**Order for STAG**

Design: UML class diagram

Setup: Link up server and client and check it passes STAG check.

Code build order:

1. Read in and parse entities file

2. Store the entities file in a data structure

3. Implement goto and look commands to be able to explore these data structures/game world

4. Then look at pickup, get, drop etc.

5. Deal with actions

**STAG Keywords / Is A, Has A**

* Game Engine (similar to DBServer?)
* Entities (unique names) – Things in game, layout, and relationships. All have **name** and **description.**
  + **Map** – ArrayList of Locations classes.
  + **Locations** – Rooms/places
    - Every game has special **starting** location. First location in entities file
    - **Unplaced** special location for entities with no initial location
    - E.g., Forest has:
    - **Paths** to other locations, **Characters** at location, **Artefacts** at location, **Furniture** in location.
    - Location is basically a node of a graph, door/node is an edge of a graph
    - Location will be an **object** with two/three attributes: ArrayList of Furniture classes, ArrayList of Artefact classes and ArrayList of “doors”/paths with next node/location.
    - Can take Artefacts from Location ArrayList and put in Artefact ArrayList in new Location, or have a player inventory. Effectively change graph/DOT file by updating the data structure.
  + **Artefacts** – things that can be collected.
    - E.g., Axe, keys, potions etc
    - Potion; red-potion or blue-potion
  + **Furniture** – things are integral to location and non-collectible.
    - Tree
  + **Characters** – Creatures/people
    - E.g., elf
  + **Players** – Represents the user
    - Inventory
    - Health
* Actions – dynamic behaviours of entities
  + **Trigger** words – Initiate the action.
    - e.g., open, unlock.
  + **Subject** entities – Entities that must be present in inv and location to do action.
    - E.g., door, key.
  + **Consumed** entities – Entities that are removed/eaten up by action.
    - E.g., key
  + **Produced** entities – Entities created as a result of action.
    - E.g., cellar
  + Game engine should verify conditions for actions to be performed,
    - Then undertake relevant additions/removals