INFO1110 / COMP9001 Assignment 2

Adventure

Deadline: 11:59 PM, Sunday 26th of May 2019 AEST.

Weighting: 15% of the final assessment mark.

> *At the heart of each adventurer burns a passion: a passion for gold, for glory, for treasure or fame, or an intense, burning desire for the world to be alright so that they could be left the hell alone. And each adventurer - no matter how noble, no matter how fickle or selfish they may be, is defined by the journey upon which they embark.*
>
> *And yours, it appears, has taken you here: to the foot of a temple to a forgotten god, long since lost to time. Seek you knowledge? Seek you fame? Seek you treasures beyond compare? Herein lay trials for you to overcome, dear adventurer, and the only way to go is in.*

Credits: Pixabay, Uploaded by Enrique Meseguer on Nov. 13, 2017 .

Overview

Description

For this assignment, you will write a simulation of a fantasy adventure, or a dungeon crawl. The user will be given controls that allows them to move through rooms and locations in search of trials and tribulations to overcome (for fun and profit).

Implementation details

Your program will be written in Python 3. The only Python modules allowed for import are `sys`.

To help you begin, a scaffold has been provided. Your entire program must be contained in the files `room.py`, `simulation.py`, `item.py`, `quest.py`, and `adventurer.py`. You should implement the functions in these files to the best of your ability. You may create new functions to help you as you see fit, but you cannot modify any existing function signatures (i.e. you cannot change the amount of arguments that an existing function can take).

Help and feedback

You are encouraged to ask questions about the assignment on the discussion board, on Ed.

> During your tutorial in Week 12, you can also ask your tutor to review your code. Your tutor may provide feedback either during the class, or outside the class on Ed.

Please ensure your code is comprehensible before requesting feedback. We recommend that your code adheres to the PEP 8 style guide, and is commented appropriately.

Staff may make announcements on Ed regarding any updates or clarifications to the assignment. You can ask questions on Ed using the assignments category. Please read this assignment description carefully before asking questions. Please ensure that your work is your own and you do not share any code or solutions with other students.

Submission

You will submit your code on the assignment page, on Ed. You are encouraged to submit multiple times. After each submission, the marking system will automatically check your code against public test cases.

These public tests do not cover all parts of the specification and your code. The complete test suite contains both public and hidden test cases, and your code will not be run through this suite until after the assignment deadline.

Please ensure you carefully follow the assignment specification. Your program output must exactly match the output shown in the examples.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specifications, or your code is unnecessarily or deliberately obfuscated.

Milestone

To ensure you are making regular progress on this assignment, you must have achieved a minimum level of functionality in your submission by **May 19th, 11:59 PM AEST** (Week 11 Sunday) to receive a portion of the marks. See the *Milestone Submission* section at the end of this document for further details.

Program Details

Unless otherwise specified, all string values discussed in this program specification can be assumed to be **single-line strings**.

The `Adventurer`

The user is represented by an `Adventurer` class, which you can define in the given `adventurer.py` scaffold. An `Adventurer` object represents the character that the user controls. It has three primary attributes:

- An `inventory`, which keeps track of all the items that the user has collected throughout the course of the game. When an `Adventurer` object is first created, this attribute is empty.
- A `skill` level. This represents the character's ability to overcome physical challenges within the game. This integer value begins at `5`, and never goes lower than `0`.
- A measure of `will` power. This represents the character's ability to overcome mental challenges, resist mind-affecting effects, and influence other creatures. This integer value also begins at `5`, and never goes lower than `0`.

The `adventurer.py` scaffold specifies a few methods that should be implemented for the purposes of handling this object. **This is true for all other scaffold files provided for this assignment.** Feel free to add as many more methods as you feel is necessary.

Rooms

Each room - or location - in the game is represented by an object of the `Room` class, which you can define in the given `room.py` scaffold. A `Room` object has the following attributes:

- A `name`
- A `quest` that can be/has been completed in this room. In some rooms, no such quest exist (i.e. this attribute has value `None`). The room's appearance changes based on whether or not the quest has been completed, or if a quest exists in it at all.
- An attribute for each of the cardinal directions: `north`, `south`, `east`, and `west`. Each of these attributes might be:
    - Another `Room` object that can be reached from this room by moving in the appropriate direction, or
    - The value `None`, in the case that no other rooms can be reached from this room by going in the specified direction.

