Design Document: 3D Chunk & Dungeon Level Integration

1. The Core Problem

- **`WorldManager` is 2D:** The entire chunk system (`loadedChunks`, `loadChunk`, save files) is keyed on `GridPoint2 (x, y)`.
- **`level` is just a Parameter:** The `currentLevel` is passed *into* generators but is not part of a chunk's unique identity. `(0, 0)` on Level 1 is treated as the same chunk as `(0, 0)` on Level 2.
- **Ladders are Broken:** The `DESCEND` logic in `GameScreen` was a remnant of the pre-chunk system. It re-generates the *current* level instead of loading a *new* level.
- **`GameMode` is a Red-Herring:** The `CLASSIC` vs. `ADVANCED` mode split is now causing confusion. The *real* difference is `MAZE` biome (vertical progression) vs. Overworld biomes (horizontal progression).

2. The Solution: A 3D Chunk Coordinate System

We will refactor the core world system to use a 3D coordinate, `(x, y, z)`, to identify every chunk.

- `z` will represent the dungeon level.
- `z = 0` will be Dungeon Level 1 (the maze entrance level).
- `z = -1` will be Dungeon Level 2.
- `z = -11` will be Dungeon Level 12.
- `z = 1` will be the Overworld (Forest, Plains, etc.).

This makes every chunk in the game uniquely identifiable. `chunk_(0, 0, 0).json` is Level 1, and `chunk_(0, 0, -1).json` is Level 2.

3. New Data Structure

`ChunkID.java` (New Class)

We will create a new class to replace `GridPoint2` as the key for all chunk management. This is essential for `Map` keys and JSON serialization.

Java
```
// in core/src/main/java/com/bpm/minotaur/gamedata/
public class ChunkID {
    public int x, y, z;
```

```
    public ChunkID() {} // For JSON

    public ChunkID(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    // Must override equals() and hashCode() to be used as a Map key
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ChunkID chunkID = (ChunkID) o;
        return x == chunkID.x && y == chunkID.y && z == chunkID.z;
    }

    @Override
    public int hashCode() {
        // A simple hash combination
        int result = x;
        result = 31 * result + y;
        result = 31 * result + z;
        return result;
    }

    @Override
    public String toString() {
        return "(" + x + ", " + y + ", " + z + ")";
    }

    public ChunkID(ChunkID other) {
        this.x = other.x;
        this.y = other.y;
        this.z = other.z;
    }

    public ChunkID cpy() {
        return new ChunkID(this);
    }
}
```

4. Refactoring `WorldManager`

This is the most critical set of changes. `WorldManager.java` must be rewritten to think in 3D.

- **Fields:**
  - `private GridPoint2 currentPlayerChunkId` becomes `private ChunkID currentPlayerChunkId`.
  - `private final Map<GridPoint2, Maze> loadedChunks` becomes `private final Map<ChunkID, Maze> loadedChunks`.
  - The `currentLevel` field is now redundant and **must be removed**. The level is `currentPlayerChunkId.z`.
- **Constructor:**
  - `public WorldManager(GameMode gameMode, Difficulty difficulty, int initialLevel)` becomes `public WorldManager(GameMode gameMode, Difficulty difficulty)`.
  - It will no longer take an `initialLevel`.
  - It will initialize the player chunk ID: `this.currentPlayerChunkId = new ChunkID(0, 0, 0);` (Start at Level 1).
- **`loadChunk(GridPoint2 chunkId)` becomes `loadChunk(ChunkID chunkId)`:**
  - The `gameMode == GameMode.CLASSIC` check must be **removed**. The logic will be unified.
  - `if (loadedChunks.containsKey(chunkId))`... (This logic remains).
  - **Biome Check:** `Biome biome = biomeManager.getBiome(chunkId);` (This now gets a 3D ID).
  - **Impassable Check:** `if (biome == Biome.OCEAN)`... (This logic remains).
  - **File Naming:** The file path check must be changed:
    - `"chunk_" + chunkId.x + "_" + chunkId.y + ".json"` becomes
    - `"chunk_" + chunkId.x + "_" + chunkId.y + "_" + chunkId.z + ".json"`
  - **Generator Call:** The call to the generator must be updated:
    - `IChunkGenerator generator = generators.get(biome);`
    - `Maze newMaze = generator.generateChunk(chunkId, difficulty, gameMode);` (The `level` parameter is removed).
  - **Saving:** `saveChunk(newMaze, chunkId);` (Passes the 3D ID).
- **Other Methods:** All other methods (`getChunk`, `saveChunk`, `getLoadedChunkIds`, etc.) must be updated to use `ChunkID` instead of `GridPoint2`.

5. Refactoring `BiomeManager`

`BiomeManager.java` will be modified to use the new `ChunkID` and separate the dungeon from the overworld.

