# Flik Debugger Documentation

## Introduction

### Overview

The objective of this research is to build a tool that allows the visualization of the behavior of a reinforcement learning agent and facilitates its debugging. Additionally, it is intended to understand the behavior of the RL agent during the training process, to help developers identify the main challenges in the correct operation of RL programs. The research has been carried out since January 2024 and will end in December 2024.

In this sense, Flik is a tool that will help your experience debugging a Reinforcement Learning program better. It is a python-based debugger and is done to be used only using the console. It will allow you to run your program step by step, check its variables and modify them during execution, and travel through time over the program.

### Requirements

To be able to do this test you'll need to have docker installed or have a [play with docker](https://labs.play-with-docker.com/) account. To download a docker image, in which you will find all the useful tools installed in order to use Flik for three case scenarios for this test. Now, please follow the instructions for loading it in your computer in the following section.

## Installation

### Option 1: Play with docker

Use the following command to download the docker image. And start running it on your computer or run it in [play with docker](https://labs.play-with-docker.com/) page. To be able to run it in [play with docker](https://labs.play-with-docker.com/) page, create a new account and add a new instance (this might take a while, but is for free and on the web, so you don’t need to install docker).

docker pull lrodriguez22/py-flik-debugger:latest

You can now run the following command:

docker run -it lrodriguez22/py-flik-debugger

### Option 2: download tar

Download the docker container from this [docker container](https://drive.google.com/file/d/1UpcZ6NZTvE9L12fV4NqpkAalY_1x0xsS/view?usp=sharing).

Once you have downloaded the following .zip. Open a terminal and make sure you go to the path you download the .zip, then type:

docker load -i py-flik-debugger.tar

This should take some minutes to complete. Once it has finished downloading, run the [docker container](https://drive.google.com/file/d/1UpcZ6NZTvE9L12fV4NqpkAalY_1x0xsS/view?usp=sharing) with the following command:

docker run -it py-flik-debugger

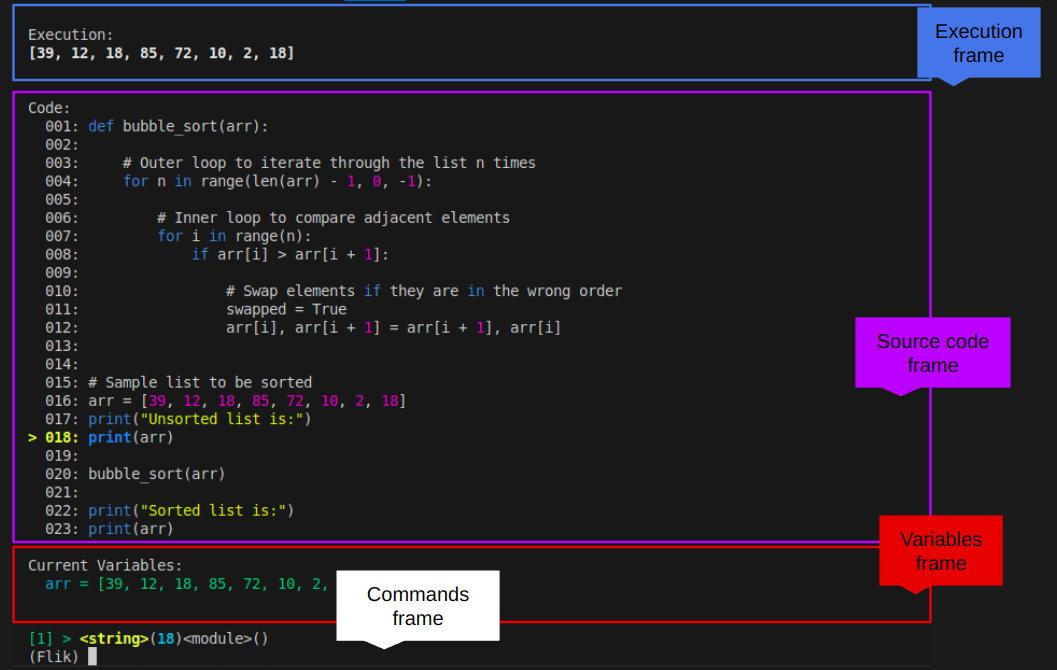
## Getting Started

### Launching Flik

Once you are inside the container to use Flik just go use the following command:

flik path\_to\_file/file.py

This will basically launch the debugger for the file you want to debug. Once you’ve done this, you should see an interface similar to the following image:



Here, you should be able to see the current variables, the source code of the program you are running, and a console in which you will be able to interact with the code.

