Here's the technical details of the project. I’ve done my best to include details in the correct order as much as possible. Much of the game's engine revolves around the GameObject and Location classes. All events in the game are attached to a GameObject and are executed through scripts that are interpreted by the Interpreter class. Here’s a description of each class:

* GameObject is all the objects that exist in the game. Objects have a state (such as “on” or “off “ for light switches) as well as a set of properties (for example the player’s health). The player is a GameObject. The player’s inventory is a Location, which contains within it all of the player’s items.
  + GameObjectManager as its name implies manages GameObjects by loadObjects() which loads GameObjects from “object” text files, verifies whether a String matches the name of a GameObject, creates new GameObjects. It also contains the private class Loader which can parse “event” text files which contain the scripts for every event that occurs in the game. (All events that occur must be tied to a GameObject.)
* Location is all of the locations that exist in the game. Locations have three maps as global variables. The “children” map contains all GameObjects that exist with the Location, each of which has as its key its name (i.e. lamp, sword, first aid kit). The “subLocation” and “adjacentLocation” maps contain the names of sublocations (such as rooms that exist within a building) as well as adjacent locations that the player can travel to from this location.
  + LocationManager can return various aspects like the original location (“root”) or something or the current location (“context”). It can get sublocations of a Location and get GameObjects from a Location. LocationManager also handles loading “location” text files.
* Event is an array of Strings. They are generally read from text files that contain the scripts for running the game. All events are called by a Trigger.
* Interpreter is the class the interprets events. This is where everything happens essentially.
  + When a user inputs a line, it is read by the interpret(String line) method. Each line should be in the form of a command, and this method calls interpretCommand(). (User inputs are preceeded by “do” causing user inputs to always go to the DoCommand.) This method automatically saves each input to a text file as well.
  + interpretCommand() identifies the command that needs to be executed and executes it.
  + skipUntilNewLine() is used to skip to a certain line in scripts and is used with IfCommands.
  + interpretSentence() is called by DoCommand and attempts to form a phrase with PhraseBuilder from user input with the Phrase class. It attempts to get a Trigger based on the phrase, which can then determine the Event that needs to be called. It then called interpret(Event event) to interpret this event, or interpretNonVerbSentence() if no phrase matched the user input.
  + interpret(Event event) interprets all the lines of the input Event.
  + interpretNonVerbSentence() attempts to interpret other types of input without a verb such as “exit”, “save” or “yes” and calls printError() if nothing matched.
* Commands are various actions that are run that change the state of the game or output something to the user.
  + The DoCommand is exclusively for user input. It calls the interpretSentence() method within Interpreter to interpret the user input. This command also prevents users from being able to access commands reserved for scripts.
  + The IfCommand handles if statements within the scripts. It determines if the words following it represent a true sentence, and if so, the next lines of the script are executed until the next “if” within the script. If it is not true, then it skips until the next “if” statement. It can also handle “if not” statements.
  + The MakeCommand allows for creating a GameObject within a Location.
  + The SayCommand is what is used for outputting text to the user.
  + The SetCommand sets can set the state, property, and Location of GameObjects.
* Phrase is just a class that contains the necessary aspects of a phrase like the phrase’s object, subject, and indirect object, as well as the response (i.e. “yes,” or “no”) and the verb.
  + PhraseBuilder attempts to match a user input sentence to a “verb structure” that the game supports, such as “answer RESPONSE to OBJECT” or “open OBJECT”. It compares the user input with a list of all verb structures found in the text file “structures”. It will also change user typed items with more than one word in its name (such as first aid kit) and makes it into a friendly form for the engine (e.g. first\_aid\_kit).
* RunGame is the class for handling running the game, as well as loading saved games, creating new games, and exiting the game. My placeholder loading system is to just re-interpret every previous event the user typed in, but will likely change this in the future. Creating a new game occurs by replacing the save file with the new game file.
* Constants are names of various files used to keep track of saves as well as scripts used to run the game.
* Utils are various utilities used by other classes.
* TestLoader is the class I’ve been using to test the various things described above. It currently runs the events under the text file “new” in the beta/Locations/World/Ship/Closet. The events from scripts appear to be working correctly, and many types of user input can be processed correctly. However, commands that change locations still are not fully bug-free. To run it, just change the ROOT String in the Constants.java class to the correct directory for your own computer. Currently, a very limited amount of scripts have been made, mostly to test the engine itself. The output also includes a substantial amount of debugging System.out.prints in addition to the actual output that will come up during gameplay. For example, inputting “examine lamp” or “examine first aid kit” both will show you the output from their respective scripts (as well as a bunch of other things).

Some early example scripts can be found in the “examples” folder. The “beta” folder contains some more updated scripts. It includes the Objects and Locations directories described in the Constants class.

* The Objects directory contains various example objects. Each object contains an “events” file which lists all of the events the object has as well as each event itself.
* The Locations directory will have all the locations in the same hierarchy that exists in the game.