Project 1: Cave Cartographer

This first section describes the entirety of the program you will be building. Only portions of it will be due at a time, on days called *Checkpoints*. To see what parts of it are due at which times, see Part I, Part II, and Part III below.

**Table of Contents**

[**The End Goal 2**](#_roh1n8r34nza)

[Game rules 2](#_jpb19s9sudsp)

[Specifications 2](#_s659i6m3mv4r)

[**Part I: Unit Tests 5**](#_syt6h2y492jo)

[Plan Tests 5](#_62p2he8whgrx)

[Test\_Cave\_1 6](#_j6q3pmeakjen)

[Starting Set-Up 6](#_vaerxseb3wk6)

[Note that the caret < indicates the current spot of the adventurer. 6](#_cl744m1l11nv)

[can\_move tests 1 6](#_ohrioa2rw4gh)

[move tests 1 7](#_tyglszrthvgo)

[We do not need additional unit tests for move on test\_cave 1 because the move function assumes a valid move and there are no other valid moves for this cave layout. 7](#_fo2n4xuncjv8)

[Test\_Cave\_2 7](#_t3lblnc5y08p)

[Layout 7](#_uthtixai6gkf)

[Starting Set-Up 7](#_3pbeae2jzbim)

[can\_move tests 2 7](#_nuld6wqu6w9v)

[move tests 2 8](#_wbctji3pg29b)

# The End Goal

**The end goal of this project** is to have a program that puts the user into an unexplored cave, and through a text UI allow the user to explore and map out the cave.

## Game rules

* The player can only move in the direction of an empty space, object, or the initial starting point.
* The player can only see the parts of the cave that have been mapped out (or *visited*).
* The player can only exit the cave when the whole cave has been explored.

## Specifications

There are two classes involved in this project: the *Cave* class and the *Adventure* class. All of the background functional aspects of the program (i.e. appear as methods in one of these two classes. All other aspects of the program, including any user input and text output must appear in the \_\_main\_\_ part of the program.

# ~ MAIN ~ #

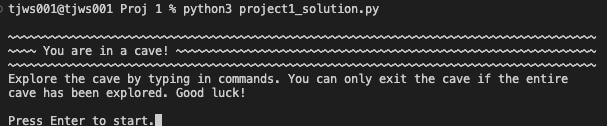
if \_\_name\_\_ == "\_\_main\_\_" :

# Code here

Running the program should do approximately the following on the user’s end.

1. Displays an ASCII art introduction screen introducing the title and author names on the project.
2. Displays a set of instructions explaining how to move around the cave and the purpose of the game.
3. Displays an ASCII art cave that indicates where you start and what is in your immediate surroundings.
4. Displays the valid commands for their current position, including the directional commands—“left”, “down”, “right”, and “up”---for the directions they can move in, as well as “quit”.
5. Based on user input, either moves the player one space in that direction, or quits the game.
6. If the user is in the Escape space and the cave has been fully explored, it prints a victory message and quits.

When the program runs, it should look a bit like this:



When you start your adventure, the program loads a text file representing an unexplored cave into a 2D Python list and creates a *Cave* object out of it (an instance of the *Cave* class). The text file I have loaded into the program below is this one:

