

# STAGE 6 – THE QUEEN ALIEN

When the player enters a module occupied by the queen alien, it will attempt to escape by making one, two or three moves to adjacent modules. The queen cannot go past the player to enter the module the player has just entered from. However, if it is possible for the queen to double back behind the player, that is a valid move. If the queen enters a module containing a ventilation shaft, it travels to a random module and stops moving, irrespective of whether it had moves left to make.

The queen cannot enter a locked module. If the queen has nowhere to escape to, the game is won, although the final boss battle is still to be programmed.

This is the most complex section of code, providing an excellent opportunity to discuss problem decomposition, algorithmic thinking and testing.



Add the following procedure to your code. Please note, an arrow indicates that the line is not new but, rather, a continuation of the one before.

```
def move_queen():
    global num_modules, module, last_module, locked, queen, won, vent_shafts
    #If we are in the same module as the queen...
    if module == queen:
        print("There it is! The queen alien is in this module...")
        #Decide how many moves the queen should take
        moves_to_make = random.randint(1,3)
        can_move_to_last_module = False
        while moves_to_make > 0:
            #Get the escapes the queen can make
            escapes = get_modules_from(queen)
            #Remove the current module as an escape
            if module in escapes:
                escapes.remove(module)
            #Allow queen to double back behind us from another module
            if last_module in escapes and can_move_to_last_module == False:
                escapes.remove(last_module)
            #Remove a module that is locked as an escape
            if locked in escapes:
                escapes.remove(locked)
            #If there is no escape then player has won...
            if len(escapes) == 0:
                won = True
                moves_to_make = 0
                print("...and the door is locked. It's trapped.")
```

0 1 2 3 4  
tabs tab tabs tabs tabs



## Programming project: Telium

```
#Otherwise move the queen to an adjacent module
else:
    if moves_to_make == 1:
        print("...and has escaped.")
        queen = random.choice(escapes)
        moves_to_make = moves_to_make - 1
        can_move_to_last_module = True
    #Handle the queen being in a module with a ventilation shaft
    while queen in vent_shafts:
        if moves_to_make > 1:
            print("...and has escaped.")
            print("We can hear scuttling in the ventilation shafts.")
            valid_move = False
            #Queen cannot land in a module with another ventilation
            shaft
            while valid_move == False:
                valid_move = True
                queen = random.randint(1,num_modules)
                if queen in vent_shafts:
                    valid_move = False
            #Queen always stops moving after travelling through shaft
            moves_to_make = 0
```

0 1 2 3 4 5 6 7  
tabs tab tabs tabs tabs tabs tabs tabs





In the main program section, underneath `check_vent_shafts()`, add:

```
move_queen()
```



# Programming project: Telium

## Investigate

<p><b>ITEM</b></p> 	<p>Q: Identify the type of iteration statement being used in the <code>move_queen()</code> subroutine.</p>
<p><b>STRUCTURE</b></p> 	<p>Q: What do we mean by problem decomposition? How is problem decomposition applied to this procedure to ensure it can be programmed successfully?</p>
<p><b>PURPOSE</b></p> 	<p>Q: What is the value of commenting large sections of code like this?</p>
<p><b>REASON</b></p> 	<p>Q: Why is a <code>while</code> loop known as a condition-controlled loop? Why is a condition-controlled loop being used over a count-controlled loop?</p>
<p><b>RELATION</b></p> 	<p>Q: By placing each different type of object in its own unique, named list, we are able to write lines such as:</p> <pre>while queen in vent_shafts:</pre> <p>What does this line do, and how has it made the program easier to understand?</p>
<p><b>APPROACH</b></p> 	<p>Q: How could we modify the <code>move_queen()</code> subroutine to allow the queen to travel from a room with a ventilation shaft to another room with a ventilation shaft?</p>



## Programming project: Telium

### Make

1 point

Add a comment above the `move_queen()` subroutine that summarises the purpose of the new module.

1 point

If the flamethrower contains less than 100 units of fuel, output a “Low fuel” warning on each turn.

2 points

Draw up a test plan with sample data that could be used to test the procedure that moves the queen alien.

3 points

The game immediately ends when the queen alien is trapped. Add a final boss battle where the player has to use all the fuel in their flamethrower in an attempt to kill it. The queen requires 100 to be killed as a minimum. If there is insufficient fuel, the player is killed.

3 points

In step 2, you may have considered putting some extra objects into the game – program these now.

### Evaluate

1. You should always white-box test your programs against a range of valid inputs to check that all the various paths through a subroutine are working correctly. Complete the test table for the `move_queen` subroutine. You can add and delete columns and rows as necessary.

What is being tested?	Expected outcome	Pass/Fail

