Se creează o serie de coridoare drepte (aleator orientate), iar capetele lor sunt reținute pentru a deveni camere posibile.

Adăugarea camerelor

Un subset aleator din capetele de coridor devine punct de start pentru camere generate prin plimbare aleatorie (RunRandomWalk). Aceasta oferă formă neregulată camerelor:

```
var room = RunRandomWalk(randomWalkParameters, roomCenter);
```

Plasarea elementelor interactive

Obiectele și inamici sunt plasați doar în camere, nu pe coridoare, folosind:

```
itemPlacementHelper.Initialize(allRoomPositions,
    roomPositionsWithoutCorridors);

itemPlacementHelper.PlaceItems();
itemPlacementHelper.PlaceEnemies();
```

4 Partiționarea binară a spațiului (Binary Space Partitioning)

Această metodă folosește un algoritm de partiționare binară a spațiului (BSP) pentru a împărți harta în dreptunghiuri disjuncte (camere), apoi le conectează prin coridoare. Este o abordare „top-down", care oferă maxim control asupra formei și distribuției camerelor, ideală pentru dungeon-uri structurate, de tip temniță clasică. [5] [6]

Implementare în Unity

Clasa *RoomFirstDungeonGenerator* extinde *SimpleRandomWalkDungeonGenerator*. Parametri cheie:

- dungeonWidth, dungeonHeight – dimensiunile totale ale hărții.
- minRoomWidth, minRoomHeight – limite pentru camere.
- offset – spațiu liber între marginea zonei și cameră.
- randomWalkRooms – controlează dacă se generează camere dreptunghiulare sau organice.

1. Partiționarea spațiului Spațiul este împărțit recursiv în 2 zone, până când fiecare zonă este suficient de mică pentru a deveni o cameră:

```
var roomsList = ProceduralGenerationAlgorithms.BinarySpacePartitioning(
  new BoundsInt(
    Vector3Int.zero,
    new Vector3Int(dungeonWidth, dungeonHeight, 0)
  ),
  minRoomWidth,
  minRoomHeight
);
```

Rezultatul este o listă de BoundsInt, fiecare reprezentând o zonă de cameră potențială.

2. Crearea camerelor

Se creează două tipuri de camere:

- Camere dreptunghiulare simple.

```
    for (int col = offset; col < room.size.x - offset; col++)
    for (int row = offset; row < room.size.y - offset; row++)
    floor.Add(room.min + new Vector2Int(col, row));
```

- Se folosește un random walk pentru a genera camere neregulate

```
var roomCenter = Vector2Int.RoundToInt(roomBounds.center);
var roomFloor = RunRandomWalk(randomWalkParameters, roomCenter);
```

Doar tile-urile care rămân în interiorul zonei sunt păstrate (se aplică filtrare după offset).

3. Conectarea camerelor

Se aleg centrele camerelor *Vector2Int.RoundToInt(room.center)*, apoi sunt conectate între ele cu coridoare în formă de „L”:

```
CreateCorridor(pointA, pointB);
```

Aceasta construiește un traseu ortogonal: se merge vertical până la același Y, apoi orizontal până la X-ul țintă.

```
while (position.y != destination.y)
{
  position += (destination.y > position.y) ? Vector2Int.up : Vector2Int.
      down;
  corridor.Add(position);
}
while (position.x != destination.x)
{
  position += (destination.x > position.x) ? Vector2Int.right :
      Vector2Int.left;
  corridor.Add(position);
}
```

Rezultatul este un graf conex, fără cicluri, toate camerele fiind accesibile.

Caracteristici ale rezultatului

- Layout ordonat, cu camere distincte și coridoare directe.
- Permite control asupra dimensiunii și formei fiecărei camere.
- Cu *randomWalkRooms*, se obțin forme mai organice.
- Fără bucle – doar un singur drum între oricare două camere.

5 Gameplay și impactul generării procedurale

Gameplay-ul propus în acest proiect se bazează pe explorare, luptă și progresie prin spații necunoscute. Jucătorul controlează un personaj echipat cu o baghetă magică cu care poate ataca inamicii. Proiectilele lansate trebuie să atingă ținta de mai multe ori pentru a o elimina. Pe hartă sunt răspândiți inamici de tip slime, care urmăresc jucătorul odată ce intră în raza lor vizuală, forțându-l să navigheze rapid și strategic prin camere și coridoare.

Figura 2: Jucătorul împuscând multipli inamici

Pe lângă supraviețuire, jucătorul are un obiectiv clar: să găsească o cheie și ulterior să descopere ușa de ieșire, ambele amplasate aleator. Astfel, fiecare generație procedurală creează un mic puzzle de navigare și decizie.

