D&D Dungeon Master Simulation

By: Joey Chuang & UV Raz

Introduction

This project models a simulated Dungeons & Dragons (D&D) combat scenario using an agent-based, grid-based approach. The goal is to explore how different configurations of player characters (PCs) and enemy non-player characters (NPCs), along with varied strategic behaviors, affect the enjoyment and pacing of combat encounters. Specifically, the simulation focuses on measuring the length of combat rounds, damage dealt and received, and emergent patterns of interaction, to identify optimal combinations of enemy count, composition, and tactics that lead to the most engaging gameplay experience.

Dungeons & Dragons, a tabletop roleplaying game first published in 1974, relies heavily on the Dungeon Master (DM) to create and manage encounters. In combat, the DM orchestrates the behavior of enemies in response to player actions. While the game is deeply narrative and improvisational, combat sequences are structured around rules governing initiative, movement, actions, and damage, making them a viable candidate for agent-based simulation. In this project, a simulated DM agent will generate and execute combat encounters under different scenarios, enabling statistical analysis of their outcomes.

The phenomenon being modeled, combat enjoyment and balance in turn-based RPG systems, has been explored in various academic and game design contexts. Yee (2006) highlighted that combat pacing and strategic depth are key components of enjoyment in roleplaying games. Adams and Rollings (2007) describe combat mechanics as a central feature of player engagement, with balance and variety being critical to long-term interest. More recently, Togelius et al. (2011) have shown that simulations and procedural content generation can be used to optimize game design elements such as enemy placement and difficulty scaling.

Most existing research has focused on modeling players or generating content automatically, but this project shifts focus to the Dungeon Master's role. By simulating a DM agent that runs battles using different enemy strategies and group compositions, the simulation can test how these changes influence key aspects of gameplay, like pacing, fairness, and engagement. For example, the simulation will analyze how tactics such as concentrating attacks on one player or spreading damage across the party affect the overall flow and enjoyment of combat. The goal is to identify patterns that consistently lead to satisfying and well-balanced encounters. Insights from this simulation could help game designers and DMs alike create more enjoyable and dynamic combat experiences, whether in traditional tabletop sessions or digital adaptations.

Model Description

Problem Analysis

The objective of this project is to model and simulate combat encounters in *Dungeons & Dragons* (D&D) using an agent-based, grid-based system. The simulation focuses on evaluating how various enemy strategies and compositions impact the pacing, fairness, and enjoyment of combat. The central question is: *What configurations of enemies and strategic behaviors lead to the most engaging combat experience for a given party composition?*

The problem is inherently stochastic, as player and enemy actions involve probabilistic outcomes (attack rolls, damage variability), and different strategies may produce highly variable results even under identical initial conditions. The system includes both deterministic rules (movement speed, initiative order) and probabilistic processes (dice rolls, action outcomes). The simulation must account for these interactions to produce useful, generalizable insights.

The model is intended to generate repeated simulations of combat scenarios with varying parameters (number of enemies, strategy types), allowing for statistical comparison across multiple runs. Key performance metrics will include combat duration (in rounds), damage dealt and received, and party member survival. These will be used to evaluate which enemy setups produce challenging but enjoyable experiences.

Model Formulation

Data Gathering

Initial model parameters and rules will be based on a heavily simplified version of Dungeons and Dragons 5th Edition (2014). These include character statistics, damage rules, turn structure, and enemy behaviors. Simplified stat blocks will be used for typical character classes (fighter, ranger) and a common enemy type.

Simplifying Assumptions Made

To make the model easier to work with, the following assumptions will be made:

- All characters act according to a basic decision tree based on their archetype (melee fighters prioritize close-range enemies, while ranged maintain distance).
- The environment will be a flat, grid-based battlefield without elevation, traps, or complex terrain.
- Player characters and enemies will have simplified, fixed stats with average HP and damage values representative of 3rd to 5th level characters.
- The only attack actions will be melee or ranged. Magic attacks such as Magic Missile and Fireball will not be included, since for the purposes of this model, they can be approximated by simpler ranged attacks.
- Enjoyment will be inferred from assumed measures: round length, damage balance, and number of surviving players.

