Utilizarea algoritmului A* pentru pathfinding în Unity

Proiect de licență

15 mai 2025

Cuprins

1 Introducere

În cadrul acestui proiect, am dezvoltat un sistem de pathfinding pentru doi monștri (unul terestru și unul aerian) folosind algoritmul A* în motorul de joc Unity, cu ajutorul bibliotecii **A* Pathfinding Project** realizată de Aron Granberg.

2 Obiective

- Implementarea unui grid dinamic care se actualizează în funcție de poziția camerei.
- Integrarea a două entități cu comportamente diferite de deplasare: una terestră, ce urmează un path pe grid, și una aeriană, ce navighează liber.
- Persistența stării jocului prin sistem de salvare și restaurare.

3 Teorie: Algoritmul A*

Algoritmul A* (A star) este un algoritm de căutare utilizat pentru a găsi cea mai scurtă cale între două puncte. Combină avantajele lui Dijkstra (cost minim) cu ale unei euristici (de ex. distanța Manhattan sau Euclidiană).

Funcția de cost:

$$f(n) = g(n) + h(n)$$

unde:

- $g(n)$ este costul drumului de la nodul de start la nodul curent
- $h(n)$ este euristica (estimarea costului până la nodul final)

3.1 Euristica utilizată în joc

În cadrul jocului platformer 2D prezentat, s-a folosit algoritmul A* furnizat de biblioteca *Aron Granberg's A* Pathfinding Project* pentru Unity. Graful utilizat este un **GridGraph**, configurat astfel încât personajele se pot deplasa și pe direcții diagonale.

Din acest motiv, euristica folosită este **distanța Euclidiană**, potrivită în medii în care mișcarea este permisă în orice direcție. Formula acestei euristici este:

$$h(n) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Aceasta oferă o estimare realistă a distanței rămase până la țintă și este **admisibilă**, ceea ce garantează că algoritmul va găsi un drum optim. Comparativ cu alte euristici, precum Distanța Manhattan (care favorizează doar mișcări ortogonale) sau Chebyshev (care tratează toate direcțiile cu același cost maxim), distanța Euclidiană este ideală pentru spații 2D continue și jocuri cu mișcare diagonală, așa cum se observă în figura **??**. Pentru o ilustrare practică a comportamentului algoritmului A* în Unity, pot fi consultate următoarele videoclipuri incluse în arhiva lucrării:

- **Modul Scene Unity**: PathFindScene.mp4
- **Modul Simulator (joc rulând)**: PathFindSimulator.mp4

4 Biblioteca A* Pathfinding Project

Această unealtă oferă:

- Graphuri predefinite (GridGraph, PointGraph, RecastGraph)
- Suport pentru monștri zburători prin `RichAI` și `AIPath`
- Recalculare dinamică a hărții
- Editor vizual în Unity

5 Implementare în Unity

5.1 Setup inițial

- Adăugare pachet A* Pathfinding Project
- Creare obiect AstarPath cu componenta GridGraph
- Configurare `AstarPath.active.Scan()` pentru inițializarea grafului

5.2 Grid dinamic în funcție de cameră

Listing 1: Actualizarea poziției gridului în funcție de cameră

```
void LateUpdate(){ //al player-ului
    _gridManager = new GridManager(scene.transform);
    _gridManager.UpdateGrid(cameraPosition, _camera);

}


//acesta fiind GridManager-ul unde creez aria pentru scanare
namespace Project.Scripts.Game.Gameplay.Floors
{
    public class GridManager
    {
        private AstarPath _pfController;
        private GridGraph _grid;
        private GameObject _astarObject;

        public GridManager(Transform parent)
        {
            _astarObject = new GameObject("A*");
            _astarObject.transform.SetParent(parent);

            _pfController = _astarObject.AddComponent<AstarPath>();
            AstarData data = _pfController.data;
            _grid = data.AddGraph(typeof(GridGraph)) as GridGraph;
```

```
            _grid.SetDimensions(36, 20, 0.98f);
            _grid.is2D = true;
            _grid.collision.mask = LayerMask.GetMask("Ground");
            _grid.collision.use2D = true;
        }

        public void UpdateGrid(Vector3 cameraPosition, Camera camera)
        {
            Vector3 bottomLeft = camera.ViewportToWorldPoint(new Vector3(0,
            Vector3 topRight = camera.ViewportToWorldPoint(new Vector3(1, 1

            float gridWidth = topRight.x - bottomLeft.x;
            float gridHeight = topRight.y - bottomLeft.y;
            int widthNodes = Mathf.RoundToInt(gridWidth);
            int heightNodes = Mathf.RoundToInt(gridHeight);
            _astarObject.transform.position = new Vector3(Mathf.Abs(camera


            _grid.SetDimensions(widthNodes, heightNodes, 0.98f);
            _grid.center = new Vector3(cameraPosition.x, cameraPosition.y,

            _pfController.Scan();

        }

    }
}
```

