Exercise4EscapeRoomAsychUserInput

Jump to bottom

sethnielson edited this page 7 days ago · 2 revisions

Exercise 4: Escape Room with Asynchronous Operations

Assigned	9/11/2019
Due	9/16/2019
Points	25

Overview

In this exercise, you're going to learn how to do "two things at once" in asyncio. It involves co-routines, so prepare to have your mind blown.

Part 1: Review Escape Room 004

Pull from GitHub the updated sample code and look at `escape_room_004.py`.

There's a lot that's changed!!! Let's walk through some of it.

First, you'll notice there is a new command represented by the function `_cmd_hit`. After you look through the code, try to answer the following questions.

1. What can you hit?
2. What can you hit with?
3. Under what conditions can you hit the target?
4. Under what conditions can you hit with something?

What's been added to the game is a flying key that you can hit with a hammer to turn into a regular key. But you can only turn it into the key with the hammer and only when the flying key is on the wall. Was that logic in the `_cmd_hit` function? If not, where else might it be?

Continuing down through the code, you might notice that `create_room_description` has changed. Can you tell how? Why do you think this change as added?

New description functions have been added for the flying key. One is for it's regular description and one is for the newly created "short" description. Can you figure out where short descriptions are used? There's also a function called `short_description()`. What does it do?

Skip `flyingkey_hit_trigger` for now. We'll come back to it!

Let's move into the game setup. You'll notice we've added a key object. It's listed as visible and gettable right from the start. How does that work? If it isn't available until the flying key is smashed with the hammer, how is it visible and gettable? Or, rewording the question in the positive direction, what keeps the user from getting the key even though it is visible and gettable?

You'll also notice that we added a `flyingkey` object (think the flying keys in Harry Potter and the Sorcerer's Stone). Take a look at its attributes. Are they fairly self explanatory? If not, what kind of comment might you write to provide better direction?

Let's skip ahead a bit to the "triggers." Triggers in this game are for causing the game to do something automatically when something else happens. For example, in your old escape room code, there was already a trigger to make the hairpin become visible when you look in the mirror. There's a general purpose "post command" trigger that causes the room to advance time.

Now take a look at the trigger for the flying key. When is this called? How does the trigger work together with the command code? Now go back and look at the `flyingkey_hit_trigger`. Does that code make sense now?

There are a few other changes in `create_game()`. See if you can figure them out yourself. And take note of the `self.agents` data. We're going to look at the agent code and all the asyncio code in the next section.

For now, the new rules to escape the room:

1. `look mirror`

2. `get hairpin`
3. `unlock chest with hairpin`
4. `open chest`
5. `get hammer in chest`
6. `hit flyingkey with hammer` (only when the flyingkey is on the wall!)
7. `get key`
8. `unlock door with key`
9. `open door`

Note I called the flying key: `flyingkey`. We really don't support spaces right now.

We just walked through the code that does 99% of this. All you need to do is create an aynchronous function that is the "brain" of the flying key and plug it into asyncio. See part 2 below!

Part 2: The Agent Brain

You may have noticed that there are two functions below `create_game()` called `move_flyingkey` and `flyingkey_agent()`. The move function is already implemented. It moves the flying key from one location to the other, outputs a message about it, and updates triggers for "post command". This last part makes the clock move when the flying key moves! Uh oh, you'd better move quickly!

But the agent command is not yet implemented. You need to do that:

1. The agent brain should repeat until the game is over or the flying key is no longer flying
2. The agent brain is in an asynchronous function and should NOT block. Use `await asyncio.sleep(n)` to sleep for n seconds in a non-blocking fashion
3. Every 5 seconds or so, the flying key should move. You can use the `move_flyingkey` function for this.

That's it. That's the whole thing. You should be able to execute the `escape_room_004.py` function now. Play the game. See how it works. Have some fun.

When you're done, we need to talk about writing this into network code. Go back to `escape_room_004.py` and look at `main()`. It is also an asynchronous function. What does it do?

Well, first, I should tell you what `add_reader` does. Remember that in earlier code, we used `input()` to get input from the keyboard. But we can't do that now. Do you know why?

If you said, "Because, professor Nielson, that's a blocking call," you get a gold star! Wear it proudly!

Yes, `input()` is a blocking call. You CANT do that! Instead, we use `add_reader` to `stdin` (input) and have a callback whenever there's input on the channel. Note that the callback doesn't give us the input, it just alerts us to input. That's why the first command in `game_next_input()` is to get the input from `stdin`.

Finally, what's with `asyncio.wait()`? This function waits in a non-blocking fashion until all the futures it's passed as input have completed. The reason for the `game.agents` stuff is that someday we might want to have more agents than just the `flyingkey`. Maybe we'll want a moving suit of knight's armor, walls that close in on us, etc. So all agents reported by the game are started here and then awaited.

The next part of this assignment is to figure out how to asynchronously start the game and the game agents inside your own networking server. You can't just call main because you need to launch this at a certain point in time. And you can't use the `add_reader` because you receive your input from `data_received()`. So think a bit about how to incorporate.

Part 3: Grading and Passing Off

As with the last exercise, you will use the auto-grader. But, because the output of the game is a bit unpredictable, this time you don't have to worry about getting the text exactly right. You pass if

1. As a client, you escape the room
2. You provide a server that the autograder can escape.

The protocol is exactly like last time. So refer back to exercise 3 if you need to know how to connect, get input, pass back answers, etc.

Oh and, of course, the port this time is 19004.

Good luck! (Note: Hastily written. Let me know if there are errors or things left out)

▼ Pages 16

Find a Page...

Home

BackgroundOverlayNetworks

Exercise1GettingStarted

Exercise2EscapeRoomSockets

Exercise3EscapeRoomAsyncio

Exercise4EscapeRoomAsychUserInput

Exercise5EscapeRoomPlayground

Clone this wiki locally

https://github.com/CrimsonVista/20194NetworkSecurity.wiki.git