Mini-Project
Madeleine Rodriguez

Project: Tic-Tac-Toe

I have p4 code (tictactoe.p4) and a python script (tictactoe.py), which are saved in my CWM-ProgNets repository: https://github.com/morodrigu/CWM-ProgNets/tree/main/MiniProject

Description of project:
I have written p4 and python code which will enable a game of tic-tac-toe to be played between a user and the raspberry pi.
An input of the desired location to place a 'cross' will be requested from the user and a 3x3 grid will be printed which shows the state of the board after the user has made their first move. The raspberry pi will then respond with their move according to the contents of the table in the p4 script, and the updated board will be displayed with the raspberry pi's move added.
The user will be prompted to enter location values until a combination in the table has been reached which indicates to the raspberry pi that the board corresponds to the user winning, the raspberry pi winning or there being no more free squares for the user to place a 'cross'. At this point, the program will output a message indicating the situation and then the program will terminate.

Architecture:

Hardware:
The lab machine is the interface between the python code and the user and accepts user input.
The raspberry pi runs the p4 code.

Software:
Inputs from the user are accepted in to the python code, and the data is packaged in to the header and sent to the raspberry pi.

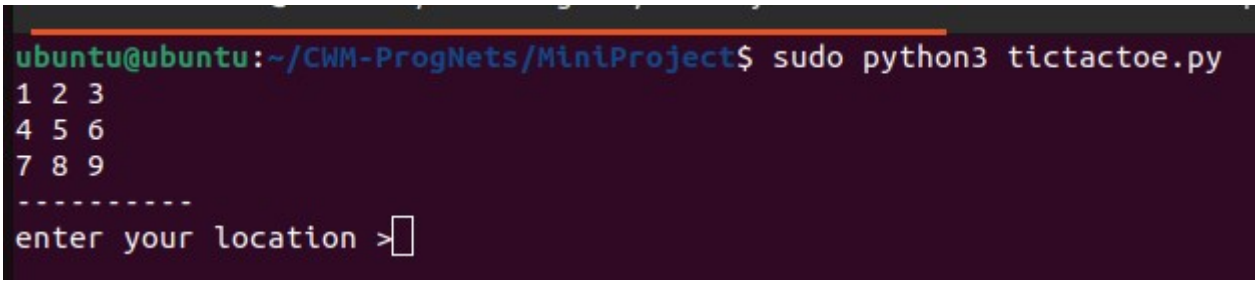
The packet header is set up as such:
P / 4 / ver / s1 / s2 /s3 / s4 / s5 / s6 / s7 / s8 / s9 / res
where s1-9 are the states of each of the squares on the board. The state can be 0 (for empty), 1(for selected by player 1) or 2 (for selected by player 2).

'res' in the header is updated by the raspberry pi when the raspberry pi's move has been selected using the table and is sent back to the lab machine. The lab machine then adds a '2' in the location indicated in 'res'.

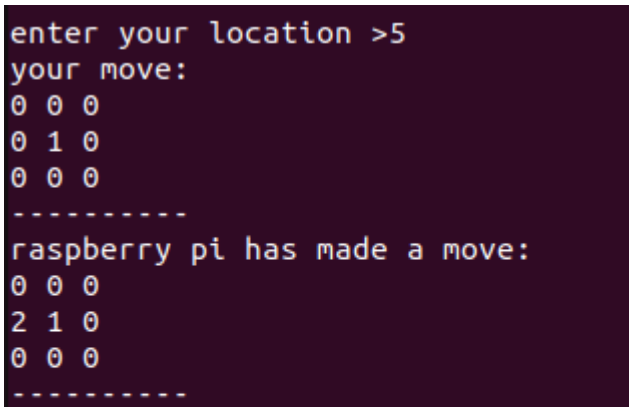
The new state (s1-9) is then displayed on the lab machine and the user is prompted to make their next move.

Operation:

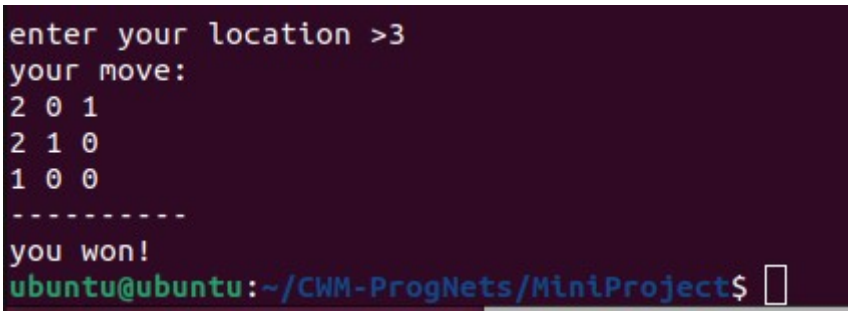
When the python script is run, the user is prompted to enter where they want to place their first ‘cross’ (which corresponds to and is written as a 2 throughout my code and the game):



