Architectural Plan: Forest Biome Implementation (Revised)

This plan details the implementation of a new Forest biome. The design focuses on an open-layout "forest maze" and uses the existing EntityRenderer to draw all scenery (trees, rocks, bushes) as "Doom-style" billboard sprites.

This approach avoids modifying the core raycaster (FirstPersonRenderer) and instead leverages the existing sprite-rendering pipeline.

The implementation is broken into four main pillars:

1. **Generation:** Defining new "forest tiles" and updating ForestChunkGenerator to populate the Maze with wall and scenery data.
2. **Data Structure & Collision:** Creating a new Scenery class and updating Maze to store it. Critically, **updating player movement** to handle collision with *impassable scenery sprites*.
3. **Rendering:** Creating a SceneryRenderer to draw the new TREE, ROCK, and BUSH sprites, and integrating it into the main EntityRenderer.
4. **Content:** Updating SpawnManager to use existing tier data from SpawnData.java to populate the forest.

Pillar 1: Generation Logic (ForestChunkGenerator)

We will replace the placeholder ForestChunkGenerator with a fully-featured generator.

1. **Define New Tile Characters:**
   - #: **Impassable Chunk Boundary Wall.** This is the *only* character that will be recorded in the wallData bitmask.
   - .: **Passable Floor (Grass).**
   - T: **Impassable Tree.** Will become a Scenery object. wallData remains 0.
   - R: **Impassable Rock.** Will become a Scenery object. wallData remains 0.
   - B: **Passable Bush.** Will become a Scenery object. wallData remains 0.
2. **Create Forest Tile Definitions:** New String[] arrays (e.g., forestTile1) will be created in ForestChunkGenerator. These will be 12x12 and use . as the dominant character, with T and R forming the open "maze." # will *only* be used on the 4 tile edges to connect to other tiles.
3. **Implement generateChunk:**
   - This method will be a copy of MazeChunkGenerator.generateChunk.
   - It will call createMazeFromArrayTiles (copied from MazeChunkGenerator) using its own forestTile definitions.
   - **It will *not*** use corridors or mergeTileRow.
   - It will call a modified createMazeFromText (see Pillar 2).

- It will call spawnTransitionGates (copied from MazeChunkGenerator).
- It will call spawnManager.spawnEntities(2) (see Pillar 4).

Pillar 2: Data Structure & Collision

We will create a new class for scenery and update Maze and Player to support it.

1. **Create Scenery.java:**
   - Create a new class Scenery implements Renderable.
   - It will contain:

```
public enum SceneryType {
    TREE(true),
    ROCK(true),
    BUSH(false);

    public final boolean impassable;
    SceneryType(boolean impassable) { this.impassable = impassable; }
}

private final SceneryType type;
private final Vector2 position;
// ... constructor, getters, isImpassable() ...
```

2. **Update Maze.java:**
   - Add a new Map to store all scenery objects:

```
private final Map<GridPoint2, Scenery> scenery = new HashMap<>();
public Map<GridPoint2, Scenery> getScenery() { return scenery; }
public void addScenery(Scenery s) {
    scenery.put(new GridPoint2((int)s.getPosition().x, (int)s.getPosition().y), s);
}
```

   - The wallTypeData array is **no longer needed.** The Maze constructor will be updated to public Maze(int level, int[][] wallData) (and the MazeChunkGenerator call updated to match).
3. **Update ForestChunkGenerator (createMazeFromText):**
   - This method will be modified to populate wallData and scenery.
   - It will **only** create the wallData array.
   - **Loop Logic:**
     - case '#': Set the bitmask in wallData (mark as a wall).
     - case 'T': Add maze.addScenery(new Scenery(SceneryType.TREE, x, y));
     - case 'R': Add maze.addScenery(new Scenery(SceneryType.ROCK, x, y));
     - case 'B': Add maze.addScenery(new Scenery(SceneryType.BUSH, x, y));
     - case '.' (and all others): wallData remains 0 (floor).
   - Call the Maze constructor: new Maze(level, wallData);