R R R R R R R R R R R R R R

R \_ \_ \_ R \_ \_ \_ \_ \_ R \_ \_ R

R \_ R \_ R \_ R \_ R \_ R \_ R R

R \_ R \_ R \_ R \_ R \_ R \_ R R

R \_ R \_ R \_ R \_ \_ \_ \_ \_ \_ R

R S R \_ \_ \_ R \_ R E R R \_ R

R R R R R R R R R R R R R R

Above,

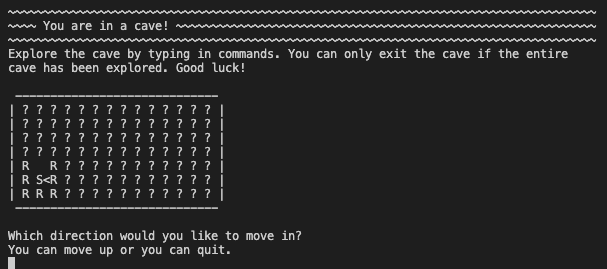
S means Start

R means Rock

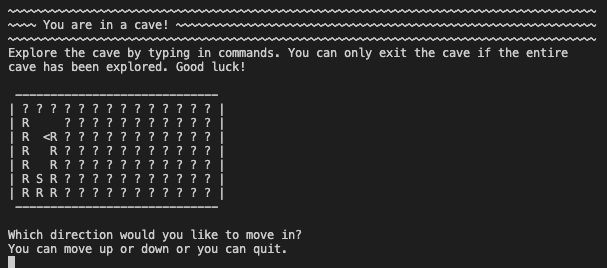
\_ means empty space

E means Exit

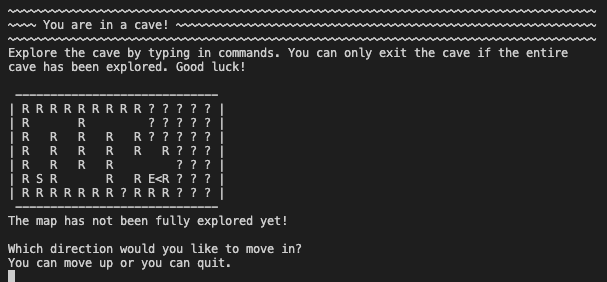
The text file **must** be a rectangle of the above letters with exactly one S and exactly one E. The program starts the adventure by creating a new *Adventure* object, which tracks information about where the adventurer currently is and how much of the cave has been explored. You can represent the cave how you like, but mine looks like this:



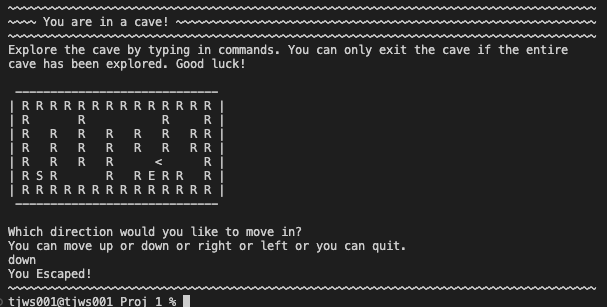
The ?s represent parts of the cave that I have not yet explored. For example, if I move up 3 times, then my cave map will look like this:



If I reach the exit but the map hasn’t been explored, I see this:



Once I have moved through the whole cave, if I reach the exit, I get this:



**Reflection Question:** Are there “bad” caves? That cannot be fully explored?

# Part I: Unit Tests

This is a project with many moving parts, and requires quite a bit of planning. The first step in the project (the first checkpoint) is to plan out what properties the fundamental methods of the game should satisfy.

Frequent, incremental testing helps ensure our program works. It also helps us find errors earlier and easier. Unit testing is a type of testing that focuses on trying the functionality of a small component of your code and ensuring it behaves properly. Practicing a bit of Test Driven Development, you will write a set of unit tests for part of your cave exploration code *before* writing any code of the actual game. This process helps solidify your understanding of how the code *should* behave and gives you a mechanism to check your progress as you go.

|  |
| --- |
| **Your assignment**  **Plan and write unit tests for the *can\_move* and *move* method in the *Adventure* class.** These should be written into a file called **cave\_unit\_test\_move\_2.py** and should use the ***TestCase*** class in the ***unittest*** Python package. |

This is a great opportunity to practice using unit testing as a way of planning out large projects that have many moving parts. You are encouraged to write Unit Tests for methods throughout the project to help guide your development.

## Plan Tests

What you are given:

* one example test cave layout
* one plan for testing *can\_move* and *move* in this layout (see the provided Python file cave\_unit\_test\_move\_1.py)
* Stubs for the *Cave* and *Adventure* classes (do not touch or submit these—they are there just so you can see the general outline of what you are testing).

|  |
| --- |
| **You need to:**   * create at least one additional test cave to more comprehensively test *can\_move* and *move* (e.g., adding open spaces and rocks)   + aim to create the simplest layout that will cover your test   + you may create as many layouts as you'd like * complete the tables below for planning out test cases   + each test case may require more steps than the example   + you may need more tests per cave than the example   + use/add as many rows to the table as you need * write these cases using the TestCase python class, as demonstrated in lecture   + create a new file called **cave\_unit\_test\_move\_2.py** (this is the file you should submit)   + follow the structure of the given unit test file to create your tests   Your tests should aim for *coverage* of valid states and particular emphasis on *edge cases.* |

### Test\_Cave\_1

[

['S'],

['E']

]

#### *Starting Set-Up*

* create cave\_1
* set layout to Test\_Cave\_1
* create adventure with cave\_1

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Description** | **Actions** | **Expected, Testable Outcomes** | **Expected Map** |
| starting set up | * starting set-up | * adventure.spot is [0,0] * adventure.visited is [[1],[1]] | S<  E |

#### Note that the *caret* < indicates the current spot of the adventurer.

#### *can\_move tests 1*

|  |  |  |
| --- | --- | --- |
| **Test Case Description** | **Actions** | **Expected, Testable Outcomes** |
| can\_move, down | * starting set-up * can\_move("down") | * *return* True |
| can\_move, up | * starting set-up * can\_move("up") | * *return* False |
| can\_move, left | * starting set-up * can\_move("left") | * *return* False |
| can\_move, right | * starting set-up * can\_move("right") | * *return* False |