### Commands

As almost all debuggers Flik lets you navigate through the code, the following list of commands will help you get started with Flik.

|  |  |
| --- | --- |
| **Command** | **Description** |
| p | Print the value of an expression. |
| pp | Pretty-print the value of an expression. |
| n | Continue execution until the next line in the current function is reached or it returns. |
| s | Execute the current line and stop at the first possible occasion (either in a function that is called or in the current function). |
| c | Continue execution and only stop when a breakpoint is encountered. |
| step\_back | goes one step in time from the line you are |
| back\_to | goes to the line you wish |
| variables | shows the local and global variables from the current execution |
| sticky | if you lose the pretty format of Flik you can go back by typing using this command. |
| args | To see the values of the function running right now. |
| setvar | It is a way in which you can modify a variable. Note that because of optimizations of python, and variables protection this will not work in some cases. |
| unt | Continue execution until the line with a number greater than the current one is reached. With a line number argument, continue execution until a line with a number greater or equal to that is reached. |
| b | With no arguments, list all breaks. With a line number argument, set a breakpoint at this line in the current file. |
| w | Print a stack trace, with the most recent frame at the bottom. An arrow indicates the current frame, which determines the context of most commands. |
| u | Move the current frame count (default one) levels up in the stack trace (to an older frame). |
| d | Move the current frame count (default one) levels down in the stack trace (to a newer frame). |
| help | See a list of available commands. |
| q | Quit the debugger and exit. |

## Tasks Description

### The idea for every task is for you to find as many bugs as you can using Flik in each of the RL programs. For each task it is expected that you find the main bug hidden in the code. You are supposed to complete the following [form](https://forms.office.com/r/fg2AnRPfD1) with that information. Good Luck!

In [this link](https://github.com/larodriguez22/Flik_Experiments) you will find the experiments to be run for these tasks. Feel free to take a look and download them.

### 

### Task 0: Example

In this task I will show you how to identify a bug using Flik. This is a basic example with an idea on what you are supposed to do. In this example we are using the division algorithm to divide two numbers. But we are missing something, we are not handling all the possible errors. If we run the following program for long division, it will run without errors.

def long\_division(dividend, divisor):

quotient = 0

remainder = abs(dividend)

divisor = abs(divisor)

while remainder >= divisor:

remainder -= divisor

quotient += 1

if (dividend < 0) != (divisor < 0):

