Project 4: Text Adventure

Getting Started

First, you should play some Text-based Adventure Games:

* [Spooky House](https://canvas.vt.edu/courses/76594/pages/example-spooky-house) is a simplistic game where you explore a creepy house.
* [Finding Adventure](https://canvas.vt.edu/courses/76594/pages/example-finding-adventure) is a more complex game where you are looking for something you forgot.
* [Zork (Links to an external site.) Links to an external site.](http://www.web-adventures.org/cgi-bin/webfrotz?s=ZorkDungeon)is a professional, classic game where you explore the ruins of an ancient empire.

Second, think critically about the kind of game you want to make. Keep the focus small and manageable. You are free to redesign your game later, but planning up-front will make it easier to develop.

Third, you should create a diagram mocking out your player's state and game's locations, and upload it to this other assignment: [Project 4: Text Adventure Diagrams](https://canvas.vt.edu/courses/76594/assignments/484271)

Fourth, download this starting code and resources: [cs1064-f18-p4\_text-adventure-v8.zip](https://canvas.vt.edu/courses/76594/files/7578981/download?download_frd=1)

Player State

A critical concept in this project is the Player's State. The game works in a while loop that repeatedly changes the current player's state, depending on the commands they choose. You will model the player's state using a Dictionary with at least 4 fields.

The Player State Dictionary has two required fields:

* 'game status', which will always be one of the following string values: 'playing', 'quit', 'won', or 'lost'
* 'location', which will always have a string value indicating the player's current position.

The 2+ other fields are up to you, but some general ideas are:

* Inventory, a list of strings or dictionaries representing collected items
* Score, an integer field representing your progress in the game
* Health, an integer field representing how much damage you can take
* Event flags, boolean fields for indicating whether a particular event has occurred

You are encouraged to build a rich abstraction unique to your game.

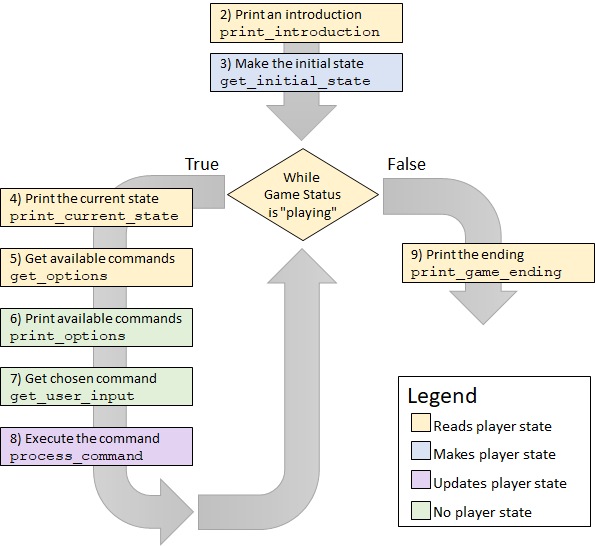
Completing the Project

This project is worth a total of 25 points, which come from three categories:

* 8 points come from writing 8 functions that pass the unit tests on Web-CAT.
* 3 points come from meeting the project's functional requirements on Web-CAT.
* 9 points come from TA review of the code's style and organization.
* 5 points come from TA review of the game's creative elements.

The Functions (8 points)

You need to implement a total of 8 functions to make your game, and define two variables..



The diagram above shows the control flow of the code. Square boxes are functions that you will need to define, and their color indicates whether they read, make, update, or have no interactions with the main Player State Dictionary.

