Category 2: Game Logic & Mechanics Scripting

GTB Score: Qwen 3.6 27B — 61 | Gemma 4 31B — 89 • Winner: Gemma 4 31B (+28 pts)

Evaluation Criteria: Evaluate: Python/GDScript correctness, first-run executability, logic completeness

---

Pathfinding & Movement (Prompts 101–120)

Prompt #101

Write a grid-based A* pathfinding algorithm in Python. Handle diagonal movement, weighted terrain costs (swamp=3, road=0.5, wall=impassable), and return the path as a list of coordinates.

Prompt #102

Implement Dijkstra's algorithm in Python for a weighted directed graph representing a game world map. Include nodes for 10 cities and edges with travel costs. Return shortest path and total cost.

Prompt #103

Write a GDScript function for smooth 8-directional player movement with acceleration and deceleration curves. Maximum speed: 300px/s. Acceleration time: 0.3s. Include diagonal speed normalization.

Prompt #104

Implement a NavMesh pathfinding wrapper in Python that converts a 2D array (0=walkable, 1=wall) into a navigation mesh and returns waypoints between two points.

Prompt #105

Write Python code for a flocking behavior system (Boids algorithm) with separation, alignment, and cohesion rules. Implement for 50 boids in a 1000×1000 grid. Include predator avoidance.

Prompt #106

Implement a GDScript jump arc system with variable jump height based on hold duration. Short press: 150px. Long press: 280px. Include coyote time (0.15s) and jump buffer (0.1s).

Prompt #107

Write Python code for a tile-based random dungeon crawler movement system that handles: wall collision, door state (open/closed), trap triggers, teleport tiles, and conveyor belt tiles.

Prompt #108

Implement a Python-based steering behavior system with: seek, flee, arrive (with deceleration radius), wander (smooth angular velocity), and pursue (predicting target future position).

Prompt #109

Write GDScript for a rope physics simulation using verlet integration. Rope has 10 segments, each 20px. Include gravity, constraint resolution, and attachment point logic.

Prompt #110

Implement 3D grid pathfinding in Python for a voxel game. Handle multi-level paths (up stairs, down ramps), flying enemies that ignore terrain costs, and underwater movement (speed penalty).

Prompt #111

Write Python for a vehicle movement system with realistic turning radius, momentum, and surface traction. Surface types: asphalt (normal), ice (reduced traction), mud (speed penalty + random slip).

Prompt #112

Implement GDScript for a platformer wall-jump system. Include: wall slide (0.7x gravity), wall jump angle (70° from wall), variable momentum carry, and prevent infinite wall climbing.

Prompt #113

Write Python code for a hex-grid movement system. Include: neighbor calculation for offset coordinates, path cost calculation, terrain types, and rotation-based facing direction.

Prompt #114

Implement a GDScript grappling hook mechanic. Include: projectile launch with angle calculation, attachment detection, swing physics using pendulum equations, release momentum transfer.

Prompt #115

Write Python for a turn-based movement system on a grid with action points. Each action costs AP (move=1, run=2, climb=3). Include AP regeneration, difficult terrain penalties, and status effect modifiers.

Prompt #116

Implement a Python damage calculation system with: base damage, critical hit (2x, 5% base rate), armor reduction (flat and percentage), elemental weaknesses/resistances, and damage type enum.

Prompt #117

Write GDScript for a hitbox/hurtbox system using Area2D nodes. Include: attack hitbox activation frames, invincibility frames on hit, hit flash effect timer, and knockback vector calculation.

Prompt #118

Implement a Python turn-based combat engine with: initiative (speed stat + d6 roll), action queue, attack/defend/flee/item actions, status effects (burn, freeze, stun, poison), and enemy AI decision tree.

Prompt #119

Write Python for a projectile system with: angle calculation from shooter to target, travel speed, gravity arc for thrown weapons, piercing (hit multiple targets), bounce (reflect off walls), and homing (gradual steering).