- **`getBiome(GridPoint2 chunkId)` becomes `getBiome(ChunkID chunkId)`:**

New Logic:
Java
```
public Biome getBiome(ChunkID chunkId) {
    // Rule 1: Check Z-axis. Anything at or below z=0 is the MAZE.
    if (chunkId.z <= 0) {
        // Check if we are inside the central maze shaft
        if (chunkId.x == 0 && chunkId.y == 0) {
            return Biome.MAZE;
        } else {
            // We are "underground" but *outside* the maze.
            // This could be an "Underdark" biome, but for now, it's impassable rock.
            return Biome.OCEAN; // Use OCEAN as a stand-in for "impassable"
        }
    }

    // Rule 2: We are at z=1 (the Overworld). Use 2D noise on X/Y.
    // (This preserves all our existing Forest/Plains/etc. logic)
    float noiseValue = noise.GetNoise(chunkId.x, chunkId.y);

    if (noiseValue < WorldConstants.OCEAN_THRESHOLD) {
        return Biome.OCEAN;
    } else if (noiseValue > WorldConstants.MOUNTAIN_THRESHOLD) {
        return Biome.MOUNTAINS;
    } else {
        // For now, treat everything else as Forest
        return Biome.FOREST;
    }
}
```

-
- **`Biome.java`:** We may want to add a `Biome.IMPASSABLE_ROCK` to be more explicit than `OCEAN`.

6. Refactoring Generators

- **`IChunkGenerator.java`:**
    - `generateChunk(GridPoint2 chunkId, int level, ...)` becomes `generateChunk(ChunkID chunkId, Difficulty difficulty, GameMode gameMode)`.
- **`MazeChunkGenerator.java`:**

    - Will now get the level from `chunkId.z`. It will pass this to `SpawnManager` (e.g., `int level = Math.abs(chunkId.z) + 1;`).
    - It must **NOT** spawn horizontal X/Y transition gates.
    - It **MUST** spawn one `LADDER` (type `DOWN`) at a consistent or random location.
    - If `chunkId.z < 0` (i.e., Level 2+), it **MUST** also spawn a `LADDER` (type `UP`) at the "entry" point, ideally linking to the `DOWN` ladder from the level above.
- **`ForestChunkGenerator.java`:**
    - Will continue to generate as normal. It will **NOT** spawn ladders.
    - It **WILL** spawn horizontal X/Y transition gates.
    - These gates will *only* transition in X/Y. The Z-coordinate (`z=1`) remains constant.
- **`Ladder.java`:**
    - Needs to be updated: `public enum LadderType { UP, DOWN }`
    - `public Ladder(int x, int y, LadderType type)`

7. Refactoring `GameScreen` & Player Input

`GameScreen.java` is the "glue" that handles the player's action and calls the `WorldManager`.

- **Fields:**
    - `private int currentLevel;` **is removed**.
    - `private GridPoint2 currentPlayerChunkId;` (if it exists) is **removed**.
- **`loadChunk(GridPoint2 targetChunkId)`** is renamed/refactored to **`transitionToChunk(ChunkID newChunkId, GridPoint2 newPlayerPos)`**:
    - This new method will be the single point of entry for all transitions (ladders or gates).
    - It calls `Maze newMaze = worldManager.loadChunk(newChunkId);`
    - It handles the `if (newMaze == null)` (impassable) case.
    - It updates *all* state: `this.maze = newMaze;`, `this.player.getPosition().set(newPlayerPos.x + 0.5f, newPlayerPos.y + 0.5f);`.
    - It updates the `hud` and `combatManager` with the new `maze` reference.
- **`keyDown(int keycode)`:**
    - The `DESCEND` key (`settingsManager.getKey("DESCEND")`) logic must be completely rewritten.
    - It will check if the player is facing a `Ladder`.
    - If `ladder.getType() == LadderType.DOWN`:
        - `ChunkID nextChunk = new ChunkID(worldManager.getCurrentPlayerChunkId().x, worldManager.getCurrentPlayerChunkId().y, worldManager.getCurrentPlayerChunkId().z - 1);`

        - `GridPoint2 arrivalPos = findArrivalSpawn(nextChunk, LadderType.UP);` (This new helper finds the "up" ladder in the next level).
        - `transitionToChunk(nextChunk, arrivalPos);`
    - (We should also add logic for `LadderType.UP` and an "Ascend" key).
- **`handleChunkTransition(GameEvent event)`:**
    - This method (which handles *gate* transitions) will be simplified.
    - It will get the `Gate` from the event payload.
    - `GridPoint2 targetXY = gate.getTargetChunkId();`
    - `ChunkID currentChunkId = worldManager.getCurrentPlayerChunkId();`
    - `ChunkID nextChunk = new ChunkID(targetXY.x, targetXY.y, currentChunkId.z);` (Note: `z` is preserved).
    - `GridPoint2 arrivalPos = gate.getTargetPlayerPos();`
    - `transitionToChunk(nextChunk, arrivalPos);`

This plan replaces the `(x, y)` chunk map with a `(x, y, z)` map, allowing the `MAZE` biome to exist as a vertical stack at `(0, 0, z)` while the overworld exists as a horizontal plane at `(x, y, 1)`.