

# Abyss Slayer

## 1. Project Overview

This project is a 2D dungeon adventure game, drawing inspiration from various dungeon crawlers, roguelikes, and action RPGs. The game focuses on strategic combat, looting mechanics, and adaptive difficulty scaling. Players explore procedurally generated dungeons, engage in combat, and enhance their abilities through a structured looting system. Players must navigate through three dungeon stages, each ending with a boss fight that increases in difficulty through adaptive attack probability and special moveset activation.

## 2. Project Review

Reference Games: Dungeon Crawlers & Roguelikes

This game builds upon mechanics found in dungeon crawlers and roguelike games while introducing distinct improvements:

- ❖ **Predefined abilities:** Instead of random weapons, players start with three core abilities: *Projectile Attack*, *AOE Damage*, and *Temporary Buffs*.
- ❖ **Stackable Ability System:** Players enhance abilities by collecting *scrolls* instead of acquiring new weapons.
- ❖ **Dynamic Boss AI:** A single boss enemy is used across all three stages but becomes stronger and more complex through probability-based move adjustments.

### **3. Programming Development**

#### **3.1 Game Concept**

The game features a dungeon-crawling combat system where players explore interconnected rooms filled with enemies, loot, and environmental challenges.

- **Player Abilities & Progression:**

- Players begin with **three core abilities**:
  1. **Projectile Attack** – Shoots a ranged attack in the aimed direction.
  2. **AOE Damage** – Deals area-based damage centered on the player.
  3. **Temporary Buffs** – Grants boosts like increased speed or damage for a limited time.
- Abilities are **enhanced by collecting scrolls**, which increase effectiveness rather than introducing new skills.

- **Combat System:**

- Players and enemies can both deal and receive damage upon contact.
- **Temporary invincibility** is granted after attacking to prevent instant counterattacks.

- **Looting System:**

- All loot is pre-generated at the start of the game to ensure fair distribution and predictable balancing.
- Chests randomly spawn at four (maybe more) predefined locations per stage.
- Each chest contains a **scroll** that enhances one of the three abilities.
- The loot drop system determines scroll rewards using a **hybrid approach**:
  1. **Weighted Random Selection** – Ensures randomness and unpredictability, providing unique experiences in every playthrough.
  2. **Dynamic Probability Scaling** – Activates after a few loot drops, adjusting drop rates based on the player's

inventory. This ensures a well-rounded power progression while preserving early-game randomness and excitement.

○

- **Boss Battles:**

- The **same boss appears in each stage**, but its **attack probability and special moves change** as the game progresses.
- The boss uses a **real-time adaptive AI system**, adjusting its moveset dynamically based on the player's behavior and stage difficulty
- At **50% and 20% health**, the boss gains **new abilities and enhanced attack patterns**, adapting its attack choices rather than following a pre-generated sequence.
- Boss moves are selected using:
  1. **Weighted Probability** – Each attack has a predefined base chance of occurring.
  2. **Adaptive AI Decision Making** – The boss reacts dynamically to the player's actions, increasing the likelihood of counterattacks or strategic shifts based on real-time inputs.

Example ideas:

- Projectile spam → Boss speeds up, gains ranged counters, or armor.
  - Slow play (low DPS) → Boss speeds up, gains healing, or forces engagements.
3. **Markov Chain Transitions (I found this method interesting)** – Ensures that the boss does not repeat the same move too frequently, leading to varied combat experiences.

## **3.2 Object-Oriented Programming Implementation**

**(\*\*\*\*\*SUBJECT TO CHANGE\*\*\*\*\*)**

<b>Class</b>	<b>Role</b>	<b>Key Attributes</b>	<b>Key Methods</b>
<b>Player</b>	Handles movement, attacking, and ability upgrades	position, health, abilities, inventory	move(), attack(), take_damage(), use_scroll()
<b>Enemy</b>	Controls enemy AI, movement, and attack patterns	position, health, attack_type	move_toward_player(), attack(), take_damage()
<b>Boss</b>	Manages boss fights, real-time adaptive difficulty, and dynamic move selection	position, health, attack_patterns, AI_state	choose_attack(), adapt_to_player(), take_damage()
<b>Projectile</b>	Controls projectile behavior and collision detection	position, direction, speed	move(), check_collision()
<b>AOEAttack</b>	Handles area-of-effect attacks centered on the player	position, radius, damage	trigger(), check_collision()
<b>BuffEffect</b>	Manages temporary boosts (speed, damage increase)	type, duration, effect_strength	apply_buff(), remove_buff()

