“Found” Game

Starter Code

Make a directory called “Activity”, this directory will be maps that you can copy and test out with your program and with the sample code as well. All of the files below should be made under the “Activity” directory. Ex. ...../Activity/first_map.map

make a file called “first_map.map” and paste this code in it:

first_map.map - Pastebin.com

Https://pastebin.com/taxTBgDj

make a file called “second_map.map” and paste this code in it:

https://pastebin.com/MVBbZht7

second_map.map - Pastebin.com

make a file called “third_map.map” and paste this code in it:

H ttps://pastebin.com/jJQRKH2G

third_map.map - Pastebin.com

make a file called “map_maker.py” and paste this code in it:

H ttps://pastebin.com/T38E0Ky0

map_maker.py - Pastebin.com

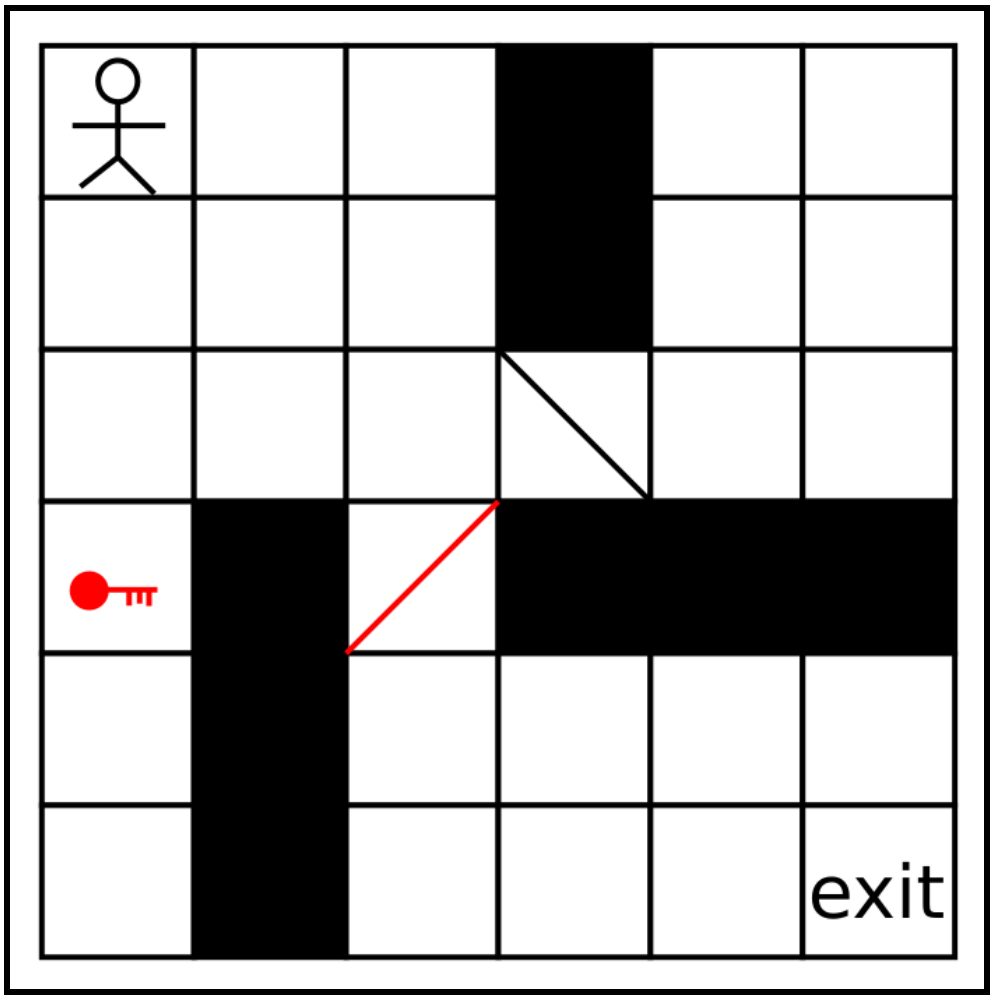
Your cod should be created under a file called “found.py