5.3 Monstru terestru

- Componentă `AIPath` + `Seeker`
- Setare tag-uri walkable și obstacole
- Obiectul se deplasează folosind path generat automat

5.4 Monstru aerian

- Componentă `AIPath` + `Seeker`
- Setare tag-uri walkable și obstacole
- Obiectul se deplasează folosind path generat automat

6 Probleme întâlnite și soluții

- **Grid-ul nu se actualizează după restaurare:** Se reapelază

– **Problema:** După revenirea în joc sau schimbarea simulatorului, gridul de pathfinding nu mai funcționa corect.
Soluție: Am apelat metoda de scanare a gridului (`AstarPath.active.Scan()`) imediat după ce datele salvate au fost încărcate, asigurând regenerarea corectă a hărții de navigare.

– **Problema:** După revenirea în joc, uneori apăreau coloane în gridul generat unde AI-ul detecta în mod eronat tag-ul Ground, deși acolo nu exista nicio platformă vizibilă.
Soluție: Am ajustat condițiile de scanare și plasarea corectă a tag-urilor pentru a preveni generarea incorectă a nodurilor walkable în acele zone.

– **Problema:** Monstrul terestru nu putea sări după player dacă acesta se afla pe o platformă, deoarece lovea cu capul în ea.
Soluție: Am prelungit platforma cu obiecte invizibile în sus și am adăugat în stânga și dreapta platformei obiecte invizibile ca niște "trambuline". Atunci când playerul este pe platformă, AI-ul schimbă target-ul spre trambulina cea mai apropiată, iar la ieșirea din colliderul acesteia, se setează din nou playerul ca target.

– **Problema:** Playerul putea sta în marginea de jos a platformei, iar inamicul de deasupra platformei nu cobora din cauza calculului făcut de pathfinding de la centrul colliderului inamicului.
Soluție: Am adăugat două obiecte invizibile laterale jos (pentru coborâre), iar AI-ul alege automat cel mai apropiat dintre ele pe baza distanței față de player. După atingerea acestui obiect, target-ul revine la player.

7 Poze

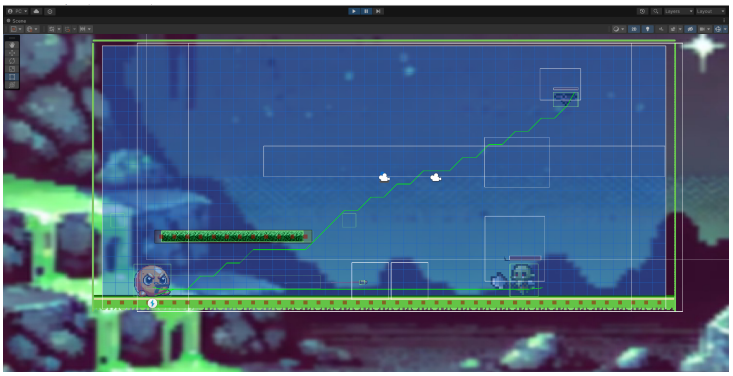

Figura 1: Demonstrație AI pathfinding

8 Concluzii

Algoritmul A* este un instrument eficient pentru pathfinding în jocuri video. Prin integrarea pachetului A* Pathfinding Project, putem modela comportamente complexe de navigare atât pentru entități terestre cât și aeriene. Problemele legate de salvare și restaurare pot fi gestionate eficient printr-un sistem bine structurat.

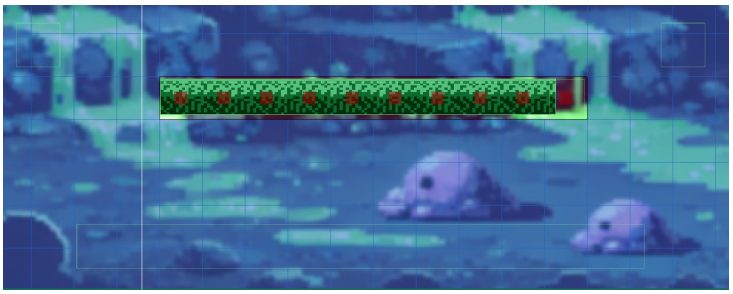

Figura 2: Demonstrație obiecte invizibile de urcare si coborare

9 Bibliografie

- Aron Granberg A* Pathfinding Project: `https://arongranberg.com/astar/`
- Algoritmul A*: `https://en.wikipedia.org/wiki/A*_search_algorithm`
- The A* Algorithm: A Complete Guide: `https://www.datacamp.com/tutorial/a-star-algorithm`
- A* Pathfinding in 2D Games: `https://shendriks.dev/posts/2024-07-13-a-star-pathf`