Adventure

- handin should have three elements – class design, pseudocode, plot

- Read through the Specifications and use textual analysis as discussed in class to develop an initial class design. For each class, give it a descriptive name, briefly identify its purpose (what does it represent?), identify methods it will have, and identify information that it needs to store (instance variables).
- Outline the main program's task in pseudocode. Keep a big-picture view — like an outline for a paper, cover all of the functionality of the main program (the paper's content) but at a high level (the main points) without all of the details (the actual text of the paper).
- Describe the plot for your version of Adventure — what's the overall theme? What tasks must the player accomplish to win the game? Also identify the rooms and items, and draw a map showing how the rooms connect, where the items are initially, where the player starts, and where the goal room / exit is.

  - if you missed any of these elements, you are strongly encouraged to come to office hours to discuss your ideas in order to make sure you are on the right track

Adventure

- make sure you are aware of the specifications for the game engine vs extras
  - your program must support the specifications given
    - (only) the commands listed in the handout
    - tasks (only) involve taking or dropping items and/or visiting rooms
    - winning the game means accumulating some number of points and moving to the goal/exit room
  - get the required version working before adding extras
    - make substitutions to accomplish the spirit of what you want to do within the specified framework
      - e.g. using an item – substitute dropping the item in a specific room
      - e.g. unlocking/revealing rooms – substitute dropping a key or other item in the room without requiring a particular sequencing of actions
      - e.g. fighting/defeating enemies – substitute dropping weapons or other items in particular rooms
  - any extras should add on – do not replace the specified elements with an alternate version
    - must still support the specifications (including the commands) given!
    - discuss your ideas with me if in doubt

Adventure

- tasks and winning
  - the objective of the game is to accumulate a certain number of points and then move to a particular goal/exit room – be sure to include this in your plot!
  - points are accumulated by dropping items in a particular room or visiting a particular room
    - can add or substitute taking an item instead of dropping, but at least one task needs to involve taking or dropping an item
    - drop/take tasks are intended to reflect the desired end location of the item – in a particular room if dropped, in the inventory if taken – so points should be removed if the player undoes the task
    - tasks can only be completed once for points (no additional points for repeated visits, points removed if drop/take undone)
  - to require the player to complete all of the tasks in order to win, make the points required to win to be the total of the points for all of the tasks

Scoring and Winning

Points are earned by completing various tasks such as dropping an item in a particular location or visiting a certain room. The player wins the game by accumulating a certain number of points and then moving to a particular goal or exit room (which might represent escaping from the game's world).

Winning the game must require the completion of at least three tasks before moving to the goal/exit. At least one of those tasks must involve an item (such as picking up an item or dropping an item in a particular room), and at least one must involve visiting a certain room.

Be careful not to allow the player to accumulate points by repeating tasks — points for visiting a room should only be awarded the first time the room is visited, and points for dropping an item in the right location should be revoked if the item is picked up again.

Adventure

Class design –

- missing elements / things to be sure to address
  - the world
    - how are the north/south/east/west connections between rooms stored?
  - whether or not a room has been discovered
    - only display the short description when entering a previously-visited room
  - tasks/scoring
    - how do you keep track of the various tasks to be completed, what has been completed, etc?
- placement of methods
  - methods go in the class whose instance variables they need or manipulate, not with the subject of the sentence
    - e.g. the player does lots of things, but only methods that deal with `Player`'s instance variables belong in the `Player` class
  - methods should fit wholly with the purpose and job of the class
    - e.g. TAKE involves removing the item from the room and adding it to the player's inventory – `Room` and `Player` will need methods to remove and add the item, respectively, but the code to carry out the whole TAKE action doesn't belong in either class

Adventure

Class design –

- *separation of concerns*
  - the user interface is separate from the code carrying out the actions the user has specified
    - there can be different user interfaces which support the same actions, or different ways to specify an action within the same UI
  - reflect this in your design by putting separate concerns in separate classes (or at least methods)
    - for a text-based program, the main program can handle the UI (getting commands from the user)
    - have methods (private helpers or methods in another class) corresponding to each of the game commands – GO, TAKE, DROP, etc
      - the main game loop in main reads a command, figures out which command it is and the relevant additional info, and calls a method to carry out that action

Adventure

Pseudocode –

- pseudocode should be structured like code, not a TODO list

```
initialize the world
repeat until the game is over
  get a command from the user
  handle the command
if win, display congratulatory message
otherwise display consolation message
```

- start broad, then refine far enough to be able to complete your class design
  - e.g. for handle the command –

```
if GO or MOVE then
  if there's a room in that direction from the current room,
    move to that room
    if the room has been visited already,
      display the short description
    otherwise
      display the long description (including exits)
    display the room's contents
    visit the room
else if …
```

need the current room

Room needs an isRoom(dir) method

need to update the current room

Room needs a isVisited() method

Room needs getters or print methods for the short description, long description, and exits

Room needs a getter or print method for displaying the contents

Room needs a setVisited() method

Adventure

Data structures and organization –

- choose appropriate collection types (and underlying implementations)
  - arrays are convenient for fixed-size collections (no add or remove)
  - choose List, Stack, Queue, Set for variable-sized collections depending on access needs
  - choose Map for lookup tasks (key-value pairs)
- for storing rooms (in the world) and items (in the inventory, room contents), do you store the name of the room/item or the object itself?
  - consider what is most convenient for how you need to access things in the code