1. main: This function is given to you. Read over its implementation very carefully to fully understand how all the game components work together.
2. print\_introduction: This function consumes nothing, takes no user input, and returns nothing. Instead, it is responsible for printing an introductory message to introduce your game. There are no requirements on what it can print, except that you must print something.
3. get\_initial\_state: This function consumes nothing, takes no user input, and prints nothing. Instead, it must return a dictionary representing the initial state of the player. Remember that one of the field's of this dictionary must be 'game status', and its value should initially be 'playing'.
4. print\_current\_state: This function consumes the current player's state dictionary and prints out information about it. It takes no user input and returns nothing. The only required thing to print is the player's current 'location'. It doesn't matter what you print alongside it, as long as the field's value gets printed. After that, you should print out some other supplementary text to describe your player's current state. There is no maximum amount of text, but you must print out something relevant to the player's state. For instance, if your game had a 'poisoned?' mechanic, you might print out, *"You are poisoned! You need to find an antidote or you will collapse."* Or if you had an inventory mechanic, you might print out each item of the inventory.
5. get\_options: This function consumes the current player's state dictionary and returns a list of strings representing commands available to the user right now (e.g., places to move to, commands to execute, things available to be collected). It should take no user input and prints nothing. You might organize this using if/else statements, dictionaries, or helper functions. For example, you could check the player's location and offer them other locations to move to. You should never return an empty list from this function.
6. print\_options: This function consumes the list of strings (commands) returned by the result of get\_options and prints each one on its own line. You are free to indent each command and add decorative text (e.g., bullets) as you see fit. The function takes no user input and returns nothing.
7. get\_user\_input: This function consumes the list of strings (commands) returned by the result of get\_options and let's the user input which command they want to choose. The function must repeatedly loop internally until the user types in one of the valid commands OR the user types "quit". The function should then return the user's inputted command. Notice that this function takes user input, takes an argument, returns a value, and can print if you want it to (you are free to print whatever you'd like). Also note that the function has its own loop separate from the main game loop (it should be similar to code you've written for the lesson on While loops!).
8. process\_command: Given a command (string) and a player' state (dictionary), modify the dictionary so that the player's state is changed. Notice that this function does not return anything, but instead mutates the given dictionary. You are free to make it print, if that is relevant to your game, but it is not required. At the minimum, this function should change the 'game status' field to 'quit' if the player quits, and should also have some way of changing the 'game status' to 'win' or 'lose'. Most likely, this is where you would have the character update their location based on their command, among many other possible updates.
9. print\_game\_ending: This function consumes the current player's state dictionary and prints out a different message depending on whether they won, lost, or quit the game. You are free to customize this message however you see fit (perhaps having different kinds of victory conditions), but you must print out a different message for the three possible outcomes. You must always print out something, no matter what game state is passed in. This function returns nothing and takes no user input.

Developing a game can be tricky. We recommend that you write and try out each function independently (e.g., by writing your own test cases). However, you could instead comment out parts of the main function that you haven't reached yet. A third option is to write in an "Agile" style, always having a "minimally working game" that you slowly and incrementally build.

Functional Requirements (3 points)

There should be AT LEAST three ways for your game to end: winning, losing, or quitting. You will need to codify the exact steps needed to win or lose the game, such that a computer inputting those commands could win the game. Your game must be deterministic - anyone who inputs those commands should get the same outcome.

You will need to define two variables:

1. WIN\_PATH: A list of strings representing commands that will be given to your game, in order to reliably achieve victory.
2. LOSE\_PATH: A list of strings representing commands that will be given to your game, in order to reliably lose the game.

Web-CAT will try out your game to achieve the three different outcomes (quitting, losing, and winning).

Code Organization (9 points)

More than previous assignments, this project requires you to focus on good code organization. You should pay close attention to the [Style Guide](https://canvas.vt.edu/courses/76594/pages/style-guide). Think carefully about each of the following:

1. Good variable names
2. Documenting functions
3. Helper functions to break up long functions (You might want to refer to [Strategies for Dealing with a Long Function](https://canvas.vt.edu/courses/76594/pages/strategies-for-dealing-with-a-long-function))
4. Helper functions to avoid redundant code
5. Avoiding extreme nesting
6. Avoiding long lines
7. Using consistent style
8. Cleaning up messy code
9. Removing unused imports

Creativity (5 points)

You are encouraged to be creative when designing your game. You are free to focus on narrative, interesting gameplay mechanics, and including things like Ascii art. Of course, you should ensure you meet the minimal programming and functional requirements first. Although it would be quite possible to complete this assignment in twenty additional lines of code, you should see this as an opportunity to develop a larger-scale program and to express yourself.

1 point for this category come from a TA evaluation that a minimal amount of effort was put into making the project. Much like the Turtle Project, this is a loose evaluation. If you finish the project in 10 minutes, you probably did not do enough. Try to make something that you wouldn't be afraid to show a loved one. If you demonstrate some creative effort in good faith, you will most likely earn these points.

One clear way to lose this point is to create unused or unmodified fields in the player state, resulting in less than the minimal number of fields (4 total).

* Unused: A field is mentioned in the player state, but is not ever read in the game.
* Unmodified: A field is mentioned in the player state, but is not ever changed during the game.

4 points for this category come from a TA evaluation of the Player State and Game Location's diagrams you made when starting the project. Although it does not have to 100% reflect your final version, it should be very close to what you create. See the other assignment for more details on these diagrams: [Project 4: Text Adventure Diagrams](https://canvas.vt.edu/courses/76594/assignments/484271)