When the user enters a `Room` or when the `LOOK` command is invoked, a **display** representing the `Room` is printed to `standard output`. Such a display consists of:

- A visualisation of the room and its possible exits: a box that is **11 lines** tall and **22 characters** wide. When an exit is present from any cardinal position (north, south, east, west), the centre of the corresponding wall on the map is *replaced* with letters, like so:

```
+---------NN---------+
|                    |
|                    |
|                    |
|                    |
W                    E
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Foyer.
There is nothing in this room.

>>> WEST
You move to the west, arriving at the Workshop.

+--------------------+
|                    |
|                    |
|                    |
|                    |
|                    E
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Workshop.
There is nothing in this room.

>>>
```

- A line indicating the name of the room. It follows the form: `You are standing at the <room_name>.`
- A separate, single line containing a short description of the room. This description changes based on whether or not a relevant `quest` has been completed in this room.
    - If there are no quests that are relevant to this `Room`, its description should read: `There is nothing in this room.`

Items

Every item in the game is represented by an object of the `Item` class, which you can define in the given `item.py` scaffold. An `Item` object has the following attributes:

- A `name`. This can be quite lengthy (e.g. `a foul-smelling bouquet of flowers`.)
- A `shortname`. This is usually a **key word** from the item's full `name`. (e.g. `bouquet`, or `flowers`.)
- `skill_bonus` - An integer value. When an `Adventurer` is carrying an item in their `inventory`, its `skill_bonus` is added to the `Adventurer`'s `skill` level.
- `will_bonus` - An integer value. Works just like `skill_bonus`, but for the `Adventurer`'s `will` power instead.

When a user invokes the `CHECK` command, they can attempt to examine an `Item` more closely by specifying an item by its `name` or `shortname`. So long as the `Item` exists in the player's `inventory`, doing so allows them to see the `Item`'s full `name`, its `skill_bonus`, and its `will_bonus`. **For example:**

```
>>> CHECK
Check what? Shield

Shimmering Shield
Grants a bonus of 5 to SKILL.
Grants a bonus of 2 to WILL.
```

Quests

You, the player, are an adventurer with a purpose: an adventurer with a *quest*, or perhaps *many quests* - tasks for you to complete in exchange for a reward (fame, glory, money, more treasure, you name it). In our program, such tasks can be represented by `Quest` objects. A `Quest` object has the following attributes:

- A `reward` - some `Item` that is added to the player's inventory once the `Quest` is complete.
- A quest `action` - a special action that can only be activated in the `Room` that the `Quest` can be completed in. More on this later.
- A quest `description` - a brief description of what the quest might entail, like a hint.
- `before_text` - This is what is printed as part of a Room's description if the `Quest` can be completed in that room, but the `Quest` is not yet complete.
- `after_text` - This is what is printed as part of a Room's description if the `Quest` can **and has been** completed in that room.
- `requirements` to complete the quest. You can expect this to always be **a single string in two parts**:
    - `<attribute being tested> <value needed to complete the quest>`
    - For example: `SKILL 10`, `WILL 6`, etc.
    - You can and probably should make some extra variables for the `Quest` object that may not be included in the scaffold.
- `fail_msg` - This is printed when an `Adventurer` attempts to complete a `Quest`, but their `skill` or `will` values are too low.
- `pass_msg` - This is printed when an `Adventurer` attempts to complete an `Quest` and succeeds!

- A `room` that the `Quest` can be completed in (i.e. a `Room` object that is affected by this `Quest`'s `before_text` and `after_text`).

This is a lot to take in, so let's illustrate this with an example:

```
Reward: Tiny Cat
Action: FEED CAT
Description: FEED a hungry cat!
Before_text: A tiny cat mewls at you pathetically from a corner of the room. It looks hungry.
After_text: There is nothing of note here - just books.
Requirements: WILL 7
Fail_msg: You offer the cat some food, but it runs away from you!
Pass_msg: You offer the cat some food. It happily accepts, and climbs up on your shoulders. Looks
like you made a friend!
Room: Library
```

Okay! Let's assume that this quest exists, and let's see what we might see when we enter the `Library`. Let's assume that our `Adventurer` is currently carrying no items.

```
+--------------------+
|                    |
|                    |
|                    |
|                    |
W                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Library.
A tiny cat mewls at you pathetically from a corner of the room. It looks hungry.

>>> FEED CAT
You offer the cat some food, but it runs away from you!

>>> L

+--------------------+
|                    |
|                    |
|                    |
|                    |
W                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Library.
A tiny cat mewls at you pathetically from a corner of the room. It looks hungry.
```

```
>>>
```

Oh no, it looks like we don't have enough `WILL` to complete the quest! Let's try another example, but this time, we *are* carrying some items that boost our `Adventurer`'s `WILL` to a value of, say, `8`.

```
+--------------------+
|                    |
|                    |
|                    |
|                    |
W                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Library.
A tiny cat mewls at you pathetically from a corner of the room. It looks hungry.

>>> FEED CAT
You offer the cat some food. It happily accepts, and climbs up on your shoulders. Looks like you
made a friend!

>>> LOOK

+--------------------+
|                    |
|                    |
|                    |
|                    |
W                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Library.
There is nothing of note here - just books.

>>> INV
You are carrying:
- A Will-Booster
- A Tiny Cat

>>>
```

Note that when a `Quest` is complete, the relevant `Room`'s description changes!

