

STAGE 2 – ALLOWING THE PLAYER TO MOVE AROUND THE SPACE STATION

The program will read the data for the module the player is currently in from the associated text file to determine the valid moves they can make. A zero in the text file represents no adjacent module.

The player can be given a map of the space station when playing the game – it is only reasonable that an astronaut aboard the station would know their way around. However, in the panic and confusion, they may not remember where objects are, so these are hidden from the player.



Create a new program in the folder above the `Charles_Darwin` folder to set up basic navigation for the game. Be very careful with the indentation – you will introduce logic or syntax errors if you make a mistake.

```
#Telium - The game

import random

#Global variables

num_modules = 17           #The number of modules in the space station
module = 1                 #The module of the space station we are in
last_module = 0            #The last module we were in
possible_moves = []        #List of the possible moves we can make
alive = True               #Whether the player is alive or dead
won = False                #Whether the player has won
power = 100                #The amount of power the space station has
fuel = 500                 #The amount of fuel the player has in the flamethrower
locked = 0                 #The module that has been locked by the player
queen = 0                  #Location of the queen alien
vent_shafts = []           #Location of the ventilation shaft entrances
info_panels = []           #Location of the information panels
workers = []               #Location of the worker aliens

#Procedure declarations

def load_module():
    global module, possible_moves
    possible_moves = get_modules_from(module)
    output_module()

def get_modules_from(module):
    moves = []
    text_file = open("Charles_Darwin\module" + str(module) + ".txt", "r")
```

0 1
tabs tab



Programming project: Telium

```
    for counter in range(0,4):
        move_read = text_file.readline()
        move_read = int(move_read.strip())
        if move_read != 0:
            moves.append(move_read)
    text_file.close()
    return moves

def output_module():
    global module
    print()
    print("-----")
    print()
    print("You are in module",module)
    print()

def output_moves():
    global possible_moves
    print()
    print("From here you can move to modules: | ",end='')
    for move in possible_moves:
        print(move,| ',end='')
    print()

def get_action():
    global module, last_module, possible_moves
    valid_action = False
    while valid_action == False:
        print("What do you want to do next ? (MOVE, SCANNER)")
        action = input(">")
        if action == "MOVE":
            move = int(input("Enter the module to move to: "))
            if move in possible_moves:
                valid_action = True
                last_module = module
                module = move
            else:
                print("The module must be connected to the current module.")

#Main program starts here

while alive and not won:
    load_module()
    if won == False and alive == True:
        output_moves()
        get_action()







if won == True:
    print("The queen is trapped and you burn it to death with your flamethrower.")
    print("Game over. You win!")
if alive == False:
    print("The station has run out of power. Unable to sustain life support, you die.")
```

0 1 2 3 4
tabs tab tabs tabs tabs



Programming project: Telium

Investigate

ITEM 	Identify the variable that controls whether you have won the game or not.
STRUCTURE 	A number of functions are using the def command – why?
PURPOSE 	Why is the <code>global</code> command needed?
REASON 	Explain how input has been validated.
RELATION 	Explain the advantages and disadvantages of using the command: <code>print("Game over. You win!")</code> instead of: <code>txt = "Game over. You win!" : print(txt)</code>
APPROACH 	<code>num_modules</code> is a constant. What does that mean, why have we chosen to use a constant, and what are the advantages of using constants?



Programming project: Telium



1 point

Comment the functions so each is explained.

1 point

The power variable should decrement by one after each move to add a time limit to the game. If the power variable equals 0, the player dies due to a lack of life support.

1 point

Create a map of the station in a suitable application that a player can use to navigate in your game.

2 points

Improve the input sanitisation so the player does not need to enter **MOVE**. Lowercase and abbreviation to **M** should be accepted too.

2 points

Create your own space station configuration by modifying the text files.

2 points

Add an extra line to the module files that contains a name for the module and outputs it to the screen – e.g., “You are in module 1 – this is the communication module where astronauts contact Earth.”

2 points

Add a title screen that allows the player to choose Play, Story, Instructions or Quit.

2 points

Add a screen that loads a page of instructions on how to play the game.

2 points

Add extra text to the module files to output a description of the module to the player.

2 points

Add an aspect of the story that the player can read.

3 points

Create a simple text parser for the game so the player can enter **MOVE** and the module in the same line – e.g., an input of **MOVE 2** would move the player to module 2.

3 points

Further extend this to work with a range of input sanitisation options – e.g., **M2** and **move2** have the same effect. Don't forget to update your instructions screen with these new input options.



Programming project: Telium

Evaluate

1. What methods and techniques have been used to make your current program robust and easily maintainable?
2. What additional work could you carry out to make your program even more robust?

