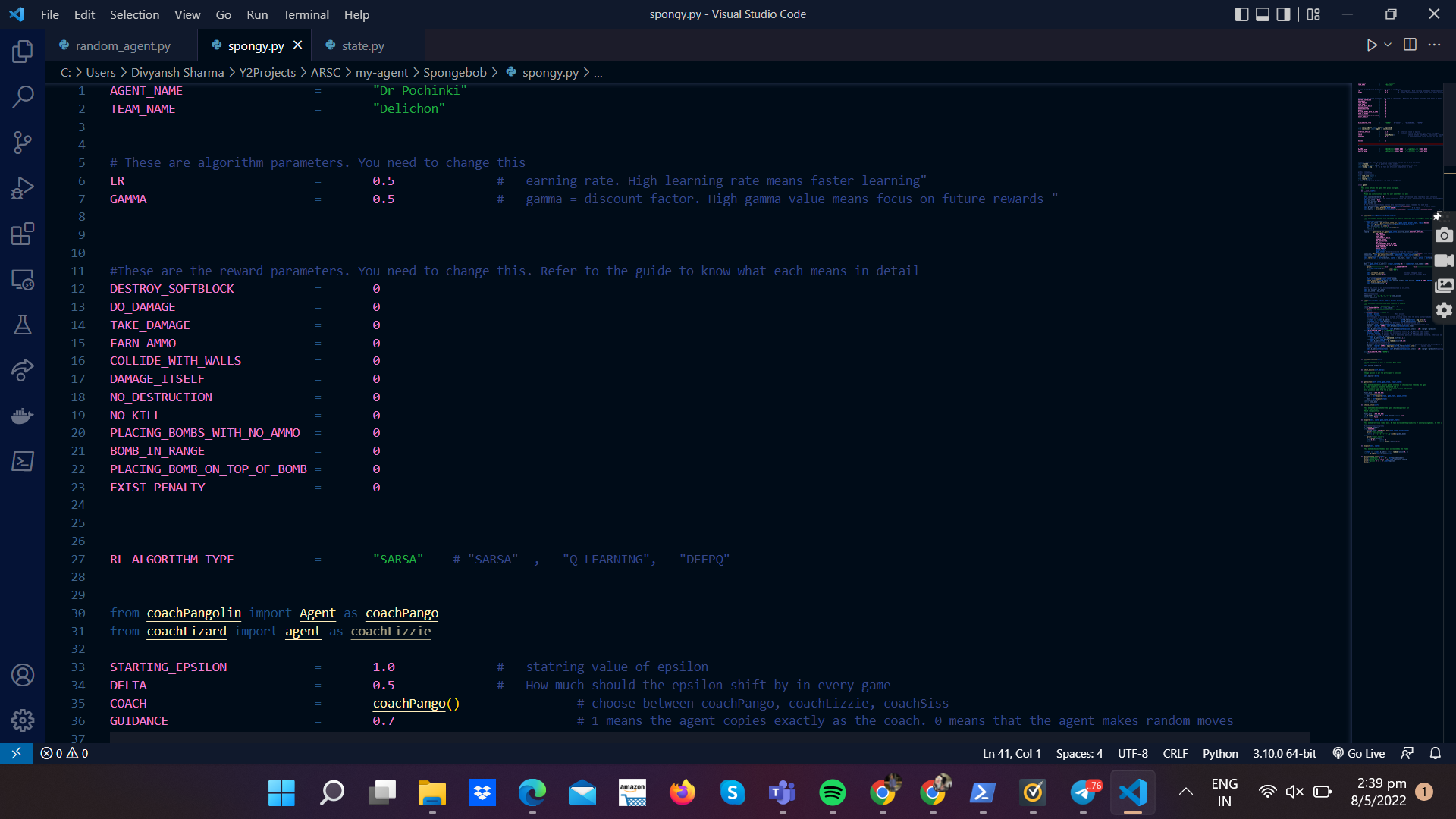
How to create a Q-Learning/ SARSA agent?

To do this, follow this guide.

