Design Document: Monster AI System (Phase 1)

1. Feature Overview

This document outlines the design for the "Monster AI System (Phase 1)". The primary goal is to transition monsters from their current static state to dynamic, mobile entities within the game world. This implementation will be turn-based, with monster actions triggered only in response to a player action.

The system will introduce a new `intelligence` attribute for monsters, which will serve as the foundation for scaling AI behavior. This initial phase will focus on implementing two basic movement behaviors:

1. **Low-Intelligence:** Random, non-purposeful movement.
2. **High-Intelligence:** Basic pathfinding (A*) to seek the player.

Future phases will build on this foundation to add more complex behaviors (fleeing, special attacks, item interaction) as described in the feature request.

2. Core Architectural Changes

A. Turn-Based Action Loop

The core of the feature is that monster logic *only* updates when the player performs an action (e.g., moves, attacks, waits). We will introduce a new manager, `MonsterAiManager`, which will be called by `GameScreen` *after* any player input is successfully processed. This ensures the turn-based "I go, you go" mechanic is preserved.

B. `MonsterAiManager`

A new class, `com.bpm.minotaur.managers.MonsterAiManager`, will be created. This class will be responsible for orchestrating monster AI. It will contain a primary method, `updateMonsterLogic(Maze currentMaze, Player player)`, which will:

1. Iterate through all `Monster` objects in the `currentMaze`.
2. For each monster, query its `intelligence` attribute.
3. Based on its intelligence, execute a specific AI behavior (e.g., `performRandomMove`, `performSeekingMove`).

C. `Pathfinder` Utility

A new utility class, `com.bpm.minotaur.gamedata.Pathfinder`, will be created. This class will house a static A* (A-star) pathfinding algorithm.

- **Method:** `public static List<GridPoint2> findPath(Maze maze, GridPoint2 start, GridPoint2 goal)`
- **Function:** It will calculate the most efficient path from the monster to the player's location, navigating the `maze`'s walkable tiles and avoiding walls. The path will be returned as a list of `GridPoint2` coordinates.

3. Data Model Updates

To support the AI logic, the monster data model must be updated.

- **`assets/data/monsters.json`:** This file will be modified to include a new integer field, `"intelligence": X`, for each monster definition. A suggested scale is 0-10.
  - `0-1`: Dumb (random movement only).
  - `2-5`: Basic (A* pathfinding).
  - `6-10`: Advanced (reserved for future features like fleeing, special attacks).
- **`MonsterTemplate.java`:** Will be updated to load the `intelligence` attribute from the JSON.
- **`Monster.java`:** Will store the `intelligence` attribute from its template. It will also gain a new `AiState` enum to manage its current behavior (e.g., `WANDERING`, `SEEKING_PLAYER`), though for Phase 1, this will be simple.

4. New / Affected Classes

- **`MonsterTemplate.java` (Modified):**
  - Add `private int intelligence;`
  - Add `public int getIntelligence()`
  - Update constructor/loader to parse `intelligence` from JSON.
- **`Monster.java` (Modified):**
  - Add `private int intelligence;`
  - Add `public int getIntelligence()`
  - Update constructor to copy `intelligence` from `MonsterTemplate`.
- **`MonsterDataManager.java` (Modified):**
  - Update `parseMonsterTemplates` to read the `intelligence` field from the JSON object and pass it to the `MonsterTemplate` constructor.
- **`GameScreen.java` (Modified):**
  - Instantiate `MonsterAiManager`.

    - In the `InputProcessor` (or wherever player actions are handled), add a call to `monsterAiManager.updateMonsterLogic(currentMaze, player)` *after* the player's turn is complete.
- **`MonsterAiManager.java` (New):**
    - `public void updateMonsterLogic(Maze maze, Player player)`: Main loop.
    - `private void performRandomMove(Monster monster, Maze maze)`: Logic for `intelligence: 0`.
    - `private void performSeekingMove(Monster monster, Player player, Maze maze)`: Logic for `intelligence > 0`.
- **`Pathfinder.java` (New):**
    - `private static class Node`: Internal class for A*.
    - `public static List<GridPoint2> findPath(Maze maze, GridPoint2 start, GridPoint2 goal)`: The A* implementation.

Implementation Project Plan

Here is a step-by-step plan to implement this feature. I will request the necessary files at the beginning of each milestone.

Milestone 1: Data Model Integration

Goal: Integrate the new `intelligence` attribute into the monster data model without changing any game behavior.