Prompt #120

Implement a GDScript combo system for a fighting game. Track input sequence with 0.5s window. Define 5 combos. Light/heavy/special buttons. Cancel windows. Include damage scaling per combo hit.

Combat & Damage Systems (Prompts 121–140)

Prompt #121

Write Python for a shield/block system with: block value (damage reduction), stamina drain on blocked hits, perfect block (parry window: 0.15s, counters), shield break at 0 stamina, and regeneration.

Prompt #122

Implement a Python RPG stat system: STR, DEX, INT, WIS, CON, CHA. Include derived stats (HP from CON, MP from INT, speed from DEX), stat caps by class, level-up bonuses, equipment modifiers, and buff stacking.

Prompt #123

Write GDScript for a dodge roll mechanic with: invincibility frames (0.3s), roll direction from input, momentum carry into roll, stamina cost, and cooldown. Include visual feedback via color modulation.

Prompt #124

Implement a Python loot drop system. Enemy has a loot table with item name, rarity (common/uncommon/rare/legendary), and drop rate. Support: roll 1–3 items, increase rare rates based on player luck stat.

Prompt #125

Write Python for an area-of-effect damage system: circle AOE (radius in tiles), cone AOE (angle + range), line AOE (width + length). Each calculates affected entities, applies falloff damage, and checks line of sight.

Prompt #126

Implement a GDScript state machine for a melee enemy: patrol → detect player → chase → attack (3-hit combo) → recover → patrol. Include transition conditions and animation signals.

Prompt #127

Write Python for an elemental combo system: Fire + Ice = Frozen Burst (stun 2 turns), Fire + Lightning = Plasma (DoT), Ice + Earth = Avalanche (AOE slow), Water + Lightning = Shock Wave (chain damage).

Prompt #128

Implement a Python bullet-hell pattern system. Create 5 patterns: spiral, ring, cross, aimed (at player position), random burst. Parameterize bullet count, speed, and spread angle.

Prompt #129

Write GDScript for a health bar with: smooth fill animation, damage number popup, color shift (green → yellow → red based on %), overflow damage (red flash), and healing animation (green pulse).

Prompt #130

Implement a Python siege/structure damage system with: structure HP, armor rating, weak points (2x damage zones), repair mechanic, structural collapse at 20% HP (random piece destruction).

Prompt #131

Write Python for a poison/DoT system: apply effect (duration, tick interval, damage per tick), stacking rules (max 3 stacks), stack refresh vs. stack add, cleanse mechanic, and immunity after cleanse.

Prompt #132

Implement GDScript for a boss fight phase transition: Phase 1 (100%–60% HP), Phase 2 (60%–30%, adds shield + new attack), Phase 3 (<30%, doubles speed + enrages). Include transition animations and invincibility window.

Prompt #133

Write Python for a weapon durability system: durability stat, damage per use (heavy attacks wear faster), breakage at 0 (reduced damage, can't block), repair cost formula, and indestructible flag.

Prompt #134

Implement a Python magic system: mana pool, spell casting (mana cost, cast time, cooldown), spell interruption on damage, mana regeneration (per second + meditation bonus), and overcast damage (cast when empty = HP cost).

Prompt #135

Write GDScript for a stealth system: detection meter (0–100), line of sight cone (angle + range), noise radius for actions, light level influence, alert states (unaware → suspicious → alert), and alarm trigger.

Prompt #136

Implement a Python sliding tile puzzle (n×n grid). Generate solvable random states, implement move logic, detect win condition, and include a hint system using Manhattan distance to suggest the next optimal move.

Prompt #137

Write Python for a pressure plate puzzle: multiple plates, specific activation order required, timer reset if wrong order, partial progress saved, and generate a solution validator function.

Prompt #138

Implement a GDScript block-pushing puzzle (Sokoban-style). Include: undo last move (stack-based), win detection (all blocks on targets), map parsing from a 2D array, and valid move checking.

Prompt #139

Write Python for a logic gate puzzle simulation: AND, OR, NOT, XOR, NAND gates. Accept gate grid input, simulate signal flow, detect circuit cycles, and verify output matches target pattern.