Determine Variables and Units

- **Stock variables**:
    - Number of enemies
    - Enemy strategy (focus fire, target spread, terrain positioning)
- **Output variables**:
    - Number of rounds until combat ends
    - Total damage received and dealt
    - Number of PCs surviving

- **Constants**:
  - Grid size
  - Turn structure and initiative rules
  - Character and enemy movement speed
  - Number of players
  - Player strategy (melee, ranged)
- **Units**:
  - Time: measured in rounds
  - Damage: hit points (HP)
  - Position: (x, y) coordinates on the grid (each grid square = 5ft)

Establish Relationships Among Variables and Submodels

The system is composed of interacting components (or submodels) that operate in discrete time steps (combat rounds). Each round represents a full cycle in which all agents (player and enemy characters) act in turn. The overall system can be viewed as a dynamic process in which the state of the simulation evolves over time according to deterministic rules and probabilistic outcomes. The submodels are defined as follows:

- **Turn Submodel**: This submodel controls the initiative system, determining the order in which agents act. It updates the simulation clock by one round and activates each agent in order. Each agent's behavior and resulting actions are passed to the decision submodel.
- **Decision Submodel**: This submodel represents the decision-making logic for each agent. For each turn, the submodel receives the current state of the system (positions, HP, etc.) and outputs an action: move, attack, or do nothing. Strategies are encoded as decision trees or conditional rules, such as "move toward nearest enemy and attack" or "focus lowest-HP target." While player strategies remain fixed, enemy strategies will be systematically varied across simulations.
- **Combat Resolution Submodel**: Embedded within the turn and decision routines, this submodel applies the mechanics of the game to resolve attacks. It calculates hit/miss outcomes and determines damage dealt, based on probabilistic functions derived from dice rolls.

The interactions among submodels follow a cyclic structure:

- At the start of each round, the **Turn Submodel** sets the order of activation.
- Each agent, when activated, uses the **Decision Submodel** to determine its action.
- The chosen action is executed through the **Combat Resolution Submodel**, modifying state variables such as health and position.
- This process repeats until a terminal condition is reached (all enemies or all players defeated).

This modular structure enables experimentation by adjusting specific parameters (enemy behavior rules or number of agents) and observing their effect on the properties of the system.

Determine Equations and Functions

While no differential equations are required, the simulation will involve the following types of functions:

- **Movement function**: Uses Manhattan distance to determine legal movement paths.
- **Targeting function**: Returns list of valid targets based on strategy and constraints (range, line of sight).
- **Damage function**: Samples from a fixed probability distribution (1d8+3 damage = uniform distribution from 4–11).

These functions will be parameterized and run across multiple Monte Carlo simulations for each enemy setup.

Analysis

The primary goal of this simulation is to identify how different enemy configurations and strategies impact combat pacing, fairness, and inferred player enjoyment. To achieve this, the simulation will be run repeatedly under varying conditions, and statistical analysis will be performed on the resulting data.

Metrics and Quantities

The following output variables will be tracked for each simulation run:

- **Combat Duration**: Number of rounds until combat ends. Short battles may feel anticlimactic, while overly long ones can lead to fatigue or frustration.
- **Damage Distribution**:
    - Total damage dealt by players
    - Total damage received by players
- **Player Survival**:
    - Number of PCs surviving
    - Turn number of first PC death

These quantities will be analyzed as functions of configuration (enemy count, enemy strategy type). Collectively, they approximate the enjoyment metric by quantifying tension (close fights), pacing (combat speed), and fairness (balance of outcomes).

Key Questions

To guide the analysis, the following research questions will be asked of the simulation:

1. How do different enemy strategies (focus fire vs. distributed targeting) affect combat duration and player survivability?

2. Is there an optimal number of enemies that creates balanced, challenging encounters without overwhelming the players?
3. Which enemy behaviors lead to the most “engaging” balance of danger and player agency, as inferred from survival and damage trends?
4. Optional: Do certain player compositions (all melee vs. mixed melee and ranged) interact better or worse with specific enemy tactics?

Statistical methods such as means, variances, and histograms will be used to summarize results across multiple simulation runs.

Model Validation and Verification

To validate and verify the model, the following steps will be taken:

- **Internal Consistency Checks**: Ensure all mechanics (movement amounts, initiative order, damage rolls) align with D&D 5e rules. Sanity tests will confirm that no character acts out of order, moves illegally, or exceeds damage thresholds.
- **Behavioral Validity**: Run sample simulations to check if model behavior aligns with intuitive or rule-based expectations (a group of enemies with low accuracy should deal less damage).
- **Sensitivity Analysis**: Assess how small changes in parameters (HP, damage range, number of players) affect outputs to confirm the model responds realistically and is not overly brittle.

Over time, if the model’s predictions correlate with expected or observed gameplay experiences, it can be considered a useful exploratory tool for understanding basic encounter design dynamics in turn-based RPGs like D&D.

Testing

A structured testing suite will be implemented to ensure the correctness, stability, and reliability of the simulation model. We plan to build a testing suite using PyTest, using unit testing to ensure consistent validation across all major components of the system. Using PyTest we can also ensure that we can easily use it to test new changes to the code.