<b>Scroll</b>	Represents ability-enhancing items obtained from chests	type, enhancement_value	apply_to_player()
<b>Chest</b>	Randomly spawns in designated areas and provides scrolls when opened	position, contains_scroll	open(), generate_loot()
<b>DungeonMap</b>	Generates and stores dungeon layout, including walls and chest locations	map_grid, enemy_spawns, chest_locations	generate_map(), place_enemies(), place_chests()
<b>GameManager</b>	Oversees game flow, enemy/boss spawns, and level transitions	current_level, player_state, enemies	start_level(), check_victory(), check_defeat()
<b>Camera</b>	Tracks player movement and adjusts the viewport dynamically	position, follow_speed	update_position()
<b>UIManager</b>	Manages on-screen elements (health bars, ability indicators, cooldown timers)	UI_elements, player_status	update_UI(), display_message()

UML Diagram (better version of implementation): [Lucidchart document](#)


### **3.3 Algorithms Involved**

The game utilizes several algorithms to improve mechanics and performance:

- **Hybrid Loot Drop System** - Uses weighted random selection early and dynamic probability scaling later.
- **Markov Chain Decision-Making** - Prevents repetitive attack patterns for bosses.

> [Markov chains](#) <

- **Real-Time Adaptive AI** - Allows the boss to react dynamically based on player behavior.
- **A\* Pathfinding** - Enables enemies to navigate towards the player while avoiding obstacles.

 Easy Pathfinding for Unity 2D and 3D Games! [Pathfinding Tuto...

- **Collision Detection** - Manages player and enemy interactions.
- **AI Behavior Systems** - Governs enemy attack decisions and movement patterns.

## **4. Statistical Data (Prop Stats)**

### **4.1 Data Features**

- **Player movement distance** – Tracks how far a player moves per stage.
- **Attack accuracy** – Measures hit vs. miss ratio for projectiles and melee attacks.
- **Stage completion time** – Records how long it takes players to clear a level.
- **Boss move frequency and adaptation rate** – Monitors the boss's reaction to player behavior.
- **Loot drop distribution** – Logs the types and frequency of looted scrolls.

## 4.2 Data Recording Method

The game will store statistical data using:

- **CSV files**
- **JSON storage**

## 4.3 Data Analysis Report

**Table 1. Data Feature Collection Details**

<b>Feature</b>	<b>Why It is Good to Have This Data? (What Can It Be Used For?)</b>	<b>How Will I Obtain 50 Values of This Feature Data?</b>	<b>Which Variable (and Class) Will You Collect This From?</b>	<b>How Will I Display This Feature Data? (Summary/Statistics or Graph)</b>
Damage Taken per Hit	Evaluates combat difficulty and player survivability by showing variation in damage received.	Record each damage value every time the player is hit during gameplay (across multiple sessions).	<code>damage_amount</code> from the <code>Player.take_damage()</code> method.	Summary statistics (e.g., average, min, max, SD) in the table; distribution graph (boxplot/histogram) to show spread.
Distance Moved per Room	Reveals player behavior (cautious exploration vs. rushing) and informs level pacing and design analysis.	Log the movement distance for each room transition (each room yields one value; 50+ rooms collected over sessions).	Calculated as <code>position_delta</code> in <code>Player.update()</code> or tracked by the <code>GameManager</code> during room transitions.	Summary statistics (e.g., average, min, max) and a line graph (time-series) or histogram displaying

				distances per room.
Player Movement Heat Map	Provides a spatial overview of where players spend the most time, helping identify hotspots, bottlenecks, and key engagement areas within the dungeon.	Continuously record the player's (x, y) coordinates at regular intervals and aggregate these into grid cell counts over sessions.	Player position (x, y) from <code>Player.update()</code> or from a dedicated tracking module in <code>GameManager</code> .	Heat map overlay on the dungeon layout.
Buff Type Preference	Indicates strategic tendencies by revealing whether players favor speed buffs or damage buffs, which is critical for balancing ability impacts.	Log each buff activation event with its specific type (one value per cast; 50+ events accumulated across sessions).	<code>buff_type</code> from the <code>Player.use_buff()</code> method.	Displayed as a pie chart showing the proportion of each buff type; summary counts/percentages may also be presented in a table.