quotient = -quotient

return quotient, remainder

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

dividend = 17

divisor = 5

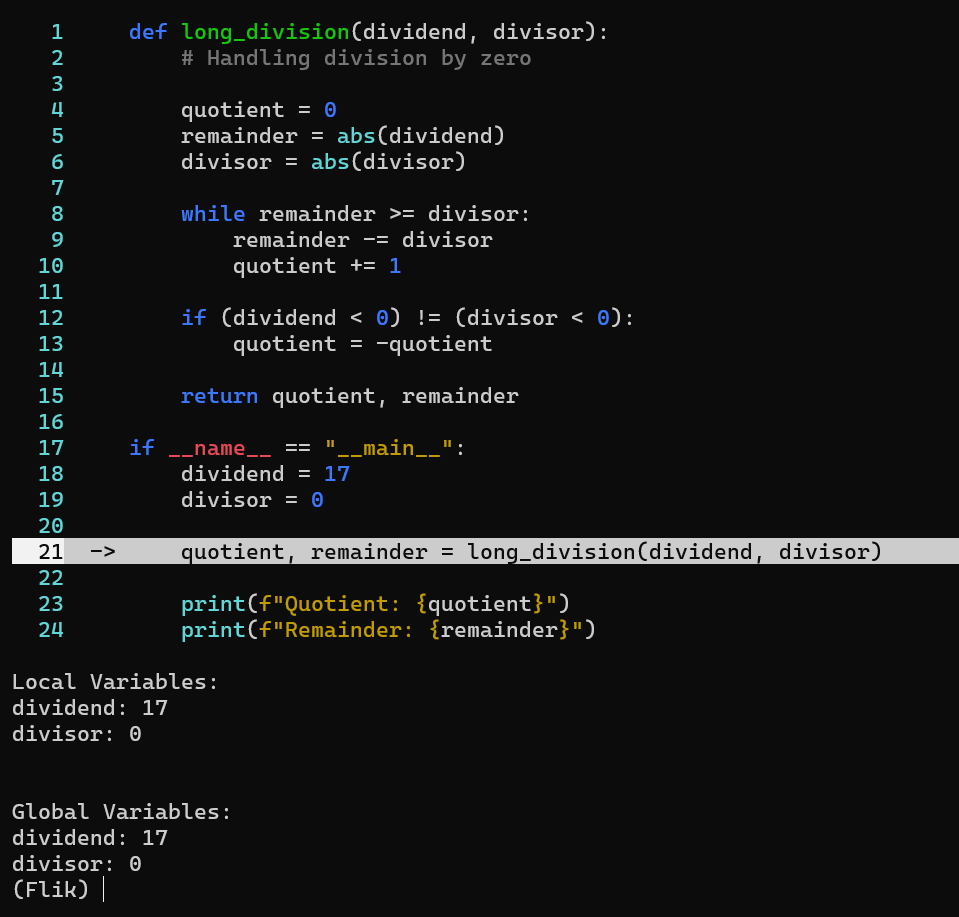
quotient, remainder = long\_division(dividend, divisor)

print(f"Quotient: {quotient}")

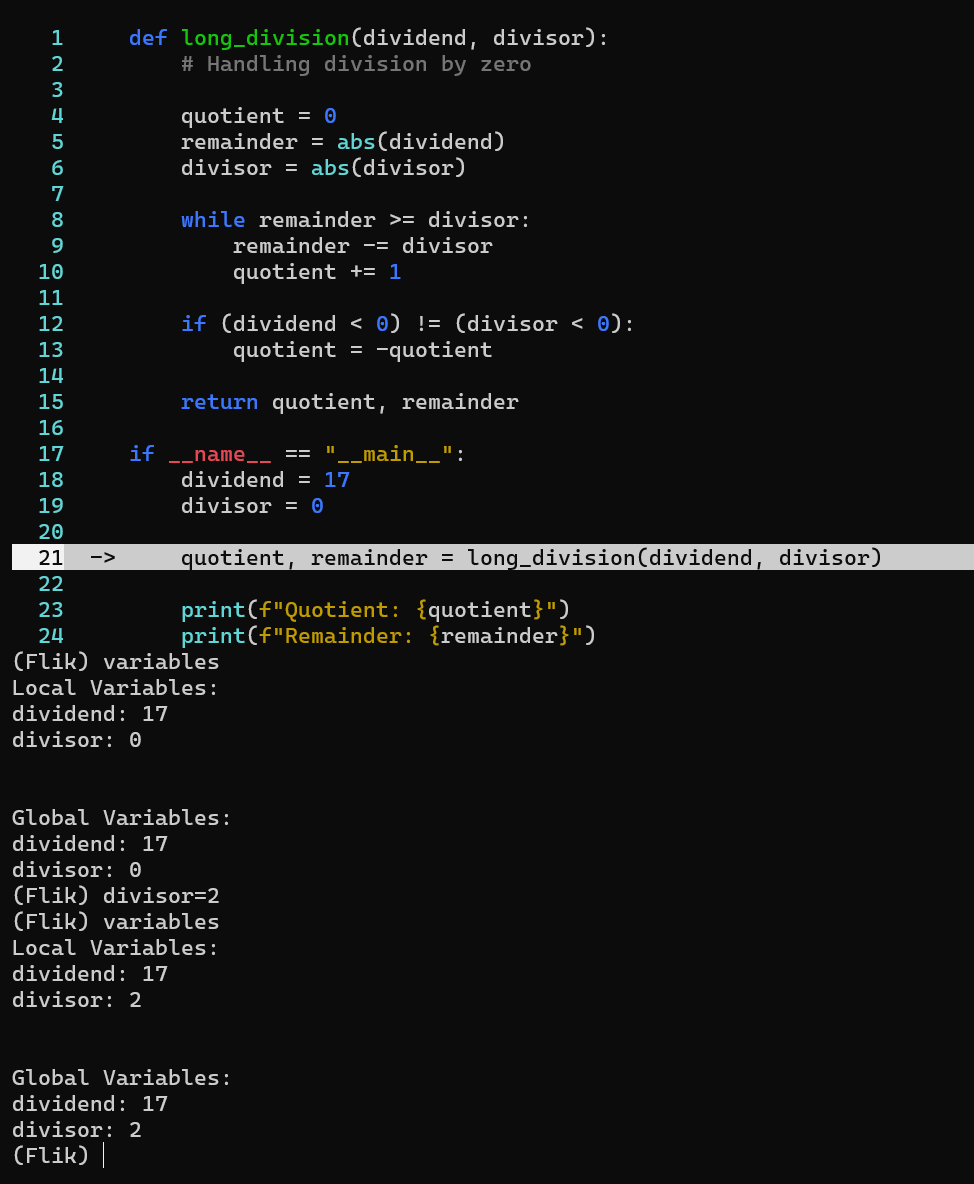
print(f"Remainder: {remainder}")

