ChatGPT

Verdant Defenders — Expanded Game Design Document

1. Game Overview & Goals - **Genre:** Roguelike deckbuilder - **Engine:** Godot 4.1+ - **Target Platforms:** PC Desktop (Windows/Mac/Linux) - **Core Loop:** 1. Build Deck → 2. Navigate Room Deck → 3. Resolve Rooms (Combat, Shops, Events, Trials, Treasure) → 4. Defeat Mini-Bosses → 5. Defeat Layer Boss → 6. Meta Progression - **Objective:** Detailed module breakdown and data specifications so dev team can implement systematically.

---

2. High-Level Modules 1. Data Layer 2. Move & Ability System (Enemies & Mini-Bosses) 3. Combat System 4. Room-Deck & Dungeon Flow 5. Sigil & Reward System 6. Event & Trial System 7. Shop System 8. UI / UX 9. Audio & VFX 10. Save/Load & Persistence 11. Analytics & Telemetry 12. CI / Automated Testing

---

3. Module Details

3.1 Data Layer

- **JSON Schemas** (stored in `res://data/`):
- `enemy_moves.json`: each entry `{ id, name, hp, charge_threshold, moves:[{id, name, type, value, charge_delta}], signature:{...} }`
- `mini_bosses.json`: list of 10 mini-boss templates with stats and move pools
- `layer_bosses.json`: boss phases, thresholds, moves
- `sigils.json`: full list of 20 Sigils with tier, rarity, effect_text
- `room_deck.json`: deck composition per Act (counts for Combat, Shop, Event, Trial, Treasure, Mini-Boss)
- `shop_data.json`, `event_data.json`, `treasure_data.json`
- **Loader**:
- Singleton `DataLoader.gd` to read and validate JSON into Typed Dictionaries
- On error, log detailed path & field

3.2 Move & Ability System

- **EnemyNode (extends Node2D)**:
- Properties: `max_hp`, `current_hp`, `charge`, `move_queue`, `is_mini_boss`, `is_layer_boss`
- Methods: `apply_damage()`, `apply_status()`, `gain_charge()`, `execute_move(move_id)`
- **Move Definitions**:
- Each move has `type` (Attack, Block, Status), `value`, `charge_delta`, optional `status_apply` (Poison, Chill, Burn, Vulnerable), `self_target` flag
- **Signature Trigger**:
- On `charge >= threshold`, flag next `execute_move()` as empowered signature, then `charge = 0`

3.3 Combat System

- **TurnManager.gd**:
- State machine: `PLAYER_TURN → ENEMY_TURN → END_TURN_CHECK → NEXT_ROOM`
- **PlayerController**:
- Handles Energy refill, hand draw, card play effects, Block reset
- **EnemyController**:
- Iterates through active enemies, each executes one move per turn
- **Status Effects**:
- Systems for Poison (deal at turn start), Chill (reduce damage), Vulnerable (take +50% damage), Intangible
- **CombatUI**:
- Show HP bars, Charge bars, block/energy counters

3.4 Room-Deck & Dungeon Flow

- **DungeonController.gd**:
- Loads `room_deck.json` for selected Act
- Maintains shuffled `room_deck`, `discard_pile`
- On each decision: draw 3, present choices, on selection move card to discard
- When encountering Mini-Boss card, spawn from `mini_bosses.json`
- **Room Types**:
- `CombatRoom`, `ShopRoom`, `EventRoom`, `TrialRoom`, `TreasureRoom`, `MiniBossRoom`
- Each has own `resolve()` method triggering appropriate controller

3.5 Sigil & Reward System

- **SigilController.gd**:
- On Mini-BossVictory: deck choose UI presents 3 random Sigils (from tiers based on Act)
- On TrialVictory: grant small bonus (card removal, gold, or Sigil choice option)
- **Sigil Effects**:
- Implement via `EffectRunner.gd` parsing effect_text (e.g., "On play 2 Attacks, gain 1 Energy")

3.6 Event & Trial System

- **EventController.gd**:
- Loads `event_data.json`, displays scenario text and multiple choice buttons
- On selection, triggers `EffectRunner.apply(event.choice.effect)`
- **TrialController.gd**:
- Predefined challenge conditions (e.g., no-Attack-turn, win-in-8), validate at combat end
- On success or failure, grant rewards or none

3.7 Shop System

- **ShopController.gd**:
- Displays purchasable items: cards, relic removal, gold for energy, Sigils
- Handles purchase transactions, deck modification

3.8 UI / UX

- **Scenes**:
- `Main.tscn`: root, holds CanvasLayer for UI

- `HUD.tscn`: hand bar, deck/discard icons, energy/HP/Block labels
- `RoomChoice.tscn`: displays 3 room cards face-up
- `SigilChoice.tscn`: pop-up with Sigil icons, tier colors, tooltips
- `EventDialog.tscn`, `ShopDialog.tscn`, `TrialDialog.tscn`
- **UX Flow**:
- **Run Start:** choose seed, initialize DungeonController
- **Room Selection:** choose from 3 shuffled cards
- **Resolve Room:** transition to Combat/Shop/Event
- **Victory:** show rewards UI (gold, Sigil, card removal)
- **Boss Node:** spawn Mini/Layer Boss scene
- **End Run:** summary screen (stats, logs)

3.9 Audio & VFX

- **AudioManager.gd**:
- Play SFX: card play, move execution, Charge tick, signature triggers
- Play music: Act-specific loops, victory/failure stingers
- **VFX**:
- Particle2D scenes for Charge buildup, damage bursts, Sigil pickup

3.10 Save/Load & Persistence

- **Save Format:** JSON file per run, storing deck composition, sigils, room_index, seed, gold
- **Preferences:** audio levels, resolution, seed history
- **Implementation:** `SaveManager.gd` using `FileAccess` to read/write JSON

3.11 Analytics & Telemetry

- **Event Hooks:** emit signals on room choice, Sigil pick, combat result, boss defeat, run end
- **Optional:** integrate with external analytics (e.g. Firebase) or local CSV logs

3.12 CI / Automated Testing

- **Headless Tests:** Godot unit tests for JSON loading, combat simulation
- **Scene Load Tests:** verify main scenes load without errors
- **GitHub Actions:** run `godot --headless --script tests/runner.gd` on push

---

4. Development Phases 1. **Project Setup & CI:** repo structure, JSON loader tests, initial Godot scenes 2. **Core Combat Engine:** TurnManager, PlayerController, EnemyController, status effects 3. **Data Integration:** import enemy_moves, mini_bosses, sigils, room_deck JSON 4. **Dungeon Flow & Room UI:** RoomChoice, controllers for each room type 5. **Sigil & Reward UI:** SigilChoice pop-up, EffectRunner wiring 6. **Event & Shop Modules** 7. **Layer Boss & Phase Logic** 8. **UI Polish & VFX/SFX** 9. **Balance & Playtesting** 10. **Analytics Integration & Final QA**

With this expanded GDD, your Godot dev team has a clear, step-by-step blueprint covering data schemas, systems wiring, UI flows, and test coverage to implement Verdant Defenders end-to-end.