#### *move tests 1*

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Description** | **Actions** | **Expected, Testable Outcomes** | **Expected Map** |
| move, down | starting set-up | (same as before) | S<  E |
|  | move(down) | * adventure.current\_spot is the list [1,0] * adventure.visited is [[1],[1]] * layout has not changed | S  E< |

#### We do not need additional unit tests for move on test\_cave 1 because the *move* function *assumes a valid move* and there are no other valid moves for this cave layout.

### Test\_Cave\_2

#### *Layout*

[

[‘R’, ‘R’, ‘R’, ‘R’, ‘R’],

[‘R’, ‘R’, ‘\_’, ‘R’, ‘R’],

[‘R’, ‘S’, ‘\_’, ‘E’, ‘R’],

[‘R’, ‘R’, ‘\_’, ‘R’, ‘R’],

[‘R’, ‘R’, ‘R’, ‘R’, ‘R’]

]

#### *Starting Set-Up*

* create cave\_2
* set layout to Test\_Cave\_2
* create adventure with cave\_2

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Description** | **Actions** | **Expected, Testable Outcomes** | **Expected Map** |
| starting set up | * starting set-up | * adventure.current\_spot is the list [2, 1] * adventure.visited is   *[[0,0,0,0,0],[1, 1, 1, 0, 0,],[1, 1, 1, 0, 0,],[1,1,1,0,0],[0,0,0,0,0]]* | ? ? ? ? ? ?  R R ? ? ?  R S ? ? ?  R R ? ? ?  ? ? ? ? ? ? |

#### *can\_move tests 2*

|  |  |  |
| --- | --- | --- |
| **Test Case Description** | **actions** | **Expected, Testable Outcomes** |
| can\_move, down | * Starting set-up * canMove(“down”) | * Return *false* |
| can\_move, right | * Starting set-up * canMove(“right”) | * Return *true* |
| can\_move, up | * Starting set-up * canMove(“up”) | * Return *false* |
| can\_move, left | * Starting set-up * canMove(“left”) | - Return *false* |

#### *move tests 2*

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case Description** | **actions** | **Expected, Testable Outcomes** | **Expected Map** |
| Move right | move(“right”) | * adventure.current\_spot is the list [2, 2] * adventure.visited is   *[[0,0,0,0,0],[1, 1, 1, 1, 0,],[1, 1, 1, 1, 0,],[1,1,1,1,0],[0,0,0,0,0]]* | ? ? ? ? ?  R R \_ R ?  R S \_<E ?  R R \_ R ?  ? ? ? ? ? |
| Move up | move(“right”)  move(“up”) | * adventure.current\_spot is the list [1, 2] * adventure.visited is   *[[0,1,1,1,0],[1, 1, 1, 1, 0,],[1, 1, 1, 1, 0,],[1,1,1,1,0],[0,0,0,0,0]]* | ? R R R ?  R R \_<R ?  R S \_ E ?  R R \_ R ?  ? ? ? ? ? |
| Move Down | move(“right”)  move(“up”)  move(“down”)  move(“down”) | * adventure.current\_spot is the list [3, 2] * adventure.visited is   *[[0,1,1,1,0],[1, 1, 1, 1, 0,],[1, 1, 1, 1, 0,],[1,1,1,1,0],[0,1,1,1,0]]* | ? R R R ?  R R \_ R ?  R S \_ E ?  R R \_ R ?  ? R R R ? |
| Move Left | move(“right”)  move(“up”)  move(“down”)  move(“down”)  move(“up”)  move(“left”) | * adventure.current\_spot is the list [2, 1] * adventure.visited is   *[[0,1,1,1,0],[1, 1, 1, 1, 0,],[1, 1, 1, 1, 0,],[1,1,1,1,0],[0,1,1,1,0]]* | ? R R R ?  R R \_ R ?  R S \_ E ?  R R \_ R ?  ? R R R ? |
| Move to finish | move(“right”)  move(“up”)  move(“down”)  move(“down”)  move(“up”)  move(“left”)  move(“right”)  move(“right”) | * adventure.current\_spot is the list [2, 3] * adventure.visited is * *[[0,1,1,1,0],[1, 1, 1, 1, 1],[1, 1, 1, 1, 1],[1,1,1,1,1],[0,1,1,1,0]]* | ? R R R ?  R R \_ R R  R S \_ E R  R R \_ R R  ? R R R ? |

#### 