H ttps://pastebin.com/J8T2Rmii

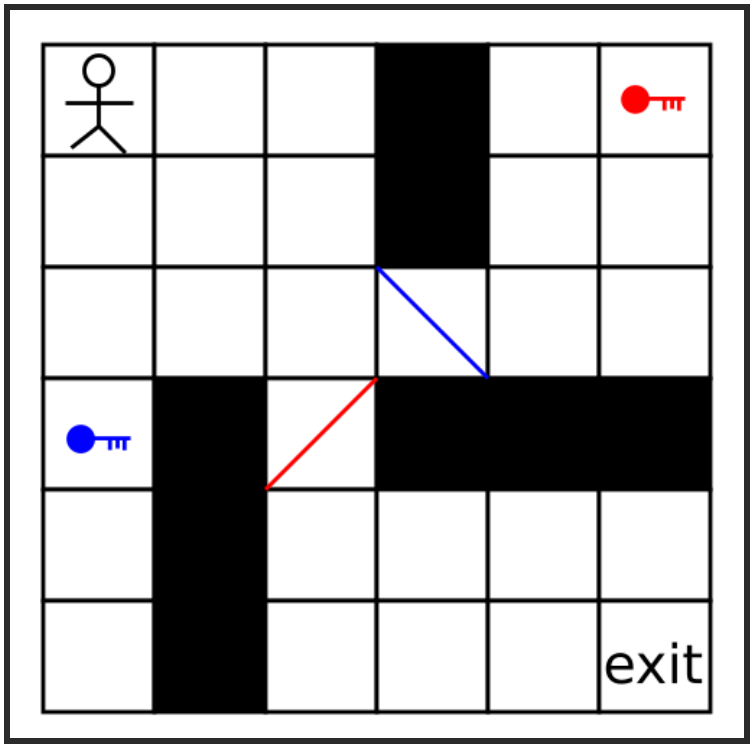
found.py - Pastebin.com

Description

The game is essentially a maze in which you must navigate your way to the 'x' or exit. You may need to collect stuff in order to open doors in order to get there. There are additional hidden doors that can be found if the player looks hard enough..

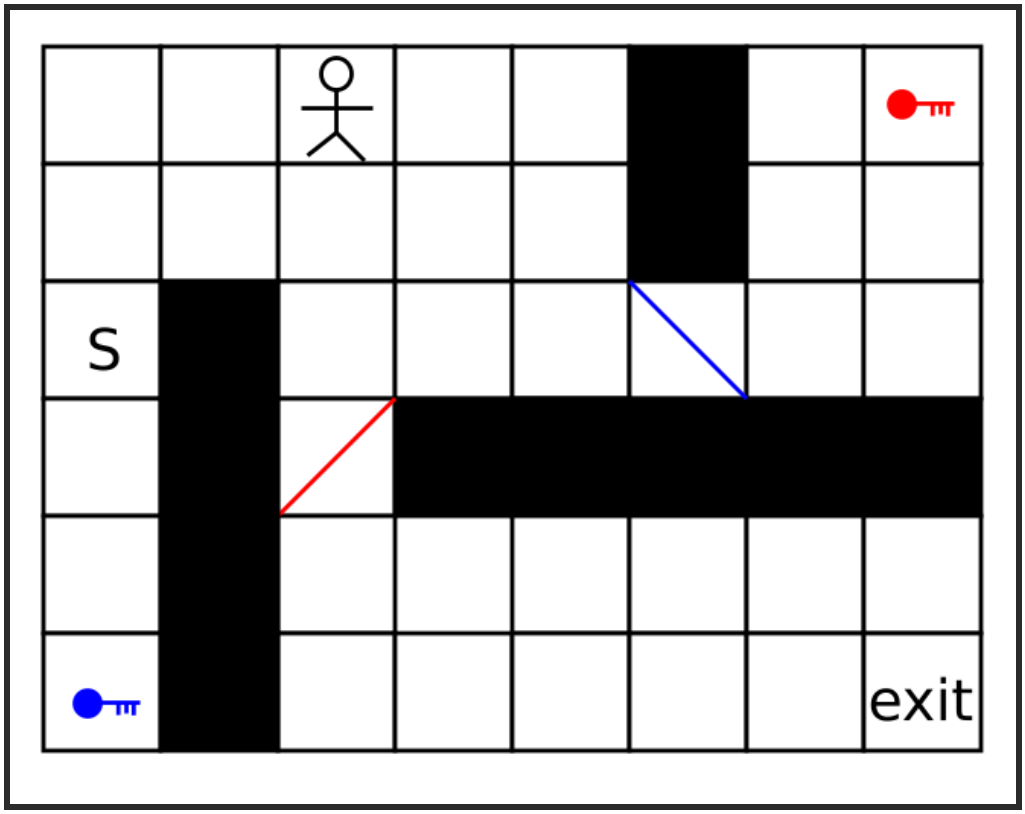


In this game, for example, there is a door (marked with a slash) that requires a red key to open. You won't be able to open the door until you obtain the key. You also have another door that you may open at any time.



