Sprint 1: Project Proposal

Procedurally Generated 2D RPG with AI-Driven Narrative

Oriol MIRÓ, Jean DIÉ, Bruno SÁNCHEZ, Dániel MÁCSAI

Master in Artificial Intelligence

Normative and Dynamic Virtual Worlds
Sprint 1: Project Proposal

October 7th, 2025

Contents

1 Introduction

Roguelike games have captivated players for decades with their promise of endless variety through procedural generation. Games like *Enter the Gungeon* or *Spelunky* demonstrate how algorithmically generated dungeons combined with intelligent enemy behaviours create highly replayable experiences.

Our project focuses on creating a 2D roguelike RPG where procedural content generation drives the core gameplay. The main challenge lies in algorithmically generating unique, playable dungeons for each playthrough while maintaining appropriate difficulty and player engagement. We will create a complete game where dungeon layouts are entirely procedurally generated, offering infinite replayability without requiring manual level design.

We will start off with a 2D roguelike tutorial, such as this one, so we can focus on the AI implementation.

2 Scenario

2.1 The Player's Journey

Our goal is to create an experience that feels alive and unrepeatable. Each run should feel like uncovering a world that has never existed before—and will never exist again. Every room, enemy encounter, and story beat is generated specifically for the current playthrough, ensuring constant novelty and replayability.

At the core of this vision lies a simple question: how can procedural generation sustain both challenge and meaning over time? We believe a *checkpoint-based roguelike loop* combines permanence with variety. Players would progress through three major dungeon regions, each culminating in a unique boss (Figure 1). Defeating a boss creates a checkpoint. Should the player fall, they return to the last checkpoint, but everything ahead is regenerated, reshaping both the level layout and narrative context.

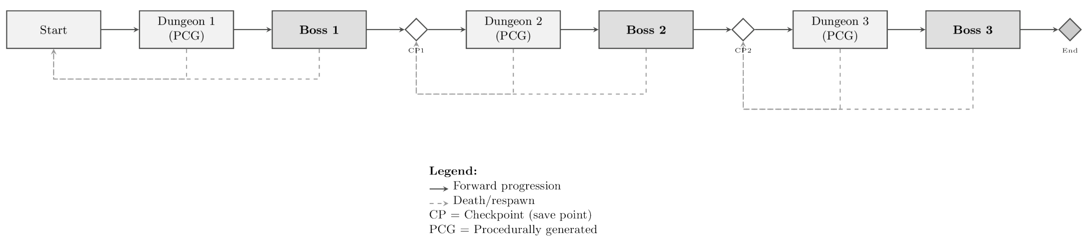

Figure 1: The player's journey through three procedurally generated dungeon sections, each culminating in a boss fight that serves as a checkpoint. Death sends the player back to their last checkpoint, where both the dungeon and narrative regenerate.

This structure naturally produces a rising difficulty curve. Early failures reset the run entirely – harsh but motivating – while later checkpoints offer meaningful progress and reduce frustration. The tension between risk and reward maintains engagement and reinforces the roguelike identity of the game.

We aim to produce a combat system akin to traditional RPGs: a health bar that depletes with each hit, encouraging players to master movement and timing. We're deliberately keeping these core mechanics straightforward to focus our development effort on the AI systems that make each encounter enjoyable rather than on complex combat mechanics.

Finally, we would like to see that aesthetic diversity supports the sense of progression. Each dungeon section should present a distinct visual and thematic identity: volcanic chambers, overgrown ruins, and crystalline caverns. This would also allow us to test different procedural generation parameters, ensuring that each stage feels mechanically and visually fresh.

2.2 Progression and Potential LLM Integration

The core progression follows a checkpoint-based system where defeating each boss creates a permanent save point. Beyond this structure, we're considering two optional features if development time permits.

A simple upgrade system could include coin collection and shops between sections, offering statistical improvements (health, damage, speed). The focus remains on getting core procedural generation working reliably.

As an experimental feature, we may explore LLM integration for dynamic narrative generation. This would involve feeding game state (current progress, bosses defeated) to an LLM to generate contextual story snippets. While technically interesting, this remains secondary to our primary goal of robust procedural dungeon generation. If implemented, we'd use a simple approach: generate multiple options, select the most coherent, and tie narrative state to checkpoints.

2.3 Game Agents

Our focus in this project is *procedural generation*; agent AI will be minimal and supportive. Following the course model, agents are specified as SENSE–THINK–ACT black boxes with simple components (navigation/physics, behaviour, animation) to keep implementation cost low.

