DreamerTheory Grid System

Overview

DreamerTheory Grid System is a robust, extensible Unity package for building grid-based games and tools in both 2D and 3D. It supports generic grid objects, world/grid conversion, rule-based tile placement, built-in pathfinding, and powerful editor tools for snapping and visualization.

Table of Contents

---

2. Package Structure

```
Packages/com.dreamertheory.gridsystem/
│
├── Runtime/
│   └── Scripts/
│       ├── Core/
│       │   └── GridSystem.cs
│       └── Ruletile/
│           └── RuleTile.cs
│
├── Editor/
│   └── Scripts/
│       └── SnapControllerEditor.cs
│
└── Documentation/
        └── README.md
```

---

4. Getting Started

Creating a Grid

1. **Add a Grid System Component:**

- Create an empty GameObject in your scene.
- Add a script inheriting from `GridSystem<TGridObject>` (see API below).
- Configure grid size, cell size, and snapping in the Inspector.

2. **Populate the Grid:**

- Use the public API to add, remove, or query objects in the grid.
- Optionally, use the RuleTile system for automated tile placement.

Example: Creating a Simple Grid

```
public class MyGrid : GridSystem3<MyTileObject>
{
        // Implement abstract methods as needed
}
public class MyGrid : GridSystem2D<MyTileObject>
{
        // Implement abstract methods as needed
}
public class MyGrid : HexGridSystem3d<MyTileObject>
{
        // Implement abstract methods as needed
}
```

---

5. Editor Tools

- **Grid Snapping:**
  Use the menu item `DT/Toggle Grid Snapping` (Ctrl+Alt+S) to enable/disable snapping for all grid systems in the scene.
- **Scene View Handles:**
  Move objects in the Scene view; if snapping is enabled, they will snap to the nearest grid cell.
- **Grid Visualization:**
  Enable `drawGizmos` and `showGridIndex` in your grid component to see grid lines and indices in the Scene view.

---

6. Rule-Based Tile Placement

- Use the `RuleTile` ScriptableObject to define tile prefabs for various neighbor configurations.
- The system automatically selects the correct prefab and rotation based on the surrounding tiles.
- Supports isolated tiles, edges, corners, inverted corners, and more.

---

6. Pathfinding

- The package includes built-in pathfinding components (A*, Dijkstra, etc.) via dedicated classes such as `AStarPathfinding` and `PathfinderAgent`.
- To calculate a path, use the API of these classes, not the grid directly.

Example

```
// Using AStarPathfinding directly
var astar = GetComponent<AStarPathfinding>();
Path path = astar.FindPath(start, end);

// Or using PathfinderAgent
var agent = GetComponent<PathfinderAgent>();
agent.SetDestination(targetPosition);
```

---

8. Public API Reference

`GridSystem<TGridObject>`

Member	Type	Description
`gridSize`	`Vector2Int`	Size of the grid (columns, rows)
`cellSize`	`float`	Size of each cell
`snap`	`bool`	Enable/disable snapping
`drawGizmos`	`bool`	Show grid in Scene view
`showGridIndex`	`bool`	Show cell indices in Scene view
`OnGridUpdated`	`UnityEvent`	Event invoked when grid changes
`GridSize`	`Vector2Int`	Property for grid size
`CellSize`	`float`	Property for cell size

Methods

- `Vector3 GetWorldPosition(int x, int y, bool center = true)`
- `Vector2Int GetGridPosition(Vector3 worldPosition)`
- `bool IsInBounds(int x, int y)`
- `void AddGridObject(TGridObject obj, int x, int y)`
- `TGridObject GetGridObject(int x, int y)`
- `void RemoveGridObject(int x, int y)`

`AStarPathfinding` / `PathfinderAgent`

- `Path FindPath(Vector2Int start, Vector2Int end)`
- `void SetDestination(Vector3 worldPosition)`

`RuleTile`

- ScriptableObject for rule-based tile placement.
- Assign prefabs for different neighbor configurations.

- Use `GetPrefabForPosition(...)` to get the correct prefab and rotation.

---

9. Extending the Grid System

- Inherit from `GridSystem<TGridObject>` to create custom grid logic.
- Override methods for custom world/grid conversion or grid behaviors.
- Extend `RuleTile` for advanced tile placement rules.

---

10. FAQ

Q: Can I use this for both 2D and 3D games?
A: Yes! The system is designed to be generic and works for both 2D and 3D workflows.

Q: How do I enable snapping?
A: Use the menu item or set the `snap` field in your grid component.

Q: How do I add custom tile rules?
A: Create a new `RuleTile` asset and assign your prefabs for each rule.