On this map, you must first obtain the blue key in order to unlock the blue door, then obtain the red key in order to enter the next chamber with the exit.

Another map with S as a secret door may be found here. When you "search/use" the wall near the secret, it will show itself as a door, which will then open without the necessity of a key.



To win, you must first unlock the hidden door, then obtain the blue key, use the blue key to obtain the red key, and then escape through the red key door to the map's exit.

Implementation Details

The game follows the following cycle:

1) Get the name of the map file to open.
    a) Open that file using the provided code.
2) Print out the current game state (the map with the player).
    a) The board should display doors as 'd' or 'D', walls as '*', exits as 'x', the player should be '\u1330' secrets as walls (until they are revealed then they become doors).
3) Get the player's action.
4) Try to move the player.
    a) If the player is trying to move into a wall or off the board, prevent them from doing that.
    b) If they are trying to move through a door that isn't open or a secret that isn't open, prevent that as well.
5) If the player moves, if there's items on that space, add them to the player's inventory and empty out the square's items.
6) If the user has tried to search for secrets or open doors (same command), then any secrets within 1 square should be revealed and any doors should be "opened." By this I mean any square next to the square where you press 'e', including diagonally.
    a) Doors being opened get replaced with floors which are passable.
    b) Some doors require certain items to pass. Check to make sure the player's inventory contains all of the items required before the door is opened.
    c) Tell the user any items that they still need to open a door.
    d) Tell them that they have revealed a secret if they find one.
7) Go back to step 2.

Documentation

```
load_map(map_file_name)
```

This function will return a 2D grid which represents the map. However it is not quite so simple, so please read this.

Each cell in the grid is composed of a dictionary. The dictionary will have two to four keys:

```
{'symbol': '_', 'items': ['Blue Key'], 'requires': [], 'start': False}
```

The symbol here is underscore meaning that it is a floor cell. The items and 'requires' are both lists. They contain strings, which are considered items, like the Blue Key. When you step onto this square you get to pick up the Blue Key.

Requirements only matter for the purpose of doors (and secret doors).

```
{'symbol': 'd', 'items': [], 'requires': ['Blue Key']}
```

For this dictionary, the symbol is "d" for door, which currently is closed because the symbol is still 'd' and hasn't been set to _ yet. If the requires list is empty, then there are no requirements to open the door.

If the requires list is not empty, then we need to check the player's inventory to make sure that all the items in the requires list is in the player's inventory, and then we can open the door. Otherwise the door stays locked.

For instance you could have a door where it requires a blue and green key:

```
{'symbol': 'd', 'items': [], 'requires': ['Blue Key', 'Green Key']}
```

For maps constructed for testing the project, doors and secret doors will not contain items, and only doors and secret doors will have a requires list in their dictionary.

For instance this is a secret door which requires two keys:

```
{'symbol': 's', 'items': [], 'requires': ['Blue Key', 'Green Key']}
```

So let's look at the overall 2d array if for instance:

the_map = [[d1, d2], [d3, d4]]

where:

```
d1 = {'symbol': '_', 'start': True, 'items': ['Red Token'], 'requires': [,
'Green Key']}
```

```
d2 = {'symbol': '*', 'items': []}
```

```
d3 = {'symbol': 'd', 'items': [], 'requires': ['Red Token']}
```

```
d4 = {'symbol': 'x', 'items': []}
```

Looking at this map, we see that we have a starting position of:

the_map[0][0] because the_map[0][0]['start'] == True.

If there's no starting place, set it to the_map[0][0].

The Map Maker Program

Here's how it works:

It asks you:

"How many rows and cols will your map be?"

Enter two integers on that line.

Then it will ask you what file you want to create, just type in a name like "best_map_ever"

It will then create a best_map_ever.premap with the right dimensions. If you want to skip this step because you already have a premap created, just type a letter and it'll go right to the next step.

Go into this map and add things like 'd' for door, 's' for secret door, 'x' for exit, and '*' for walls.

After that there are five commands you can do once you select an index (two integers).

You can add-item [an item] or add-requirement [an item] or remove-requirement [an-item] or remove-item [an-item]. Only add requirements to doors since those are the only objects that use them. Lastly you can use 'add-start' to set the start attribute to true. Only do this once.

Once you're done type quit and you should write out a map file with the map extension. Technically there's also a premap file made with the grid drawn out but it doesn't have the items in it.

The map maker may have bugs in it, but it's what I've used to generate test maps. It's being provided without guarantee/warranty/and with all proper disclaimers.

