Legal Move Generators

There are many ways to write legal move generators (LMGs). For the numbers game, ie: how can you get from one number to another (for example from 27 to 154) using the following rules (i) you can multiply by 3 (ii) you can subtract 7 (iii) you can add 10.

A LMG for this problem could be...

```
(defn lmg1 [n]
  (list (* n 3)
        (- n 7)
        (+ n 10)
        ))
```

To use this LMG, you will need to load the search routine. The search mechanism is called *breadth-search*, it's called with 3 arguments: start-state, goal-state and the LMG function.

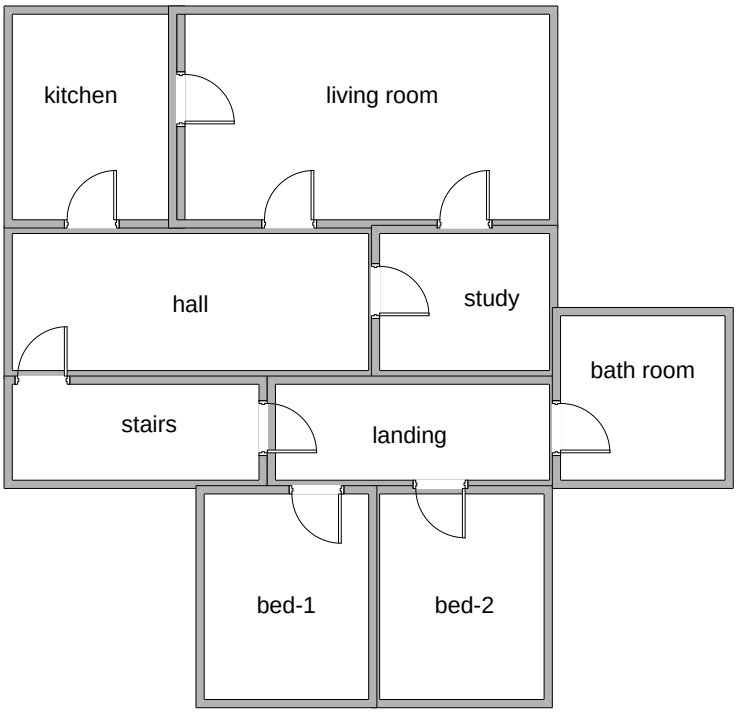
```
user=> (breadth-search 27 154 lmg1)
(27 37 47 141 134 144 154)
```

We can demonstrate an alternative approach with the words problem (as long as we assume a very small dictionary). The words problem is concerned with getting from one word to another, changing one letter at a time & always spelling a word. The LMG is a map (or an implicit function using a map).

```
(def words
  '{coat (boat moat cost)
    cost (most lost coat cast)
    boat (moat coat boot)
    moat (moot most boat)
    moot (soot boot loot)
    lost (last cast loot)
     ;;...etc...
     })


user=> (breadth-search 'boat 'last words)
(boat coat cost lost last)
```

The diagram below shows the layout of a house.



Think about how you could write a legal move generator to drive a search routine which can provide routes around the house for a robot, ie: advise the robot on the route between the study & the bath room, etc.

Write your LMG & try it out with the search routine.

Once you have a working LMG, think about how you could extend the problem by adding objects, eg: an apple in the bathroom, a book in the study. You should consider how to extend your representation of the house (to include objects as well as rooms) and how to modify your legal move generator to deal with picking up & dropping objects as well as moving between rooms. NB: you can assume that the robot can only hold one object at a time.