Pe hartă apar și două tipuri de cufere, cu recompense diferite:

- Cufărul simplu: *5 monede*
- Cufărul mare: *10 monede*

Totodată, într-unul dintre cuferele mari se află ascunsă cheia, de care jucătorul are nevoie. În plus, fiecare inamic învins aduce o monedă.

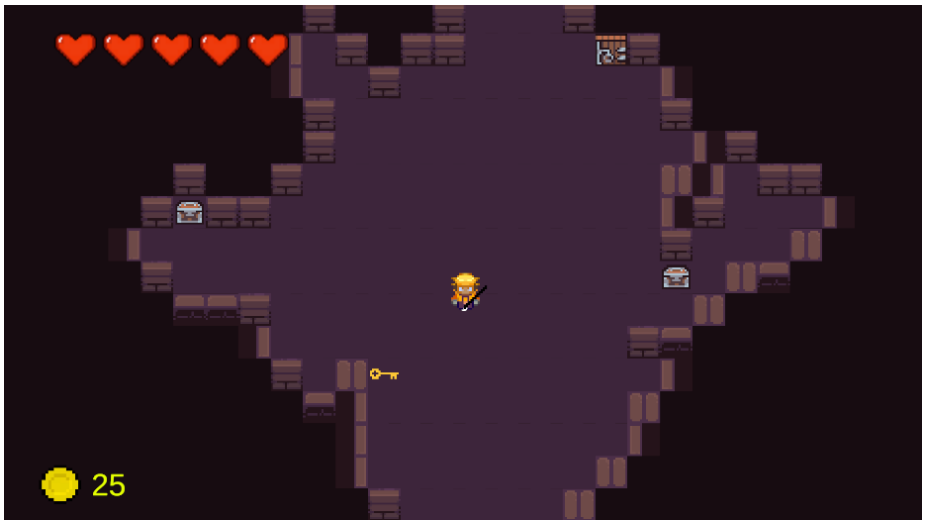

Figura 3: Jucătorul aflandu-se în aceeași cameră cu cheia și ușa

6 Concluzie

Proiectul de față demonstrează versatilitatea generării procedurale aplicate în Unity, folosind trei abordări algoritmice fundamentale: Simple Random Walk, Corridor-First și Binary Space Partitioning. Fiecare dintre acestea oferă avantaje specifice în funcție de tipul de nivel dorit, de la spații organice, la structuri semi-lineare sau dungeon-uri clar compartimentate.

Prin integrarea acestor algoritmi cu o logică de gameplay simplă, dar funcțională, proiectul reușește să creeze niveluri dinamice, navigabile și rejucabile, în care jucătorul este provocat să exploreze, să înfrunte inamici și să urmărească obiective strategice precum găsirea cheii și ieșirea din dungeon.

7 Bibliografie

Bibliografie

[1] Procedural generation, Wikipedia (ultima accesare: 12 mai 2025). Disponibil online la: https://en.wikipedia.org/wiki/Procedural_generation

[2] Constructive Generation Methods for Dungeons and Levels - Noor Shaker, Antonios Liapis, Julian Togelius, Ricardo Lopes și Rafael Bidarra. Disponibil la https://antoniosliapis.com/articles/pcgbook_dungeons.php

[3] Random Walk Cave Generation, RogueBasin Wiki (articol, 2010). Algoritm *drunkard's walk* pentru generarea de peșteri 2D. Disponibil la: https://www.roguebasin.com/index.php/Random_Walk_Cave_Generation

[4] J. H. Kaiser, Corridor path generator algorithms, Preprint, (mai 2020). Disponibil la: https://www.researchgate.net/publication/341787742

[5] Basic BSP Dungeon generation, RogueBasin Wiki (articol). Prezentare a metodei de partiționare binară a spațiului pentru dungeon-uri, inclusiv conectarea camerelor prin coridoare. Disponibil la: https://www.roguebasin.com/index.php/Basic_BSP_Dungeon_generation

[6] Dungeon generation using BSP trees,eskerda.com (blog), 2013. Explicație a algoritmului BSP; descrie împărțirea recursivă și conectarea centrului regiilor surori. Disponibil la: http://eskerda.com/bsp-dungeon-generation/

[7] A. Adonaac, Procedural Dungeon Generation Algorithm, GameDeveloper.com, 2015. Articol de blog ce detaliază algoritmul folosit în jocul *Tiny Keep* pentru generarea de dungeon-uri, oferind o perspectivă complementară asupra tehnicilor de mai sus. Disponibil la: https://www.gamedeveloper.com/programming/procedural-dungeon-generation-algorithm