Sample Run - Map Maker

Here is a sample run of the map maker program:

```
linux3[24]% python3 map_maker.py

How many rows and cols will your map be? 2 2
What file do you want to create and then modify to add "s" for secrets "d" for
doors, "x" for exit, "*" for walls "_" or space for floors: simple_map
Go and modify the file with the walls, exits, doors and secrets, press enter key
to continue…[go and modify the premap file, then press enter]
 0   1
0 _   d
1 s   x
Select position or quit: 0 0
What do you want to do (add item)? Add requirement?
add-item Golden Key
Select position or quit: 0 1
What do you want to do (add item)? Add requirement? add-requirement Golden Key
Select position or quit: quit
```

```
[[{'symbol': '_', 'items': ['Golden Key']}, {'symbol': 'd', 'items': [],
'requires': ['Golden Key']}], [{'symbol': 's', 'items': []}, {'symbol': 'x',
'items': []}]]
```

Sample Run - Simple Map

Here I'll show a sample run of the simple map since it's short and small enough.

```
linux3[25]% python3 found.py
What map do you want to load? simple_map.map
0 Åd
1 *x
Your inventory is:
Enter Move (wasd) (e to activate doors or secrets): e
You found a secret!
0 Å_
1 Dx
Your inventory is: Golden Key
Enter Move (wasd) (e to activate doors or secrets): d
0 _Å
1 Dx
Your inventory is: Golden Key
Enter Move (wasd) (e to activate doors or secrets): a
0 Å_
1 Dx
Your inventory is: Golden Key
Enter Move (wasd) (e to activate doors or secrets): s
0 Å_
1 Dx
Your inventory is: Golden Key
Enter Move (wasd) (e to activate doors or secrets): d
0 _Å
1 Dx
Your inventory is: Golden Key
Enter Move (wasd) (e to activate doors or secrets): s
You win!
```

Allowed Built-ins/Methods/etc

- **list unpacking**

  ex. my_position = [y, x]

  pos_y, pos_x = my_position

- Global constants **are permitted** for this activity, and should be written in all capital letters, placed where the other constants are.
- Declaring and assigning variables, ints, floats, bools, strings, lists, dicts.
- Using +, -, *, /, //, %, **; +=, -=, *=, /=, //=, %=, **= where appropriate
- Comparisons ==, <=, >=, >, <, !=, in
- Logical and, or, not
- if/elif/else, nested if statements
- Casting int(x), str(x), float(x), (technically bool(x))
- For loops, both *for i* and *for each* type.
- While loops
    - sentinel values, boolean flags to terminate while loops
- Lists, list(), indexing, i.e. my_list[i] or my_list[3]
    - 2d-lists if you want them/need them my_2d[i][j]
    - Append, remove
    - **list slicing**
    - **list unpacking**
- If you have read this section, then you know the secret word is: createous.
- String operations, concatenation +, +=, split(), strip(), join(), upper(), lower(), isupper(), islower()
    - **string slicing**
- Print, with string formatting, with end= or sep=:
    - '{}'.format(var), '%d' % some_int, f-strings
    - Really the point is that we don't care how you format strings in Python
    - Ord, chr, but you won't need them this time.
- Input, again with string formatting in the prompt, casting the returned value.

- **Dictionaries**
    - creation using dict(), or {}, copying using dict(other_dict)
    - .get(value, not_found_value) method
    - accessing, inserting elements, removing elements.
- Using the functions provided to you in the starter code.
- Using import with libraries and specific functions **as allowed**

- Tuples are allowed, as they are immutable lists**.**

Forbidden Built-ins/Methods/etc

This is not a complete listing, but it includes:

- break, continue
- **Global variables**
- **recursion**
- methods outside those permitted within allowed types
    - for instance str.endswith
    - list.index, list.count, etc.
- Keywords you definitely don't need: await, as, assert, async, class, except, finally, global, lambda, nonlocal, raise, try, yield
- The *is* keyword is forbidden, not because it's necessarily bad, but because it doesn't behave as you might expect (it's not the same as ==).
- built in functions: any, all, breakpoint, callable, classmethod, compile, exec, delattr, divmod, enumerate, filter, map, max, min, isinstance, issubclass, iter, locals, oct, next, memoryview, property, repr, reversed, round, set, setattr, sorted, staticmethod, sum, super, type, vars, zip
- If you have read this section, then you know the secret word is: argumentative.
- exit() or quit()
- If something is not on the allowed list, not on this list, then it is probably forbidden.
- The forbidden list can always be overridden by a particular problem, so if a problem allows something on this list, then it is allowed for that problem.