Step 1: Please provide me with the latest versions of the following files:

- `core/src/main/java/com/bpm/minotaur/gamedata/monster/MonsterTemplate.java`
- `core/src/main/java/com/bpm/minotaur/gamedata/monster/Monster.java`
- `core/src/main/java/com/bpm/minotaur/gamedata/monster/MonsterDataManager.java`

Step 2: I will provide you with the updated code for these three files. The changes will:

1. Add `private int intelligence;` and a getter to `MonsterTemplate.java`.
2. Add `private int intelligence;` and a getter to `Monster.java`, and update its constructor to pull this value from its `MonsterTemplate`.

3. Update `MonsterDataManager` to read the `intelligence` value from the JSON data. If it's not present, it will default to `0`.

Step 3: After you've updated the files, you will need to **manually edit `assets/data/monsters.json`**. Please add the `"intelligence"` field to a few monsters for testing. For example:

JSON
```
{
  "id": "SKELETON",
  "name": "Skeleton",
  "family": "UNDEAD",
  "color": "WHITE",
  "health": 10,
  "damage": "1d6",
  "defense": 2,
  "intelligence": 0  // <--- ADD THIS (set to 0 for random movement)
},
{
  "id": "MINOTAUR",
  "name": "Minotaur",
  "family": "BEAST",
  "color": "RED",
  "health": 50,
  "damage": "2d8",
  "defense": 5,
  "intelligence": 3  // <--- ADD THIS (set > 0 for seeking)
}
```

Milestone 1 Check-in: Once these changes are made, **compile and run the code**. The game should run *exactly* as it did before. Monsters will still be static. This milestone is complete if the game compiles and runs without crashing.

Milestone 2: AI Manager & Random Movement

Goal: Create the `MonsterAiManager` and implement the basic AI loop. Monsters with `intelligence: 0` will now move randomly.

Step 1: I will provide the full code for the new class, `core/src/main/java/com/bpm/minotaur/managers/MonsterAiManager.java`.

- This initial version will include the `updateMonsterLogic` method.
- It will loop through `maze.getMonsters()`.
- If `monster.getIntelligence() == 0`, it will call `performRandomMove(monster, maze)`.
- The `performRandomMove` method will find a valid (walkable, unoccupied) adjacent tile and update the monster's position.

Step 2: Please provide me with the latest version of:

- `core/src/main/java/com/bpm/minotaur/screens/GameScreen.java`

Step 3: I will provide the updates for `GameScreen.java`. The changes will:

1. Add a `MonsterAiManager` field: `private MonsterAiManager monsterAiManager;`
2. Initialize this manager in the `GameScreen` constructor: `monsterAiManager = new MonsterAiManager();`
3. Find the locations in your `InputProcessor` (and any other relevant methods) where a player turn is considered "finished" (e.g., after a move, attack, or using an item).

Immediately after the player's action, I will add the line:
Java
monsterAiManager.updateMonsterLogic(currentMaze, player);

4.

Milestone 2 Check-in: Compile and run. Find a monster you designated with `"intelligence": 0` (like the Skeleton). When you take an action (e.g., move one tile), the Skeleton should move one tile in a random, valid direction. Monsters with `intelligence > 0` (like the Minotaur) should remain static.

Milestone 3: A* Pathfinding for Intelligent Monsters

Goal: Implement the A* algorithm and make monsters with `intelligence > 0` use it to hunt the player.

Step 1: I will provide the full code for the new utility class, `core/src/main/java/com/bpm/minotaur/gamedata/Pathfinder.java`. This will be a self-contained class with the A* logic and no external dependencies (other than LibGDX's `GridPoint2` and our `Maze`).

Step 2: Please provide me with the `MonsterAiManager.java` file created in the previous milestone.

Step 3: I will provide the updates for `MonsterAiManager.java`. The changes will:

1. Add the `performSeekingMove(Monster monster, Player player, Maze maze)` method.
2. Update the main `updateMonsterLogic` loop:
    - If `monster.getIntelligence() > 0`:
        1. Call `List<GridPoint2> path = Pathfinder.findPath(maze, monster.getPosition(), player.getPosition());`
        2. If a path is found (`path != null && !path.isEmpty()`), the monster will attempt to move to the first step in that path (`path.get(0)`).
        3. The logic will check if that next tile is the player's. As per your requirement, engagement rules remain the same, so if the *next step* is the player, the monster will **not** move and will instead initiate combat (or in this simple implementation, it will just stop adjacent to the player, ready for combat).

Milestone 3 Check-in: Compile and run. Monsters with `intelligence: 0` should still move randomly. Now, find a monster with `intelligence > 0` (like the Minotaur). As you move, it should take one step per turn directly toward you, navigating the maze. It should no longer be static.