Phase 1 Design Document

Updated Specification:

No changes.

Major Design Decisions:
Created combat, decided only player can use items, player will never have allies in combat, but there can be multiple foes, as well as many other design choices.

We chose to use JSON for saving.

How to project adheres to Clean Architecture:

The project adheres to clean architecture because our classes are divided into distinct levels, with a set of entities (GameCharacter and subclasses, Quest and subclasses, Item and subclasses, and Scene), use cases (CharacterManager, QuestManager, SceneManager), controllers(GameLogic, combat), and command line interfaces (CommandLine,). However, we haven't finished removing entity parameters.

Brief description of how the project is consistent with the SOLID design principles:

The project is consistent with the single-responsibility principle because all of the classes have a single responsibility. The project is consistent with the open-closed principle because the abstract classes are open for extension by some of the concrete classes, while they are no longer heavily modified. The project is consistent with the Liskov substitution principle because objects of superclasses can largely be (and frequently are) substituted for objects of their subclasses. The project is consistent with the interface segregation principle because the interfaces for all of our classes were built with methods only added as needed. The project is consistent with the dependency inversion principle because higher level modules never depend on lower level modules.

Description of which packaging strategies we considered, which we decided to use and why:

Initially, we were split between packaging by feature and packaging by layer. Packaging by layer would allow us to confirm more easily that we were adhering to clean architecture through the course of development. However, in the end, we packaged our classes by feature because our program is composed of a number of distinct, highly cohesive features. We decided that this was the most intuitive strategy and easiest to work with.

Summary of design patterns we implemented:

We've implemented a facade to facilitate pairing inventories and characters. We've also implemented dependency inversion.

Progress report:

- ”open questions your group is struggling with”
    - How can we implement more design patterns?
- ”what has worked well so far with your design”
    - Everything technically works.
- ”a summary of what each group member has been working on and plans to work on next”
    - contributions so far
        - Tim: style + Javadoc, participation in decision making, maintaining design document
        - Bernie: Made TA requested changes, tests, and a lot of random fixes post merging, helped others with any issues
        - Eric: Wrote worldstate, partook in decision making, and design choices, upheld clean architecture.
        - John: Thought about the idea for saving/loading and started implementing code. Added a few needed getter methods.
        - George: Handled writing code for facilitating combat, partook in decision making.
        - Edward: Wrote Character, Inventory, Item entities, use cases, used Facade design pattern to combine Inventory and Character subsystems by following Clean Architecture. refactoring code and partook in design choices.
    - next steps
        - Magic
        - Build on the status effect skeleton created
        - Actual gameplay (more quests, npcs, etc.)
        - A story