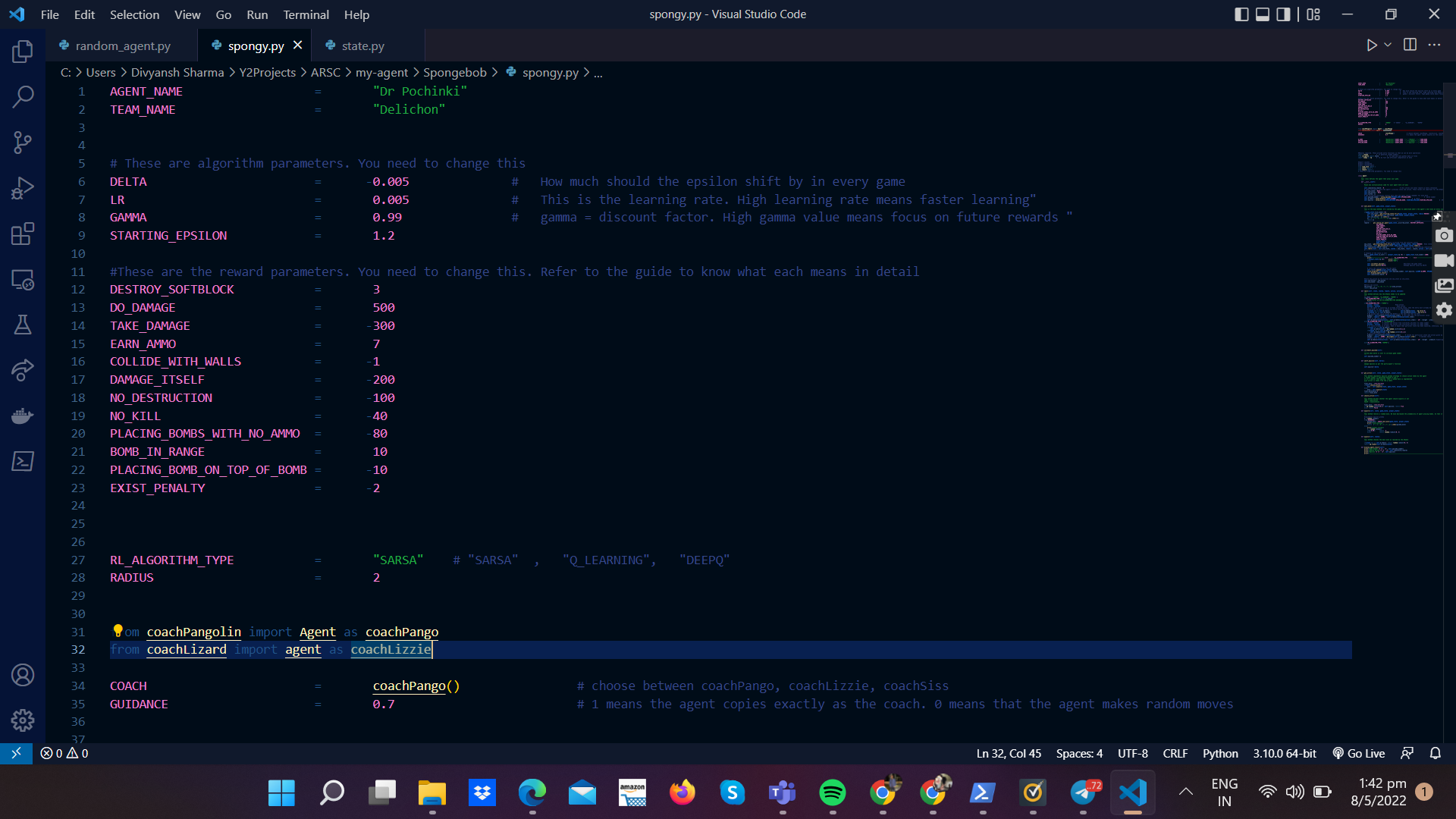
1. Download the provided-agents.zip from the google drive.
2. Extract the zip file and copy all the contents of the folder.
3. Create a folder inside the my-agent folder. Call this folder by your agent name.
4. Open the agent\_name.py
5. You will see the following file:



