**STEMposium: Previously Acknowledged - Documentation**

**Story Overview  
  
  
Mechanics Overview**

This is a text-driven RPG adventure game built in Java. The game is console-based and takes a series of text inputs to navigate its world. The game features character creation, branching narrative scenes, inventory management, looting, combat, and world state management. The mechanics of the game are handled entirely with Java, and the story elements of the game are read dynamically from a series of .txt  
  
**Was AI Used?** Yes! However, the game was entirely coded by our team. The use of AI was as a helper tool, rather than actively coding. When our team had an idea for mechanics but was unsure of how to implement it or whether the concept was realistically feasible, that is when we would use our AI tools. We would query specific questions about the feature we wanted, and it would point us in the right direction. In this way, we were able to leverage its knowledge without it directly taking over the work. Additionally, AI has been used to format the scenes into the sceneBank.txt file and balance the game items, rather than us spending hours testing each item ourselves for balancing.

**Core Mechanics**

***1. Character Creation***

*Class Selection*: Players choose a class (e.g., Guard, Ranger, Technician), each with unique base stats. These stats are used in roll checks and combat.

*Stat Rolling:* After class selection, base stats are rolled and combined with class bonuses.

*PlayerValues*: Stores all player-related data (name, health, class, stats, currency, faction, etc.).

***2. Scene System***

*Scenes:* Defined in sceneBank.txt. Each scene has an ID, description, and a set of action options.

*SceneManager:* Loads and manages scenes from the sceneBank.txt, parses available actions, and handles scene transitions.

*WorldSceneTemplate*: Creates the actual scene following a template. Including its ID, description, and available actions.

*ActionOption*: Represents and handles a single action the player can take in a scene (label and trigger).

***3. Game State & World State***

*GameState*: Tracks the current scene, world state (exploring, combat, looting), and manages transitions.

*WorldStateEnum*: Enumeration of the primary world states (EXPLORING, COMBAT, LOOTING).

***4. Inventory & Looting***

*GenericItem*: Represents an item (name, ID, description, value, weight, stat requirements).

*Container*: Represents a lootable container (e.g., chest, barrel) containing a set of items.

*Looting*: Handles the logic for player looting containers and moving items to inventory.

*DataIO*: Reads/writes inventory and item data from/to text files.

***5. Combat System***

*CombatSystem*: Handles turn-based combat between the player and NPCs, using stats and equipped items.  
  
 - Combat Flow: Initiated by scene triggers, resolved via dice rolls and stat checks

*GenericNPC*: Represents an NPC (name, ID, description, health, etc.).

***6. Data Management***

*DataIO:* Central class for reading all game data (items, containers, NPCs, inventories, scenes) from text files in configFiles.

*Text Files*: All game content (items, containers, NPCs, scenes, etc.) is editable via plain text files for easy modification and expansion. The sceneBank.txt is where all the game's scenes live, and new scenes are loaded dynamically. Every time a scene loads in, the player is presented with a range of options. Every option the player can choose from has a scene ID or a world state, such as "LOOTING" or "COMBAT," tied to it. So, when an option is chosen, the following scene ID or world state loads in.

**How the Game Runs**

Windows: Use run\_game.bat to launch the game with the correct working directory. Simply unzip the game and double-click the .bat file; the game will then launch.  
  
Want to restart the game? Close and re-launch the .bat!  
 **Requirements:**  
  
Latest Java runtime; do you have it? Get it [here](https://www.java.com/en/download/manual.jsp)!