As per the selected AI approach, for Sprint 1 we will use a compact *Finite State Machine* (FSM) per enemy (idle/patrol → investigate → chase/attack → reset). This is lightweight, debuggable, and sufficient for showcasing PCG-driven encounters; a behaviour tree could be considered later if needed.

We do not yet know what enemies we might include, but some sketches are the following:

- **Sentinel (melee grunt).** *Sense:* short-range vision cone, footstep noise. *Think:* FSM with alertness timer. *Act:* waypoint patrol, pathfind-to-player, simple swing attack. *State vars:* health, alertness, patrol_index.
- **Spitter (ranged hazard).** *Sense:* line-of-sight at medium range. *Think:* fire–cooldown cycle; strafe if player closes distance. *Act:* projectile volley, step-back, telegraphed reload. *State vars:* health, cooldown, preferred_distance.

These two templates exercise the SENSE–THINK–ACT loop, basic navigation, and animation hooks without shifting focus away from level generation and checkpointed progression. They also let us validate difficulty profiling (spawn density, sight ranges, patrol graph) within generated layouts.

3 Related work

Several commercial roguelikes and action platformers demonstrate viable patterns for combining procedural generation with authored constraints.

Spelunky popularised a hybrid, template-driven approach: hand-crafted "room chunks" are assembled into a grid with connectivity and placement rules (treasure, exits, hazards) to guarantee solvable, varied levels while retaining strong authored identity.

Enter the Gungeon drives level assembly from a *flow graph*: a directed graph of room types and connections is sampled to define progression, loops, and special rooms (shops, boss antechambers), after which rooms are instantiated and spatially embedded.

Unexplored introduced *cyclic dungeon generation*: designers specify high-level gameplay "cycles" (e.g., lock–key, shortcut, hub–spoke). The generator realises these dependencies in space to produce dungeons that feel hand-placed, with meaningful backtracking and pacing.

Dead Cells uses a hybrid of authored chunks and procedural stitching to achieve Metroidvania coherence: local constraints keep traversal rhythms and landmarks consistent while macro-scale randomisation preserves replayability.

These patterns inform our design choices: we prioritise *structure-first* generation (graphs/constraints) combined with *chunk-level variety* (templates/CA decoration) to balance reliability (connectivity, difficulty curves) and novelty (layout, encounters).

4 Proposal

4.1 AI Design

Procedural Content Generation (PCG) represents a family of algorithms that create game content algorithmically rather than manually. For 2D games, particularly roguelikes, PCG offers infinite replayability by generating unique levels, items, and encounters for each playthrough. This section surveys prominent PCG algorithms applicable to 2D game development, evaluating their strengths and use cases.

4.1.1 Space Partitioning Algorithms

Binary Space Partitioning (BSP) forms the foundation for structured level generation. The algorithm recursively subdivides rectangular space into smaller regions, places rooms within leaf nodes, and connects them with corridors. BSP excels at creating architectural layouts—dungeons, buildings, space stations—where rooms feel deliberately placed. Games like *Spelunky* use BSP variants to ensure consistent level structure while varying room contents. The algorithm guarantees connectivity and offers fine control over room density, corridor width, and structural balance.

Quadtree Partitioning extends BSP by allowing non-uniform subdivisions. Rather than always splitting spaces in half, quadtrees can create four unequal child regions based on content requirements. This produces more organic variation in room sizes while maintaining the structured feel of partitioning algorithms. Quadtrees particularly suit games requiring hierarchical detail levels—larger combat arenas connected to smaller treasure rooms.

4.1.2 Cellular Automata and Rule-Based Generation

Cellular Automata (CA) generates organic, cave-like environments through iterative local rules. Each cell examines its neighbors and transitions between states (wall/floor) based on simple thresholds. Conway's Game of Life demonstrates this principle; for dungeon generation, typical rules convert cells to walls if surrounded by many walls, creating natural-looking caverns. CA requires minimal parameters yet produces complex, irregular spaces that feel hand-sculpted. Post-processing removes disconnected regions and smooths jagged edges.

CA variants include:

- **Drunkard's Walk**: A moving agent carves paths through solid space, creating winding corridors and interconnected caves. Controlling walk length and branching probability yields varied topologies.
- **Diffusion-Limited Aggregation**: Particles randomly walk until encountering existing structure, then stick. This creates branching, tree-like level layouts resembling river deltas or root systems.