The CONFIG files

When the program begins, it creates a series of rooms, items, and quests with different attributes based on the configuration files passed to it through the command line. Your program will receive the following information (in the order given) as command line arguments:

- `path_config` - the name of a file containing the list of all connections between rooms in the program. Use this file to determine how many `Room` objects you have to create! Each line is of the form:

  ```
  START > DIRECTION > DESTINATION
  ```

  Where `START` and `DESTINATION` are the names of rooms, and `DIRECTION` indicates a cardinal direction (north, south, east, west) that the user can use to move between `START` and `DESTINATION`. For example:

  ```
  Entrance > NORTH > Foyer
  Foyer > SOUTH > Entrance
  ```

  When the program starts, the `Adventurer` begins in the FIRST room specified by this file.

- `item_config` - the name of a file defining all the items to be found in the adventure on each line. Each line is of the form:

  ```
  item_name | shortname | skill_bonus | will_bonus
  ```

  Where `item_name` indicates the item's full name, and `shortname` indicates an abbreviation of `item_name` that the user can use to refer to it when entering commands. For example:

  ```
  Singing Sword | sword | 10 | 2
  ```

- `quest_config` - the name of a file defining all of the quests to be completed throughout the course of the game. Each line is of the form:

  ```
  Reward | quest action | quest description | before_text | after_text | requirements |
  fail_message | pass_message | room
  ```

Examples of these configuration files are available in the provided scaffold. Empty lines encountered in any config file can be safely skipped/ignored. Check for the files `paths.txt`, `items.txt`, `quests.txt` respectively.

If fewer than 3 arguments are supplied, print: `"Usage: python3 simulation.py <paths> <items> <quests>"` and exit the program.

If any one of the configuration files are invalid (that is, they don't exist), print: `"Please specify a valid configuration file."` and exit the program.

Similarly, if an empty `path_config` file is given, print: `"No rooms exist! Exiting program..."` and exit the program. If an empty `item_config` or `quest_config` file is given, the program should run normally.

Commands

Unless stated otherwise, all commands are case insensitive.

`QUIT`

At any point, the user may end the simulation.

```
>>> QUIT
Bye!
```

`HELP`

The simulation lists all valid commands and their usage.

```
>>> HELP
HELP       - Shows some available commands.
LOOK or L  - Lets you see the map/room again.
QUESTS     - Lists all your active and completed quests.
INV        - Lists all the items in your inventory.
CHECK      - Lets you see an item (or yourself) in more detail.
NORTH or N - Moves you to the north.
SOUTH or S - Moves you to the south.
EAST or E  - Moves you to the east.
WEST or W  - Moves you to the west.
QUIT       - Ends the adventure.
```

On each line of this output, the left side before the dash is always padded so that it is **11 characters in width.**

`LOOK` and `L`

Displays the room that you are currently in.

```
>>> LOOK

+---------NN---------+
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Entrance.
There is nothing in this room.

>>>
```

```
>>> L

+---------NN---------+
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Entrance.
There is nothing in this room.

>>>
```

`QUESTS`

Shows a list of all the quests in the game.

```
>>> QUESTS
#00: Singing Sword        - PULL the sword from the stone.
#01: Shimmering Shield    - SHINE an old shield until it shimmers.
#02: Trembling Tome       - CALM a trembling tome in the workshop.
#03: Glistening Goblet    - STEAL a glistening goblet from a distracted denizen.
```

Each line of the list comes in four parts:

- A two-digit number of the form `#XX`, padded by `0`s.
- A quest name (the quest's `reward`), padded out to **21 characters**.
- A quest description, and
- If the quest is complete, a tag that says `[COMPLETED]` at the end.

```
>>> QUESTS
#00: Singing Sword        - PULL the sword from the stone.
#01: Shimmering Shield    - SHINE an old shield until it shimmers. [COMPLETED]
#02: Trembling Tome       - CALM a trembling tome in the workshop.
#03: Glistening Goblet    - STEAL a glistening goblet from a distracted denizen.
```