The testing suite will include the following categories:

- **Unit Tests**: Core functions (including movement logic, initiative sorting, damage resolution, and strategy selection) will be tested in isolation. These tests will verify that mechanics such as turn order, legal movement, and combat resolution conform to D&D 5e rules.
- **Sanity and Edge Case Tests**: These tests will check for internal consistency by simulating simplified scenarios with predictable outcomes. For example, a player surrounded by enemies should not be able to move illegally, and damage values should never exceed defined limits. Edge cases such as zero enemies, maximum grid size, or characters at 1 HP will be explicitly tested.
- **Invalid Input Handling**: Defensive checks will ensure that the model responds appropriately to malformed or missing data. This includes rejecting impossible grid

configurations or character stats and confirming that invalid inputs produce useful error messages rather than silent failures or crashes.

- **Regression Testing**: As new features are added, previously validated behaviors will be re-tested to confirm that no unintended side effects have been introduced.
- **Acceptance Testing**: Higher-level tests will be performed by executing full simulation runs with known inputs to confirm that the system behaves as expected. These will also serve as early user experience checks to ensure that interacting with the simulation (configuring a scenario or launching a batch of runs) is intuitive and free of errors.

Collectively, this testing approach will provide confidence in both the mechanical reliability and robustness of the simulation, supporting ongoing reproducibility of results across different configurations.

Personnel

UV will lead the design and implementation of the simulation model. This includes defining the agent behaviors, building the grid-based combat system, and implementing the turn structure, movement, and combat resolution logic. In addition to programming responsibilities, UV will also format the analysis documentation, ensuring the model's architecture and logic are clearly communicated and reproducible.

Joey will lead the development of the testing infrastructure used to validate the model's internal consistency and performance. He will also lead the writing and organizing the analysis report, including the development of data analysis routines and result visualization. Joey will assist with model implementation where needed, particularly in integrating testing and analysis protocols with the simulation framework.

Together, the team will coordinate design decisions, review each other's code and written material, and ensure that the final project is cohesive and well-structured across both technical and analytical components.

Technologies

The primary programming language for this project will be **Python 3.12.7**, managed through the **Anaconda distribution**. Anaconda provides a stable development environment with built-in support for scientific computing libraries such as **NumPy** (for array operations and numerical calculations), **Matplotlib** (for visualizing simulation results), and **PyTest** (for unit testing and verification). This setup enables efficient simulation design and analysis while maintaining modularity and reproducibility.

To support collaborative development, the team will use **GitHub** for version control. The project repository will contain all code, documentation, and presentation materials, with full commit histories enabling transparency around each team member's contributions. GitHub's branching and pull request workflows will allow for parallel development and systematic code reviews.

For team communication, **Discord** will be used to coordinate meetings, discuss design decisions, and troubleshoot issues in real time through both voice and text channels. This supports rapid iteration and keeps all members aligned during implementation.

For written documentation and presentation materials, the **Microsoft Office 365** suite will be used. Its real-time collaboration features in Word and PowerPoint will allow both members to contribute simultaneously to reports and slides, streamlining the production of deliverables throughout the project lifecycle.

Together, these technologies provide a robust, flexible foundation for collaborative software development and project management.

Benchmarks

Below is a timeline of benchmarks for the completion of our project. These milestones represent key stages in development, from initial planning through final submission. One benchmark has been designated as the formal **Milestone** for grading.

- **May 22, 2025 – Team Project Plan Completed**
  Finalized project plan submitted, outlining objectives, model structure, testing approach, and analysis framework.
  (✓ Completed)
- **May 26, 2025 – Initial Structure of Program Completed**
  Core architecture established, including simulation scaffolding, agent definitions, and input/output format. Early commits pushed to GitHub. The focus will be on ensuring that modules can interface and run in a minimal test environment.
- **May 28, 2025 – Project Alpha Complete [*Milestone*]**
  Basic functionality implemented. Key components should be operational, albeit with limited polish or error handling. At this point, the simulation should be runnable.
- **May 30, 2025 – Project Beta Complete**
  Full feature set implemented. The simulation should now reflect planned strategies, metrics, and edge cases. Attention will shift to fixing bugs, refining agent logic, and beginning structured data collection for analysis.
- **June 2, 2025 – Project Functionality Implemented**
  All simulation features finalized. Codebase enters freeze mode, with remaining work limited to bug fixes, parameter tuning, and finalizing visualizations, reports, and presentation materials.
- **June 4, 2025 – Project Completed**
  Submission of final deliverables, including simulation code, statistical analysis report, and team presentation, via Canvas.

Citations

Yee N. (2006). Motivations for play in online games. *Cyberpsychology & behavior : the impact of the Internet, multimedia and virtual reality on behavior and society*, *9*(6), 772–775. https://doi.org/10.1089/cpb.2006.9.772

Adams, E. (2025, January 28). *Fundamentals of Game Design*. Choice Reviews Online. https://www.academia.edu/81247135/Fundamentals_of_game_design

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 172–186. https://doi.org/10.1109/tciaig.2011.2148116