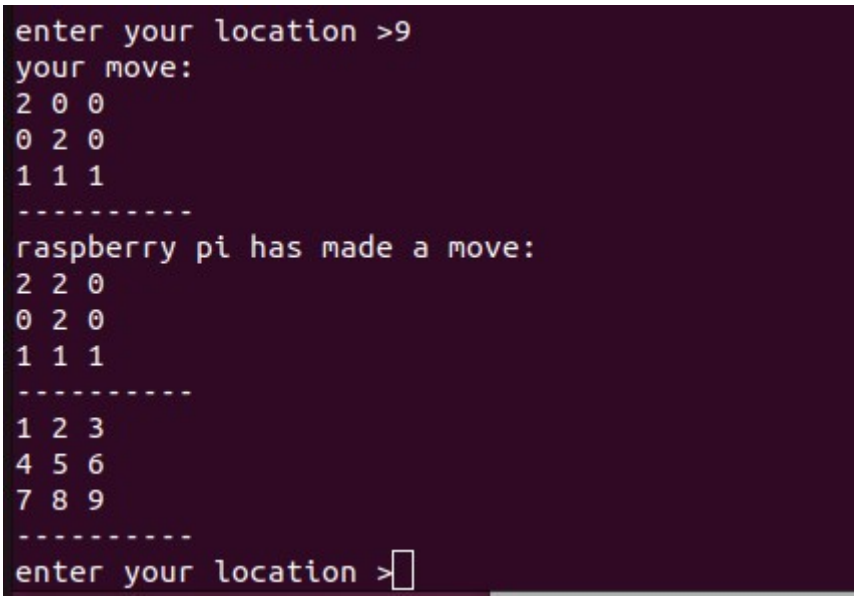
After the move has been entered, the updated board is displayed along with the raspberry pi’s response:



The user will then be prompted to enter another move until eventually a combination of states has been reached which causes the raspberry pi to declare a winner:



I was unable to add a number of combinations to the p4 table which has led to some winning scenarios or full board not being recognized, for example:



Although the user has one, no winner is declared as the combination of states does not appear in the p4 table.

Performance tests:
I performed multiple performance tests:

1. Testing if the raspberry pi responded to unprogrammed states/states which did not appear in the table:

I am testing (1,0,1,0,2,1,0,0,0), which does not appear in the p4 table:

```
your move:
0 0 1
0 2 1
0 0 0
----------
raspberry pi has made a move:
2 0 1
0 2 1
0 0 0
----------
```

The raspberry pi successfully makes a move according to the default action of selecting the first empty square. This is not a good move, but it shows that the default action works.

2. Testing if squares which were already filled were accepted as an input:

I attempted to select square 1 twice and received the error message ‘this is not an empty square’

```
ubuntu@ubuntu:~/CWM-ProgNets/MiniProject$ sudo python3 tictactoe.py
1 2 3
4 5 6
7 8 9
----------
enter your location >1
your move:
1 0 0
0 0 0
0 0 0
----------
raspberry pi has made a move:
1 0 0
0 2 0
0 0 0
----------
1 2 3
4 5 6
7 8 9
----------
enter your location >1
This is not an empty square
```

3. Testing if squares which were outside the range of 1-9 were accepted as an input:

I attempted to enter 11 as my input location and received the corresponding error message.

```
ubuntu@ubuntu:~/CWM-ProgNets/MiniProject$ sudo python3 tictactoe.py
1 2 3
4 5 6
7 8 9
----------
enter your location >11
Location needs to be a number from 1 to 9.
```

4. Testing if the raspberry pi could recognized a winning state which had been entered in to the p4 table:

(2,0,1,2,1,0,1,0,0) was a programmed winning state, and it triggers a player 1 (user) winning message:

```
enter your location >3
your move:
2 0 1
2 1 0
1 0 0
----------
you won!
ubuntu@ubuntu:~/CWM-ProgNets/MiniProject$
```

There were multiple failed performance tests as well such as:

- Recognising an unprogrammed winning state, which could not be recognised as it was not in the p4 table.
- Recognising an unprogrammed table full state. The user's next input was prompted even with a full 3x3 table.

With additional entries to the p4 table, I believe that these performance tests could be passed as there is nothing fundamentally wrong with my program which would cause them to fail.