If ALL quests are complete (or if there are no incomplete quests), print the list normally. Print a new line, then: `=== All quests complete! Congratulations! ===`, and end the program.

```
>>> QUESTS
#00: Singing Sword        - PULL the sword from the stone. [COMPLETED]
#01: Shimmering Shield    - SHINE an old shield until it shimmers. [COMPLETED]
#02: Trembling Tome       - CALM a trembling tome in the workshop. [COMPLETED]
#03: Glistening Goblet    - STEAL a glistening goblet from a distracted denizen. [COMPLETED]

=== All quests complete! Congratulations! ===
```

`INV`

Shows a printout of the user's inventory.

```
>>> INV
You are carrying:
- A Shimmering Shield
- A Singing Sword
```

If the user is carrying nothing, it instead says:

```
>>> INV
You are carrying:
Nothing.
```

`CHECK`

Allows the user to examine items. it will ask them for a second input, which can be an item's name or its short name.

```
>>> CHECK
Check what? Shimmering Shield

Shimmering Shield
Grants a bonus of 5 to SKILL.
Grants a bonus of 2 to WILL.



>>> CHECK
Check what? Shield

Shimmering Shield
Grants a bonus of 5 to SKILL.
Grants a bonus of 2 to WILL.



>>>
```

If no such item exists in the user's inventory, it will instead print:

```
>>> CHECK
Check what? Grass Sword

You don't have that!

>>>
```

Inputting `ME` the second time around allows one to examine their in-game statistics, and will print out the statistics of any item they are carrying, as well.

```
>>> CHECK
Check what? ME

You are an adventurer, with a SKILL of 5 and a WILL of 5.
You are carrying:

Shimmering Shield
Grants a bonus of 5 to SKILL.
Grants a bonus of 2 to WILL.

With your items, you have a SKILL level of 10 and a WILL power of 7.
```

The final line talks about what the user's statistics are after all bonuses from items have been applied. If the adventurer is carrying nothing, print the following:

```
>>> CHECK
Check what? ME

You are an adventurer, with a SKILL of 5 and a WILL of 5.
You are carrying:

Nothing.

With your items, you have a SKILL level of 5 and a WILL power of 5.
```

`NORTH` or `N` | `SOUTH` or `S` | `EAST` or `E` | `WEST` or `W`

Moves the user to a connecting room in that specified direction.

```
+---------NN---------+
|                    |
|                    |
|                    |
|                    |
W                    E
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Foyer.
There is nothing of note here.

>>> EAST
You move to the east, arriving at the Parlour.

+--------------------+
|                    |
|                    |
|                    |
|                    |
W                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
You are standing at the Parlour.
A couple of fat cats are here, playing cards.

>>>
```

If there is no room that can be accessed from by moving in the specified direction, they will instead print:

```
>>> EAST
You can't go that way.
```

Invalid Commands

If the user enters an invalid command, print `You can't do that.` and ask for another command.

```
>>> CRY
You can't do that.

>>>
```

Quest Actions

A **quest action** is a special action that can only be activated in the `Room` that the relevant `Quest` can be completed in. This is an attribute that has been stored as a string in a relevant `Quest` object. Each `Room` can only contain one `Quest`, and so can only have one relevant quest `action`.

The output following a quest action is contingent on two things:

1. Whether or not the quest has been completed, and if not,
2. Whether or not the `Adventurer` has enough `SKILL` or `WILL` to complete the quest.

For cases 1 and 2, let's assume that we are working with a quest that has yet to be completed.

Case 1: If the `Adventurer` inputs a quest `action` without enough `SKILL` or `WILL` to complete the quest, they will receive the quest's `fail_msg`, and nothing changes.

```
+--------------------+
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Courtyard.
A beautiful silver sword has been wedged deep into a stone pedestal here.

>>> PULL SWORD
You struggle to pull the sword from the stone, but despite your best efforts, it doesn't budge.

>>> L

+--------------------+
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Courtyard.
A beautiful silver sword has been wedged deep into a stone pedestal here.

>>>
```

Case 2: If the `Adventurer` inputs a quest `action` and is capable of completing the quest, they will receive the quest's `pass_msg`, the quest's `reward` will be placed in their `inventory`, and the description of the room should change.

```
+--------------------+
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Courtyard.
A beautiful silver sword has been wedged deep into a stone pedestal here.

>>> PULL SWORD
You heave against the stone and pull the sword free from its grasp. You hear the sharp hum of the
blade as it moves through the air - a job well-accomplished.

>>> L

+--------------------+
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Courtyard.
A stone pedestal lies in the centre of the room, devoid of swords.

>>>
```

