**Adventure Game Project**

In this assignment, you will develop a text adventure game (TAG), also known as interactive fiction. The characteristic elements of TAGs include gameplay driven by exploration and puzzle-solving, and a text-based interface in which users type natural-language commands and the game responds with text. The classic example of this genre is ZORK, by the company Infocom.

For this assignment, you are to implement a game engine that could be used to play many adventures. Here, the game engine is an OCaml program that implements the gameplay and user interface. You will build a data set that describes a particular gaming experience: exploring a cave, hitchhiking on a spaceship, finding the missing pages of a powerful magical book, etc. This factoring of responsibility between the engine and data set is known as data driven design in games.

The gameplay of TAGs is based on an adventurer moving between rooms through corridors. Rooms might represent actual rooms, or they might be more abstract—for example, a room might be an interesting location in a forest, and a corridor might be the path leading between such locations. Normally corridors can be traversed in both directions, but not always. Each room also has a text description associated with it. Some rooms have items in them. These items can be taken by the adventurer and carried to another location, where they can then be dropped. The adventurer begins the game in a predetermined room, possibly with some predetermined items.

The player does not so much win a TAG as complete the TAG by accomplishing various tasks: exploring the entire map of rooms and corridors, finding items, moving items to specified locations, etc. To indicate the player's progress toward completion, a TAG gives a player a numeric score at each moment in the game. The TAG also tracks the number of turns taken by the player. Savvy players attempt to achieve the highest score with the lowest number of turns.

The interface to a TAG is based on the player issuing text commands to a prompt; the game replies with more text and a new prompt, and so on. Thus, the interface is a kind of read-eval-print-loop (REPL), much like the OCaml top level. For this assignment, commands will generally be two-word phrases of the form VERB OBJECT:

* **Movement** is performed by using the verb "go" followed by the direction. The room itself determines what the allowed directions are; common possibilities include "north", "south", "east", "west", "up", and "down". If such movement is possible, the movement occurs, the adventurer is in a different room, and the description of that room is displayed. If the movement is impossible, an error message is displayed. As a shorthand, the player may simply type the direction itself without the verb "go".
* **Items** can be manipulated by the verbs "take" and "drop" followed by the name of the item: "take" transfers an item from the current room to the adventurer's *inventory*, and "drop" transfers an item from the inventory to the current room. Of course, if an item is present in a room, it must be mentioned when the room is described.
* **Other commands** include "quit", which ends the game, "look", which re-displays the description of the current room, "inventory" (shorthand: "inv"), which gives a list of what the adventurer is currently carrying, "score", which displays the current score, and "turns" which displays the current number of turns.

For this assignment, scoring will be simple: every room is worth a certain number of points, which are earned simply for entering the room at least once, and every item is worth a certain number of points. The points for an item are earned if the item is in a particular room, which depends on the item. Dropping the item in that room earns points, and taking the item away from that room loses points. Both cause the engine to print a message telling the player so. When the player earns the maximum score possible for the adventure, the game engine informs the player that they have completed the adventure.

And, for this assignment, you may choose your own definition of what counts as a turn. As a starting point, we recommend that each "go", "take", and "drop" count as a turn.

Your task is to develop a game engine and to write a small adventure of your own.

**Objectives**

1. Design user-defined data types, especially records and variants.

2. Write code that uses pattern matching and higher-order functions on lists and on trees.

3. Interact with the environment outside the program by reading and writing information from the user.

4. Practice writing programs in the functional style using immutable data.

**Requirements**

The primary requirements are the following:

1. Your game engine must implement all the functionality described in the overview above.

2. Your game engine should be compatible with the adventure format below, so I can test your engine on my own adventure.

3. Your game engine should be robust: there should be no unhandled exceptions, infinite loops, or other faulty behavior.

4. Your own small adventure must have at least 5 rooms and 3 items.

5. Your code must be written with good style and be well documented.

6. You may use ONE OCaml loop.

**Design Hints**

**Model:** One of the first things you should do is design the data type(s) you will use to capture the state of the game: what the map is, where the adventurer is located, what items are in the inventory, etc. Those data type(s) are called the model of the game. It will be difficult to implement your game engine if you repeatedly make changes to the model during the implementation.

**REPL:** Think carefully about how to structure the main read-eval-print "loop" of your game: keep the amount of code in it small by defining and calling additional functions. Sketch out how you will implement the eval phase with an eye toward whether your model is appropriate.

**Implementation Hints**

Text interface: All the I/O functions you need are in the Pervasives module. The Printf module might be helpful for output. The Scanf module is probably higher power than necessary for input in this assignment.

Parsing player commands: Although you are unlikely to need any functionality outside of the standard String module, you are welcome to use the OCaml Str library.

**Testing Hints**

Spend time developing a plan as to how you will test your game engine.

Model: Write tests for functions that update your model, e.g., functions that implement the commands. You could hardcode some models and commands for testing purposes, rather than relying on parsing code.

Parsing: Write tests for your player-command parser to ensure that verbs and objects are correctly understood. Create small adventure data sets to test your parser. Note: debugging parsers is hard.

Adventures: Write small adventures to test specific commands and functionality, e.g., the "go" command, the "take" and "drop" commands, etc. In the best case, write code that automates the running of these adventures, to frequently check whether any changes to your implementation have broken these tests.

Playtesting: When your implementation is reaching maturity, get your friends to play an adventure using your game engine. A good game that no one can learn to play is a bad game!

What do your playtesters say about the game?

Where and how do they run into trouble?

Bear in mind not to coach them while they play.

Don't take any criticism personally.

Remind them to think out loud as the play—you want to know everything going on in their head.

Gameplay will be at most a small part of the evaluation of your engine. But bear in mind that really poor gameplay could make it difficult or impossible for the grader to evaluate your engine.

**Adventure Format**

According to the schema, an adventure file contains four main entries:

The rooms. Each room contains six entries:

an id,

a description,

the items in it,

the number of points exploring the room is worth,

the exits from the room (an exit itself contains two entries: the direction of the exit, and the room to which it leads), and

the treasures that should be dropped in the room.

The items. Each item contains three entries:

an id,

a description, and

the number of points the item is worth when it is dropped in its designated room.

The starting room (where the adventurer begins).

The starting items (which are initially in the inventory).

**Extra Credit**

If you finish the assignment, you can add the following features:

1. Figure out how to run an Ocaml program from a command line. Although the OCaml standard library provides a full-featured Arg module for parsing command-line arguments, you won't need it for this assignment. Rather, the following expression evaluates to a string list containing all the command-line arguments: Array.to\_list Sys.argv. The first element of the list will be the name of the program as it was invoked; the second element is where you should expect to find the name of the JSON adventure file to be loaded. Using this expression will not be construed as violating the prohibition on imperative data structures, even though it uses the Array module. In fact, the expression does not cause any mutation or side effects.

2. Learn about the data format JSON, and migrate your adventure data into a JSON file. There is a chapter of Real World OCaml about JSON. You will need to learn about reading files in and figure out how to create your data structure from the data you read in.