Now, let’s consider a case in which a developer wants to execute the following division:

long\_division(15, 0)

This will generate an error. We can use Flik to run different examples of this code, to change the variables and to see why an error is happening, for example, we can go to line 23, which is going to have an error.  


Then we can run s, to get into the method. And see in real time the execution of the variables. At some point in the execution, we realized that for this case it is not working properly with our method. We then use the commands back\_to 18. Then we change the variable divisor value to 2, and we go step by step again.



Once we have realized the error, we can change the code and add the following exception:

# Handling division by zero

if divisor == 0:

raise ZeroDivisionError("Cannot divide by zero")

Now, it is your turn to find the bugs in these other tasks.

### Task 1: GridWorld

The idea of this task is for you to figure out what is happening and what is the wrong behavior happening on this Reinforcement Learning program. The idea is for you to identify bugs on it. The gridworld environment consists of a nXn (10 X 10 in our example) rectangular board/grid, in which each tile (i,j) represents a specific state of the board. Tiles in the board may be walls, which agents cannot cross. Additionally, there are special exit tiles that give a positive or negative reward to agents, as shown in the following image. All tile types are unknown to the agent that moves from a given starting point in the board, searching for the goal state (example: exit states with positive reward of 1). The agent moves from state to state, avoiding obstacles and incorrect exit states (which give a reward of -1 when used to exit). For 25 episodes, the result should look like:  


Note that the agent is learning so the values are being propagated on the Q-table.

You can also run the gridworld experiment using the following command:

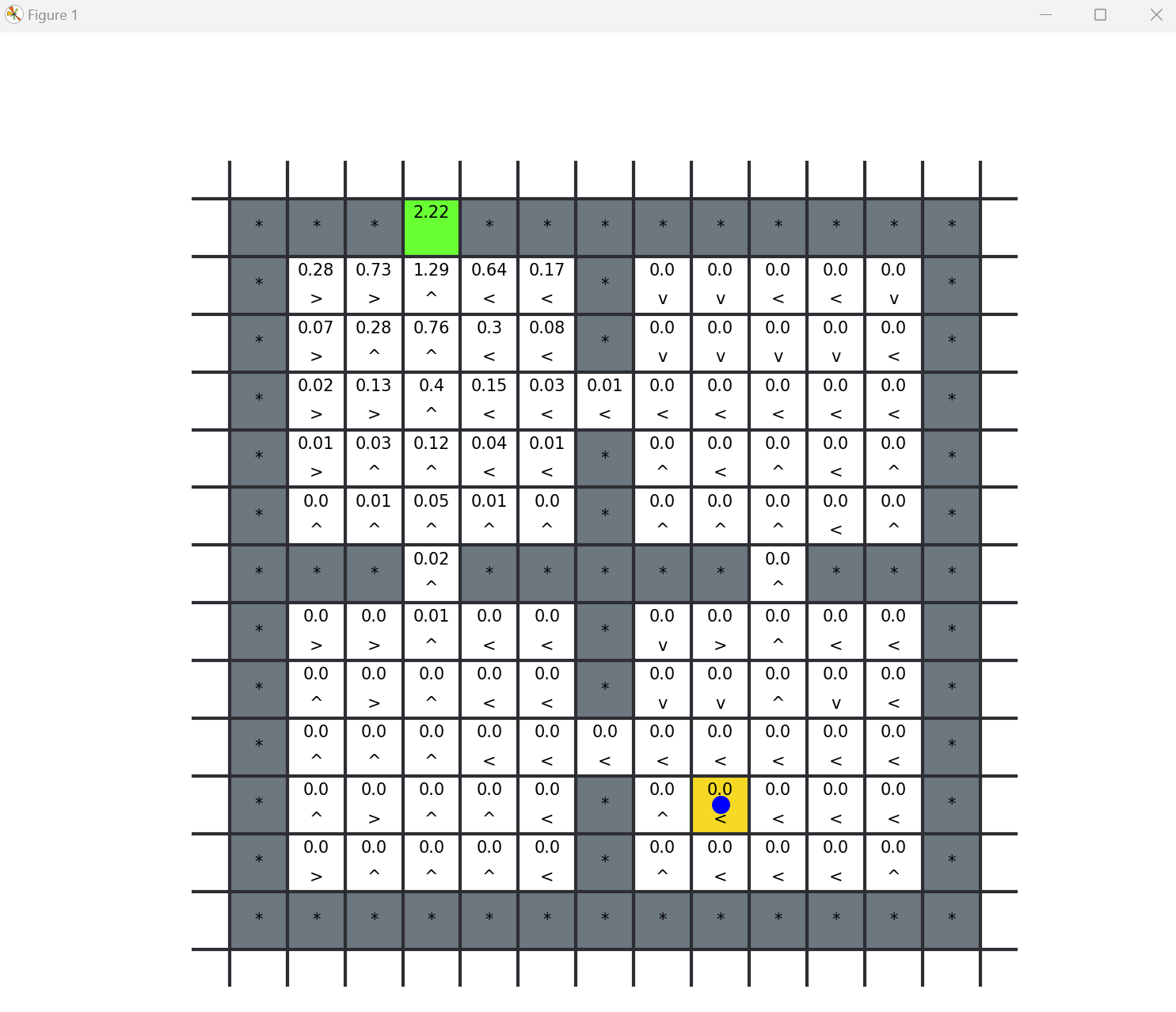
python3 .\gridworld\gridworld.py

To run this using flik run (inside the docker container):

flik experiments/gridworld/gridworld.py

### Task 2: Rooms

The four rooms maze environment consists of a 13 X 13 board/grid divided in 4 sections, with walls between them, and a door opening to go from one room to another, as shown in the figure above. The agent's objective in this environment is to exit through the upper-left room (the green square) in the fewest possible steps. Reaching the exit state gives a reward of 1, and no other action gives a reward to the agent. In each episode the agent starts from any valid position in the grid, for example, the yellow square in the bottom-right room in the figure. For 25 episodes, the results should look like:



Note the influence of the learning rate. You can also run the rooms experiment using the following command:

python3 .\rooms\rooms.py

To run this using flik run (inside the docker container):

flik experiments/rooms/rooms.py

### Task 2: Driving Assistant

In this example the agent learns to drive on a two-lane road, on the driving lane, at the speed limit, and overtaking slow traffic ahead. The idea of this task is that the agent goes as fast as possible on the road. In the road there are only two lanes, and there are other cars that the agent must pass without crashing on them. The possible actions for the agent are: straight, slow\_down, speed\_up, steer\_left, steer\_right. The following is the visual interface of this environment.

Note that we want the agent to learn how to drive at the maximum velocity allowed, we don’t want the agent to stop, or to crash.

You can also run the rooms experiment using the following command:

python3 .\cars\environment.py

To run this using flik run (inside the docker container):

flik experiments/cars\_example/environment.py