Prompt #140

Implement Python for a combination lock puzzle: 4-digit code, input validation, hint system ('X digits correct, Y in right position'), attempt counter, lockout after 5 failures with 30s cooldown.

Puzzle & Logic Systems (Prompts 141–160)

Prompt #141

Write a Python implementation of a river-crossing puzzle engine (farmer, fox, chicken, grain — generalized). Accept any set of items and incompatibility rules. Find all valid solutions.

Prompt #142

Implement a GDScript mirror/light-beam puzzle: shoot beam from source, reflect off angled mirrors (45°/135°), hit target to win. Support mirrors that can be rotated by player, beam splitting at half-mirrors.

Prompt #143

Write Python for a color-mixing puzzle: primary colors (R, G, B) combine per rule set. Players mix paints in sequence to produce target colors. Include undo, hint (one correct next color), and solution check.

Prompt #144

Implement a Python cryptarithmetic puzzle solver (SEND + MORE = MONEY style). Accept equation with letter variables, return all valid digit assignments, with optional constraint for leading zeros.

Prompt #145

Write a GDScript stepper puzzle: player must reach end tile on a grid moving exactly N steps (no revisiting tiles). Generate solvable puzzles for given grid size and step count. Show solution on hint press.

Prompt #146

Implement Python for a tower-defense wave system: spawn enemy waves with escalating difficulty, support mixed enemy types per wave, wave cooldown, bonus rewards for completing wave without casualties.

Prompt #147

Write Python for a card game battle system (simplified): deck of 52 cards, deal 7 per player, turn-based play, trick-taking rules, trump suit logic, score tracking, and AI that plays highest valid card.

Prompt #148

Implement a GDScript timing puzzle: press button when moving indicator enters target zone. Variable zone size (hard=small), 3 difficulty tiers, score based on center accuracy, failure on miss.

Prompt #149

Write Python for a water flow puzzle: grid of pipes (straight, elbow, T-junction, cross), rotate pipes to connect source to drain without leaks. Validate connection graph. Show flow animation path.

Prompt #150

Implement a Python mastermind/codebreaker game: 4-color, 6-option code. Accept guesses, return black/white peg feedback, solve in minimum guesses using Knuth's 5-guess algorithm as the AI opponent.

Prompt #151

Write GDScript for an electricity/wire puzzle: connect power source to device through wire tiles. Support series circuits (all must connect), switches, fuses (one-use), and short-circuit detection.

Prompt #152

Implement Python for a weight-balance puzzle: scale with two pans, set of weights (including unknowns), series of weighing operations, deduction logic, and final guess validation.

Prompt #153

Write Python for a safe-cracking puzzle: rotate dial left/right, target sequence of 3 numbers (10–60), each number has a ±3 tolerance zone, wrong sequence resets, audio feedback (clicks) at multiples of 5.

Prompt #154

Implement a GDScript map-painting puzzle: fill all tiles with a single color using flood-fill-style moves, limited move count, color adjacency rules, and undo. Win = all tiles same color in fewest moves.

Prompt #155

Write Python for a music-box puzzle: sequence of notes (MIDI-like), player arranges note blocks in correct order, playback comparison, transposition-invariant match (same melody, different key counts as correct).

Prompt #156

Implement a Python behavior tree system from scratch: Sequence, Selector, and Leaf nodes. Build a guard AI behavior tree: patrol → detect player → chase → lost-sight → return to patrol.

Prompt #157

Write Python for a finite state machine (FSM) for a shopkeeper NPC: states = idle, greeting, transaction, haggling, farewell, suspicious. Transitions based on player proximity and dialogue flags.

Prompt #158

Implement GDScript for enemy group tactics: 3 enemy types (tank, damage, healer). Tank taunts player, damage flanks, healer stays back. Re-evaluate formation every 3 seconds.

Prompt #159

Write Python for an adaptive difficulty system: track player death count, time per room, damage taken. Adjust enemy HP, damage, and spawn count dynamically. Prevent death-loop by capping adjustments.