Case 3: The quest is already complete. Regardless of the `Adventurer`'s `SKILL` or `WILL` value, they will see the following message: `You have already completed this quest.`

```
+---------NN---------+
|                    |
|                    |
|                    |
|                    |
W                    E
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Foyer.
An old, battered shield rests against the wall here.

>>> SHINE SHIELD
The shield shines and shimmers like a mirror made of steel. You decide to take it with you.

>>> SHINE SHIELD
You have already completed this quest.

>>>
```

Case 4: If you attempt the right quest `action`, but in the wrong room, you'll receive this message (as if you'd just entered an invalid command):

```
+---------NN---------+
|                    |
|                    |
|                    |
|                    |
W                    E
|                    |
|                    |
|                    |
|                    |
+---------SS---------+
You are standing at the Foyer.
An old, battered shield rests against the wall here.

>>> PULL SWORD
You can't do that.

>>>
```

Submission and Mark Breakdown

Submit your assignment on Ed in the *Assignments* section of the **Assessments** tab. The marking breakdown of this assignment is as follows (**15 marks total**).

- **3 marks** will be awarded as a progress mark, as described in the *Milestone Submission* section below.
- **4 marks** will be awarded for code correctness, assessed by automatic test cases on Ed. Some test cases will be hidden and will not be available before the deadline.
- **8 marks** will be given through hand-marking.
    - **2** of these marks will be for code style, readability, and appropriate code comments.
    - A further **2** marks will be on general code/logical correctness (does your program basically function as it should, but fails the automarking for some reason?)
    - The remaining **4** marks will be awarded for the submission of *test cases*.

Submitting Test Cases

A test case is numbered, and consists of the following files (where `XX` is the number associated with that test case, e.g. `01`):

- `XX_input.txt` - The input for your test case.
- `XX_path.txt`, `XX_item.txt`, `XX_quest.txt` - path, item, and quest configuration files for your test case.
- `XX_expected.txt` - The expected output for your test case.

Such test cases should be placed in a `tests` directory, which should be included when you upload your program for submission on Ed. Justifications for each test case must be written in a `README.txt` file in this `tests` directory.

- For the sake of clarity, an example test case (numbered `00`) and the `README.txt` file has also been included the scaffold, inside the `tests` directory.

You are expected to build a suite of at least **9** test cases. If you're unsure of where to start, it is recommended that you attempt to build one trivial and one non-trivial test case for each command, because a portion of the marks awarded will be for the amount of coverage offered by your test cases.

Using Test Cases

For reference, this is how we expect to use your test cases. You can do the same thing in your terminal in order to test your code yourself:

```
python3 simulation.py XX_path.txt XX_item.txt XX_quest.txt < XX_input.txt > XX_actual.txt
diff XX_expected.txt XX_actual.txt
```

The first line creates a new file, `XX_actual.txt`, that contains the output of your program given your test case's input. The second line compares the contents of `XX_actual.txt` and `XX_expected.txt`, and will notify you if they are any different (i.e. the program fails your test case).

If the second line produces no output, the program has passed your test case (`XX_actual.txt` and `XX_expected.txt` are the same).

Milestone Submission

2 marks will be awarded for a submission before **May 19th, 11:59 PM AEST** (Week 11 Sunday) that meet the following criteria:

1. The program runs without crashing.
2. The program can successfully detect the existence of configuration files (and appropriately handles cases where no configuration files exist).
3. The `draw()` function in `room.py` can draw an empty room, with no exits.
4. The `take()` function in `adventurer.py` actually changes the contents of the `Adventurer`'s inventory.
5. Given a complete `path_config` file and empty `item_config` and `quest_config` files, the game can receive commands (in any order) without crashing. The following commands should **print the first line of output correctly:**
    - `HELP`
    - `QUIT`
    - `LOOK` and `L`
    - `INV` (Since the user has no way to receive any items, you only have to account for the case where the user is carrying nothing).
    - `CHECK`

1 mark will be awarded for submitting a suite of three *test cases*. Make a test case for each of the following cases:

- There are 3 quests to complete. It is possible to complete all 3 quests, but only if you do them in a specific order.
- There are 3 quests to complete. It is possible to complete one of the quests, but due to the requirements of the other two, no more can be completed.
- There are 3 quests to complete. It is possible to complete all 3 quests and end with a `WILL` value of `0`.

Academic declaration

By submitting this assignment, you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.