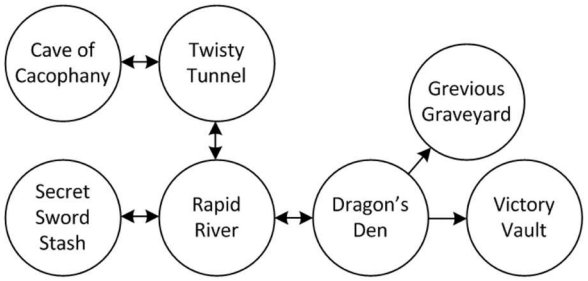
# Adventure Game FSM

## Problem Statement

Design an adventure game with seven rooms and one object (a sword). The game begins in the Cave of Cacophony. To win the game, you must first proceed east through the Twisty Tunnel and south to the Rapid River. From there, you will need to find a Vorpal Sword in the Secret Sword Stash to the west. The sword will allow you to pass east from the Rapid River through the Dragon Den safely into Victory Vault (at which point you have won the game). If you enter the Dragon Den without the Vorpal Sword, you will be devoured by a dangerous dragon and be tossed into the Grievous Graveyard (where the game ends with you dead). The game remains in the Graveyard or Vault until you reset it. The game map is shown below.




This game can be implemented using **two separate state machines** that **communicate** with each other. The **room state machine** keeps track of which room you are in, while the **sword state machine** keeps track of whether or not you currently have the sword. The machines must communicate back and forth.

- Sketch a state transition diagram for each of the FSMs. The state machines should receive an asynchronous reset and a clock. The system inputs are N, S, E, and W, corresponding to the four directions. The system should produce two outputs, WIN and DIE. The room FSM should have one state for each room and transitions based on the directions you could move. **Assume** the player will apply exactly one input each cycle and will never apply an invalid input.

- Write **behavioural SystemVerilog** for your system. Be sure to think about the hardware you want and write the appropriate idiom rather than approaching this like a programming exercise. You should have **one module for each FSM** and one **top-level module** connecting the two together.

When you describe the combinational logic for the next state, remember that you must always specify a next state. The assumption that the player applies a valid input every cycle allows for some simplification. For example, when you are in the Cave of Cacophony, the next state will necessarily be the Twisty Tunnel.

Run your system with the self-checking test bench provided in adventuresome.sv below. View waveforms all of the important information, including the clock, reset, inputs, outputs, and state of each FSM. Remember that to see waveforms that are not at the top level of the hierarchy, you will need to browse to the appropriate cell in the hierarchy in the upper left Structure pane to see signals within that cell. Expand ***testbench*** to see *dut*, then expand that to see other submodules such as the *room* and *sword* FSMs. For example, add the internal signal indicating the current room to your waveforms.

Debug any errors.