Scripts Implementation Review: What's Good Checklist

Script 1: MazeGenerator.cs

- Uses an enum `GenerateType` and a `switch` to cleanly separate three strategies (one-off, batched, pre-instantiated).
- Employs a `Dictionary` (`cellsByLocation`) and a `List` with a preset capacity for fast lookups and minimal reallocations.
- Incorporates a batched coroutine (`GenerateMazeBatched`) with yielding to avoid frame spikes, plus a pre-instantiated mode that retrieves existing cells for better performance.
- Exposes settings via `[Header]`, `[Tooltip]`, `[Range]`, and `[SerializeField]` for clear, editor-friendly configuration.
- Validates inputs and encapsulates internal state with clear property access patterns and dimension adjustments to ensure odd-sized grids.
- Organizes the scene hierarchy by parenting instantiated cells under a designated `Transform`.

Script 2: MazeGridPoolManager.cs

- Uses `[RequireComponent]` to enforce presence of `MazeGenerator` and safely retrieves it in `Awake()`.
- Reuses the `cellsByLocation` dictionary from `MazeGenerator`, avoiding redundant data structures.
- Provides bulk operations (`EnableAllCubes`/`DisableAllCubes`) that efficiently toggle all maze cubes.
- Offers granular control via `SetCubeActive`, with safe `TryGetValue` checks and clear debug logging.
- Visually simulates activation state by adjusting `Renderer.material.color` rather than expensive instantiation.

Script 3: MazeRenderCulling.cs

- Provides clear class summary and separates concerns for culling based on player distance.
- Compares squared distances to avoid costly square-root calculations.
- Runs culling at adjustable intervals via a coroutine and `updateInterval` setting.
- Toggles both `Renderer.enabled` and `Collider.enabled` for comprehensive performance benefits.
- Exposes references and settings in the Inspector for easy configuration.

Script 4: MazeDictMeshRenderer.cs

- Ensures `MazeGridPoolManager` dependency via `[RequireComponent]` and retrieves it automatically.
- Allows fine-tuned batch toggling with inspector-exposed `batchSize` and `delayBetweenBatches`.
- Implements coroutines to gradually enable/disable renderers, spreading workload over time.
- Stops ongoing coroutines before starting new ones to prevent overlapping operations.
- Invokes an `OnRenderFinished` event to signal completion, facilitating decoupled reactions.
- Safely iterates over a copied list of cells and null-checks each `Renderer`.

Script 5: GridSizeManager.cs

- Leverages Unity's Job System and Burst compiler for parallel resizing of thousands of cubes.
- Utilizes `TransformAccessArray` and `NativeArray` to minimize managed-to-native transitions.
- Disposes native collections in `OnDestroy` to prevent memory leaks.
- Exposes `cubeSize` and `spacing` in the Inspector via `[Range]` for interactive adjustments.
- Initializes transform arrays only once for efficiency on repeated resizes.
- Synchronously completes jobs (`handle.Complete()`) before applying post-resize animations.

Script 6: PrimsHelperMethods.cs

- Centralizes Prim's maze generation utilities—bounds, neighbors, carving, and animations—in one component.
- Generates dynamic direction offsets based on configurable `stepSize`.
- Implements robust neighbor detection and bounds checks to prevent errors.
- Offers multiple carving methods with input validation and visual feedback via cube coloring.
- Includes animation support (`Yanimation` and `PullDownPath`) for dynamic path visualization.
- Provides additional utilities for recoloring, resetting cubes, debugging map output, and initializing maps.

Script 7: PrimsFirstApproach.cs

- Enforces required dependencies (`MazeGridPoolManager`, `PrimsHelperMethods`) via `[RequireComponent]`.
- Outlines a clear coroutine pipeline (`GenerateAndRun`) for generating, configuring, and animating the maze.
- Supports interactive re-runs (`G` key) with state reset for rapid iteration.
- Visualizes algorithm steps with color-coded cubes and adjustable `stepDelay`.
- Validates and adjusts the starting point to ensure a valid interior cell.
- Finalizes generation by coloring unused walls, updating the maze map, and animating the path drop.

Script 8: PrimsSecondApproach.cs

- Structures the entire execution flow in `OrderOfExecution`, handling all generation modes uniformly.
- Supports automatic re-running (`G` key) without manual reinitialization.
- Encapsulates Prim's logic in `StepPrim` with clear visited/frontier management and logs.
- Enforces dependencies via `[RequireComponent]` and auto-wires references.
- Exposes `inspectorStartCell` and `stepSize` for easy algorithm parameter tweaking.
- Integrates debug logs and map prints at each stage for full visibility of the maze state.