Prompt #160

Implement Python for a memory-based AI enemy: remembers last 5 player positions, predicts next position using linear extrapolation, sets ambush point ahead of predicted path.

AI Behavior & Procedural Generation (Prompts 161–200)

Prompt #161

Write GDScript for a stealth AI with sound propagation: enemy hears sound events (footstep, combat, door slam), assigns noise level, investigates closest recent noise if not in combat.

Prompt #162

Implement Python for a dialogue-driven quest system: NPC states that advance by conversation flags, quest journal auto-update, branching objectives, fail conditions, and reward assignment.

Prompt #163

Write Python for a resource-gathering AI (RTS style): unit evaluates nearest resource node, path to it, collect, return to base, re-evaluate if node depleted. Handle multiple simultaneous units.

Prompt #164

Implement GDScript for a boss AI with telegraphed attacks: wind-up animation → attack delay → hit window → recovery. 3 attack patterns with different telegraphs. Learn player position during wind-up.

Prompt #165

Write Python for a faction relationship system: 5 factions, reputation values (-100 to +100), relationship matrix (allied/neutral/hostile pairs), action effects on reputation, and automated war declaration at -80.

Prompt #166

Implement Python for a survival AI: NPC has hunger, thirst, fatigue, fear stats. Prioritizes needs by severity. Seeks food if hungry, water if thirsty, rests if fatigued, flees if fear > 70.

Prompt #167

Write GDScript for a competitive NPC in a racing game: rubber-band speed adjustment based on player lead/lag, blocking maneuvers, slipstream drafting, and crash recovery pathing.

Prompt #168

Implement Python for a procedural quest generator: selects from task templates (escort, fetch, defeat, collect, protect), assigns to valid NPCs based on location and faction, generates unique flavor text, and checks completion.

Prompt #169

Write Python for an emotion system for NPC agents: emotions = happy, sad, angry, fearful, surprised. Emotion changes based on events, decays toward neutral over time, influences dialogue selection.

Prompt #170

Implement GDScript for a companion AI with personality: personality traits (brave, cautious, optimistic, pessimistic). Traits influence combat decisions, item usage, dialogue triggers, and reaction to player choices.

Prompt #171

Write Python for a dungeon room generator: BSP tree splitting, minimum room size 5×5, corridors connecting rooms, guaranteed entry/exit placement, and random door/chest placement.

Prompt #172

Implement Python for Wave Function Collapse tile generation: 10×10 grid, 4 tile types with adjacency rules. Collapse from random start.

Prompt #173

Write Python for a loot table generator with rarity tiers: common (60%), uncommon (25%), rare (10%), legendary (4%), mythic (1%). Support biome modifiers that shift rarity probabilities.

Prompt #174

Implement GDScript for a random weather system: 6 weather types, transition probabilities based on current weather, duration with variance, visual/gameplay effects (rain = slippery, fog = reduced vision).

Prompt #175

Write Python for a procedural name generator: fantasy names using phoneme patterns, city names using geographic prefixes/suffixes, and NPC names by culture type (Nordic, Arabian, East Asian).

Prompt #176

Implement Python for a random event table for a survival game: 50 events, categories (good/neutral/bad), conditions for triggering (day count, player stat thresholds), cooldown to prevent repeat events.

Prompt #177

Write Python for a terrain heightmap generator using Perlin noise: 100×100 grid, normalize to 0–255, apply erosion simulation (5 passes), classify tiles by height (deep water/shore/plains/hills/mountains).

Prompt #178

Implement GDScript for a random island generator: noise-based heightmap, flood-fill ocean, place 3–7 POIs (village, cave, ruins, tower, shrine) without overlap, generate basic road network.

Prompt #179

Write Python for an encounter table: player level and biome determine encounter pool, weight by difficulty tier, prevent same encounter twice in a row, boss encounter every 10 random encounters.

Prompt #180

Implement Python for a crafting recipe system: ingredient combinations (2–4 ingredients), fuzzy matching (any wood type counts as 'wood'), quality calculation from ingredient quality average, bonus recipe discovery on perfect ingredients.