4.1.3 Graph-Based Generation

Mission Graph Generation treats levels as graphs where nodes represent rooms/encounters and edges represent connections. Designers specify high-level structure (e.g., "linear sequence with optional side branches") and the algorithm places rooms to match this topology. Graph-based PCG separates spatial layout from gameplay structure, enabling precise control over progression pacing, challenge curves, and optional content placement. *Unexplored* uses cyclic graph generation to create lock-and-key puzzles with guaranteed solutions.

Wave Function Collapse (WFC) generates levels by satisfying local constraints. Given tile patterns and adjacency rules, WFC iteratively collapses superpositions (possible tile choices) to valid configurations.

The algorithm ensures visual coherence—brick walls only connect to compatible tiles—while producing organic variety. WFC excels at generating levels that match hand-crafted aesthetic quality, though it's computationally expensive and can fail to converge without careful constraint design.

4.1.4 Grammar-Based and L-Systems

L-Systems use rewriting rules to generate complex structures from simple axioms. Originally designed for modeling plant growth, L-Systems apply to dungeon generation by treating level elements as symbols. For example, "C $\rightarrow$ C+R-C" might mean "a corridor branches into a room then continues." Successive rule applications create intricate, self-similar level topologies. L-Systems provide compact representation of complex structures and naturally create themed variations by modifying rule sets.

Grammar-Based PCG generalizes L-Systems using formal grammars. Non-terminal symbols represent abstract concepts (e.g., "combat area," "puzzle section") that expand into concrete implementations. This enables hierarchical generation: high-level narrative structure generates mid-level room sequences that finally instantiate as specific layouts. Grammar-based approaches excel at maintaining design coherence across scales.

4.1.5 Noise Functions and Heightmap-Based Generation

Perlin/Simplex Noise generates smooth, continuous random fields useful for terrain generation. Thresholding noise values creates landmasses, water bodies, and elevation changes. Multiple octaves (frequencies) of noise layered together produce naturalistic terrain with both large features and fine detail. While primarily used for outdoor/terrain generation, noise functions can define dungeon density maps—high noise values become dense wall clusters, low values open chambers.

Voronoi Diagrams partition space based on distance to seed points, creating irregular polygonal regions. Each region becomes a room or zone, with shared boundaries suggesting natural connection points. Voronoi generation produces more organic room shapes than BSP while maintaining clear spatial organization. Adjusting seed distribution (uniform, clustered, Poisson-disc) controls room size variation.

4.1.6 Hybrid Approaches

Modern PCG systems combine multiple algorithms to leverage complementary strengths:

- **BSP + CA**: BSP creates overall structure; CA decorates rooms with organic details (rubble, vegetation, water pools).
- **Graph + WFC**: Mission graphs define gameplay flow; WFC fills graph nodes with visually coherent room layouts.
- **Grammar + Noise**: Grammars generate high-level structure; noise functions add environmental variation and decorative elements.

4.1.7 Validation and Quality Assurance

Regardless of generation algorithm, PCG systems require validation to ensure playability:

- **Connectivity Analysis**: Graph algorithms (BFS/DFS) verify all areas are reachable. Unreachable regions are removed or connected.
- **Difficulty Profiling**: Simulations or heuristics estimate challenge level. Rooms with excessive enemy density or insufficient resources are rejected.
- **Solution Verification**: For puzzle-focused games, validators confirm intended solutions exist and unintended shortcuts don't trivialize challenges.

- **Aesthetic Constraints**: Pattern detection removes visually unappealing artifacts (e.g., single-tile gaps, excessively thin walls).

4.1.8 Selected Algorithms for Our Project

For our roguelike implementation, we will focus on two complementary approaches:

Cellular Automata for organic cave systems emphasizing exploration and environmental storytelling. CA's simplicity enables rapid iteration while producing naturalistic environments.

Binary Space Partitioning for structured dungeon sections emphasizing combat and tactical positioning. BSP's predictability ensures fair encounters while room variation maintains interest.

This combination provides aesthetic variety (caves vs. fortresses), supports different gameplay modes (exploration vs. combat), and demonstrates both rule-based and space-partitioning paradigms. Validation systems will ensure connectivity, appropriate difficulty progression, and fair player challenges across both generation methods.

5 Implementation Plan

5.1 Development Platform

We'll use **Unity 2D** for its mature 2D development tools, extensive asset store for graphics, and strong component-based architecture that suits modular AI implementation. This lets us focus on AI systems rather than building basic game infrastructure.

5.2 Team Roles

- **Manager:** Oriol Miró - Coordination, integration, deliverables
- **AI Designer:** Dániel Mácsai - Level design, game balance, enemy placement
- **AI Tech:** Jean Dié & Bruno Sanchez - Procedural generation algorithms, validation systems