Enemy Encounter Duration	Measures how long players remain engaged in combat per room, providing insight into encounter intensity and pacing without merely counting enemies.	Record the start and end times of enemy encounters in each room and calculate the duration for each (50+ values across sessions).	Encounter timestamps ( <code>encounter_start_time</code> and <code>encounter_end_time</code> ) in the <code>Room</code> class or <code>GameManager</code> .	Summary statistics (e.g., average, SD, median) in a table; distribution graph (histogram or line graph) to show encounter durations.
Scroll Type Collected	Tracks loot balance and fairness by recording the distribution of scroll types collected, which informs ability progression and RNG tuning.	Log each scroll collection event (one value per event; 50+ collection events over sessions).	<code>scroll_type</code> from the <code>Scroll.apply_to_player()</code> method.	Displayed as a stacked bar chart or stacked area graph showing frequency over time, with summary counts also available.

**Table 2. Data Display Plan for Proposal #4****4.3.1 Table Display: Summary Statistics****(A) Features for the Table:**

- Damage Taken per Hit
- Distance Moved per Room
- Enemy Encounter Duration

**(B) Statistical Values to Present:**

- Damage Taken per Hit: Average, Minimum, Maximum, Standard Deviation (SD)
- Distance Moved per Room: Average, Minimum, Maximum
- Enemy Encounter Duration: Average, Standard Deviation, Median (or a chosen percentile)

**4.3.2 Graph Display: Visualizing Key Features**

Graph #	Feature	Graph Type	Graph Objective	X-axis	Y-axis
1	Damage Taken per Hit	Boxplot or Histogram	Display the distribution of damage values to evaluate combat difficulty, variability, and outliers.	Hit Index or Session	Damage Value (HP lost)
2	Buff Type Preference	Pie Chart	Show the proportion of each buff type used to reveal player strategic preferences (e.g., speed vs. damage).	<i>(Not applicable for pie charts)</i>	Proportion (% usage per buff type)
3	Distance Moved per Room	Line Graph	Illustrate trends in player movement across rooms to assess exploration	Room Number (sequential order)	Distance Moved (game units)

			patterns and identify pacing issues.		
--	--	--	---	--	--

Each graph is designed to address a specific gameplay aspect:

- **Graph 1:** Helps in understanding combat intensity and identifying any damage spikes or balance issues.
- **Graph 2:** Provides insights into how players utilize their buffs, indicating overall strategy.
- **Graph 3:** Offers a view into player movement behavior over time, which can inform level design and pacing improvements.

5. Project Timeline

Week	Task
1 (10 March)	Proposal submission / Project initiation
2 (17 March)	Full proposal submission
3 (24 March)	Set up project structure, implement basic player movement and UI
4 (31 March)	Develop core combat mechanics
5 (7 April)	Implement enemy AI and looting system
6 (14 April)	Submission week (Draft)

6. Document version

Version: 1.0

Date: 4 March 2025

Date	Name	Description of Revision, Feedback, Comments
13/3	Rattapoom	Very Good! Since the project you've proposed has very high complexity, I'm a little bit worried whether you can finish the project in time without disturbing work-life balance. "Real-Time Adaptive AI" needs more explanation. ✓
16/3	Parima	Your project has very great detail and is well formatted. Good Work 🌟
28/3	Rattapoom	<p>Very detailed UML diagram 👍 Although, for something like</p> <p>Class Player</p> <ul style="list-style-type: none"><li>- weapons: list[Weapon]</li><li>- ...</li></ul> <p>Class Weapon:</p> <ul style="list-style-type: none"><li>- ....</li></ul> <p>You don't need to draw both "composition" and "association" arrows. Just a composition arrow is enough.</p> <p>Now for data collection, "Time taken to complete each stage" might be too tedious to collect since you'd have to play the game 50 times to collect all of that.</p>