(CSCI 421 VA) Project Six: The Adventure Game

Refer to Exercise Six on page 444.

Add the following features to the adventure game from chapter 20 of the textbook.

1. There is a gate between the fork in the path and the mountaintop. The gate is a separate location; that is, the player must move from at (you, fork) to at (you, gate) and then to at (you, mountaintop).
2. To move forward through the gate, the player must first unlock it with a key.
3. The key is somewhere in the maze. The player must find it and explicitly pick it up
4. If the player tries to pass through the gate while still holding the key, he or she is killed by lightning. (To get the treasure, the player must first open the gate, then put down the key and then pass through)

Start from the code in this chapter, which is copied below (it is also on this book's web site. A link is provided in this module's folder). Part of your implementation should be a general way for the player to pick thing up, carry them, and put them down. Design your solution so that it would be easy to add additional objects for the player to pick up and put down.

```
/*
  This is a little adventure game.  There are three
  entities: you, a treasure, and an ogre.  There are
  six places: a valley, a path, a cliff, a fork, a maze,
  and a mountaintop.  Your goal is to get the treasure
  without being killed first.
*/

/*
  First, text descriptions of all the places in
  the game.
*/
description(valley,
  'You are in a pleasant valley, with a trail ahead.').
description(path,
  'You are on a path, with ravines on both sides.').
description(cliff,
  'You are teetering on the edge of a cliff.').
description(fork,
  'You are at a fork in the path.').
description(maze(_),
  'You are in a maze of twisty trails, all alike.').
description(mountaintop,
  'You are on the mountaintop.').

/*
  report prints the description of your current
  location.
*/
report :-
  at(you,X),
  description(X,Y),
```

```
  write(Y), nl.

/*
  These connect predicates establish the map.
  The meaning of connect(X,Dir,Y) is that if you
  are at X and you move in direction Dir, you
  get to Y.  Recognized directions are
  forward, right, and left.
*/
connect(valley,forward,path).
connect(path,right,cliff).
connect(path,left,cliff).
connect(path,forward,fork).
connect(fork,left,maze(0)).
connect(fork,right,mountaintop).
connect(maze(0),left,maze(1)).
connect(maze(0),right,maze(3)).
connect(maze(1),left,maze(0)).
connect(maze(1),right,maze(2)).
connect(maze(2),left,fork).
connect(maze(2),right,maze(0)).
connect(maze(3),left,maze(0)).
connect(maze(3),right,maze(3)).

/*
  move(Dir) moves you in direction Dir, then
  prints the description of your new location.
*/
move(Dir) :-
  at(you,Loc),
  connect(Loc,Dir,Next),
  retract(at(you,Loc)),
  assert(at(you,Next)),
  report,
  !.
/*
  But if the argument was not a legal direction,
  print an error message and don't move.
*/
move(_) :-
  write('That is not a legal move.\n'),
  report.

/*
  Shorthand for moves.
*/
forward :- move(forward).
left :- move(left).
right :- move(right).

/*
  If you and the ogre are at the same place, it
  kills you.
*/
ogre :-
  at(ogre,Loc),
  at(you,Loc),
```

```
  write('An ogre sucks your brain out through\n'),
  write('your eye sockets, and you die.\n'),
  retract(at(you,Loc)),
  assert(at(you,done)),
  !.
/*
  But if you and the ogre are not in the same place,
  nothing happens.
*/
ogre.

/*
  If you and the treasure are at the same place, you
  win.
*/
treasure :-
  at(treasure,Loc),
  at(you,Loc),
  write('There is a treasure here.\n'),
  write('Congratulations, you win!\n'),
  retract(at(you,Loc)),
  assert(at(you,done)),
  !.
/*
  But if you and the treasure are not in the same
  place, nothing happens.
*/
treasure.

/*
  If you are at the cliff, you fall off and die.
*/
cliff :-
  at(you,cliff),
  write('You fall off and die.\n'),
  retract(at(you,cliff)),
  assert(at(you,done)),
  !.
/*
  But if you are not at the cliff nothing happens.
*/
cliff.

/*
  Main loop.  Stop if player won or lost.
*/
main :-
  at(you,done),
  write('Thanks for playing.\n'),
  !.
/*
  Main loop.  Not done, so get a move from the user
  and make it.  Then run all our special behaviors.
  Then repeat.
*/
main :-
  write('\nNext move -- '),
```

```
  read(Move),
  call(Move),
  ogre,
  treasure,
  cliff,
  main.

/*
  This is the starting point for the game.  We
  assert the initial conditions, print an initial
  report, then start the main loop.
*/
go :-
  retractall(at(_,_)), % clean up from previous runs
  assert(at(you,valley)),
  assert(at(ogre,maze(3))),
  assert(at(treasure,mountaintop)),
  write('This is an adventure game. \n'),
  write('Legal moves are left, right, or forward.\n'),
  write('End each move with a period.\n\n'),
  report,
  main.
```