Prompt #181

Write Python for a random merchant inventory generator: pool of 50 items, daily refresh, stock size scales with settlement size, rare items appear with 5% chance, illegal items only in criminal faction settlements.

Prompt #182

Implement GDScript for a day/night cycle: 24-minute real-time day, sun position (arc), dynamic lighting color shifts (dawn orange → day white → sunset red → night blue), ambient sound swaps by time.

Prompt #183

Write Python for a population simulation: village starts with 50, grows by birth rate, shrinks by death rate (age + disaster + war), trades population with adjacent villages, triggers events at thresholds.

Prompt #184

Implement Python for a spell combination system: 8 base elements combine into 28 possible two-element spells. Generate all combinations, assign effect names, and calculate power from ingredient stats.

Prompt #185

Write GDScript for a chunk-based infinite world loader: 32×32 tile chunks, load 3×3 chunk radius around player, unload distant chunks, persist chunk state to dictionary (planted seeds, opened chests).

Prompt #186

Implement Python for a random dialogue flavor text system: NPC has archetype, location, and time-of-day inputs, generates contextually appropriate ambient conversation line from tagged template library.

Prompt #187

Write Python for a random boss ability generator: each boss gets 3–5 abilities from a pool of 20, no duplicate damage types, must include at least 1 movement ability and 1 AOE, calculate difficulty score.

Prompt #188

Implement GDScript for a procedural music system: select from mood tracks (combat, explore, mystery, sad), crossfade between states, add dynamic layers (add percussion in combat, remove melody in stealth).

Prompt #189

Write Python for a shipwreck randomizer: ship has 10 hold sections, each randomly damaged (0–3), damaged sections flood, player must seal 6 of 10 sections before sinking, randomize leak rate and seal time.

Prompt #190

Implement Python for a random puzzle generator: select puzzle type, generate solvable instance, verify solution exists using backtracking solver, calculate difficulty (steps to solution), and reject unsolvable instances.

AThinkerLab

Prompt #191

Write GDScript for a modular building generator: snap grid, prefab room tiles (bedroom, kitchen, hall, staircase), generate valid floor plans, place furniture within rooms by type rules.

Prompt #192

Implement Python for a faction war simulator: 5 factions, territory map, monthly battle resolution, territory gain/loss based on army strength + terrain bonus, diplomacy events (alliance, betrayal, peace treaty).

Prompt #193

Write Python for a relic/artifact generator: base type + modification rolls (material, enchantment, curse, history), generate descriptive name and flavor text from components, assign stat bonuses.

Prompt #194

Implement GDScript for a seasonal event calendar: 4 seasons, each 30 in-game days, seasonal events tied to dates (harvest festival, winter storm, spring bloom), NPC schedules shift by season.

Prompt #195

Write Python for a random NPC relationship web generator: 20 NPCs, assign 3–5 relationships each (friend, enemy, family, rival, lover), check for graph consistency, flag impossible combinations.

Prompt #196

Implement Python for a spell-scroll generator: base spell + scroll quality (1–5) affects range/power/duration, random discovery chance for unknown spells, degradation system (uses left), and forgery detection.

Prompt #197

Write GDScript for a city builder resource chain: wood → lumber → furniture, iron ore → ingot → tools, grain → flour → bread. Each chain has production time, worker requirement, and spoilage timer.

Prompt #198

Implement Python for a conversation memory system: NPC stores up to 10 player interactions as flags, references earlier conversations contextually, ages memory (older facts get fuzzier), and forgets if enough time passes.

Prompt #199

Write Python for a random dungeon key/lock puzzle: generate room graph, place N keys in rooms, lock N doors requiring matching keys, validate that all rooms are reachable from start with correct key order.

Prompt #200

Implement GDScript for a complete mini-game: whack-a-mole variant with 9 holes, escalating speed, miss penalty, hit combo multiplier, high score saving to a dictionary, and victory/defeat screen.