4. **Update ChunkData.java:**
    - Add public List<SceneryData> scenery = new ArrayList<>();
    - Create public static class SceneryData { ... } (modeled after ItemData, storing SceneryType).
    - **Scrap** public int[][] wallTypeData;.
    - **ChunkData(Maze maze) (Constructor):**
        - Save maze.getScenery() into this.scenery.
    - **buildMaze() (Reconstruction):**
        - Call Maze maze = new Maze(this.level, this.wallData);
        - Loop through this.scenery and call maze.addScenery(...) to restore scenery.
5. **Update Player.java (move method):**
    - This is critical for collision. The move method must be updated to check for impassable scenery.
    - Inside move(), after calculating nextX and nextY:

```
// ...
int nextX = currentX + (int)direction.getVector().x;
int nextY = currentY + (int)direction.getVector().y;
GridPoint2 nextTile = new GridPoint2(nextX, nextY);

// ... (existing transition gate check) ...

// Check for walls (original logic)
if (maze.isWallBlocking(currentX, currentY, direction)) {
   Gdx.app.log("PlayerMovement", "Movement blocked by a wall or closed door.");
   return;
}

// NEW: Check for impassable scenery
if (maze.getScenery().containsKey(nextTile)) {
   Scenery s = maze.getScenery().get(nextTile);
   if (s.isImpassable()) {
      Gdx.app.log("PlayerMovement", "Movement blocked by impassable scenery: " +
s.getType());
      return;
   }
}

// ... (existing logic for moving through doors / normal movement) ...
```

Pillar 3: Rendering (EntityRenderer)

All rendering will be handled by the existing sprite-rendering pipeline. **No changes are**

needed to FirstPersonRenderer or RetroTheme.

1. **Create SceneryRenderer.java:**
   - Create a new class SceneryRenderer.java (modeled after ItemRenderer.java).
   - It will have a public method drawScenery(ShapeRenderer, Scenery, ...).
   - This method will contain a switch (scenery.getType()).
   - case TREE: Implement ShapeRenderer logic to draw a simple tree (e.g., brown rectangle trunk, green rectangle top).
   - case ROCK: Implement ShapeRenderer logic to draw a rock (e.g., a grey polygon).
   - case BUSH: Implement ShapeRenderer logic to draw a bush (e.g., two overlapping green rectangles).
2. **Update EntityRenderer.java:**
   - Add a SceneryRenderer: private final SceneryRenderer sceneryRenderer = new SceneryRenderer();
   - Add a list for visible scenery: private final List<Scenery> visibleScenery = new ArrayList<>();
   - **In render():**
     - Clear visibleScenery.
     - Add a new loop (after items, before monsters) to gather and sort maze.getScenery().values() into visibleScenery, then add them all to allRenderables.
   - **In renderLoop() (or equivalent):**
     - In the while loop that processes allRenderables, add the new condition:

```
if (r instanceof Scenery) {
    sceneryRenderer.drawScenery(shapeRenderer, (Scenery) r, ...);
} else if (r instanceof Item) {
// ...
```

   - This ensures all scenery objects are correctly depth-sorted and rendered as sprites.

Pillar 4: Content (SpawnManager.java)

1. **Update SpawnManager.java:**
   - Modify spawnMonster() and spawnItem() to accept a minTier parameter.
   - Inside these methods, update the logic to filter the monsterData and itemData lists (loaded from SpawnData.java) to only include entries where spawnData.tier >= minTier before making a random selection.
   - Modify the main spawnEntities() method to accept int minTier and pass it down.

```
public void spawnEntities(int minTier) {
    // ... logic ...
    spawnMonster(..., minTier);
    // ... logic ...
    spawnItem(..., minTier);
    // ... logic ...
```

```
}
```

- The existing spawnEntities() (no-arg) can be kept, calling spawnEntities(1) by default.

2. **Update ForestChunkGenerator.java:**
   - In generateChunk, update the call to:
     SpawnManager spawnManager = new SpawnManager(maze, difficulty, level, this.finalLayout);
     spawnManager.spawnEntities(2); // Start spawning Tier 2+ content

Summary of New/Modified Files:

- **ForestChunkGenerator.java:** (Replaced) New tiles, new generation logic (spawns Scenery objects).
- **Scenery.java:** (New) Data class for TREE, ROCK, BUSH with impassable flag.
- **Maze.java:** (Modified) Added scenery map. **Removed** wallTypeData. Constructor updated.
- **Player.java:** (Modified) move() method updated to check for impassable scenery.
- **ChunkData.java:** (Modified) Added serialization for scenery map. **Removed** wallTypeData.
- **SceneryRenderer.java:** (New) Renders Scenery sprites.
- **EntityRenderer.java:** (Modified) Collects and renders Scenery objects.
- **SpawnManager.java:** (Modified) spawnEntities() and helper methods updated to accept and filter by minTier.