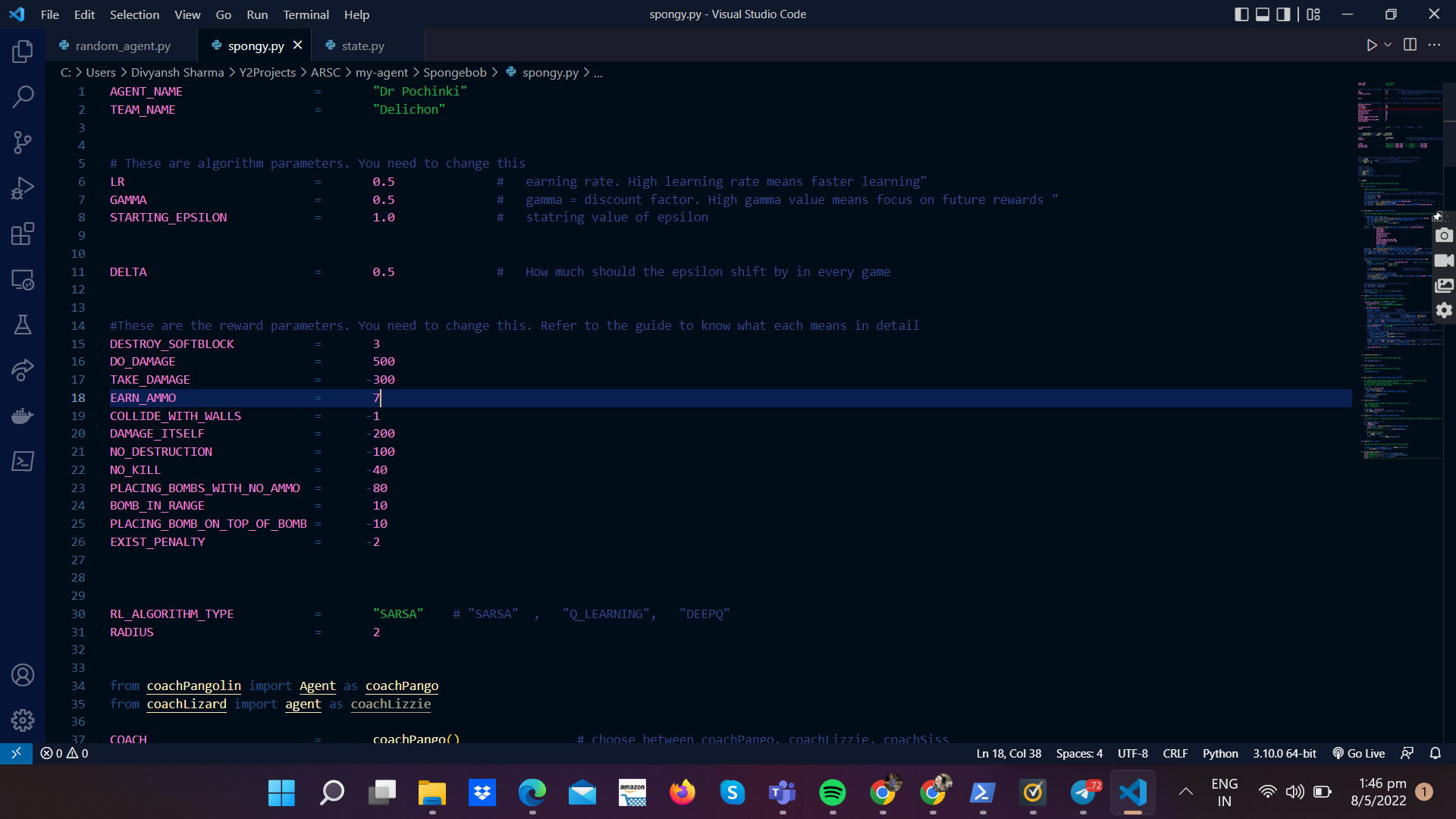
These values are currently set to 0.

It’s your task to assign the values. Let's go step by step through all the functions.

# Team Name and Agent Name

It’s important that each agent you create has a unique name and your correct team name. These variables are used to create names for files that store agent progress.

# GAMMA and LEARNING RATE

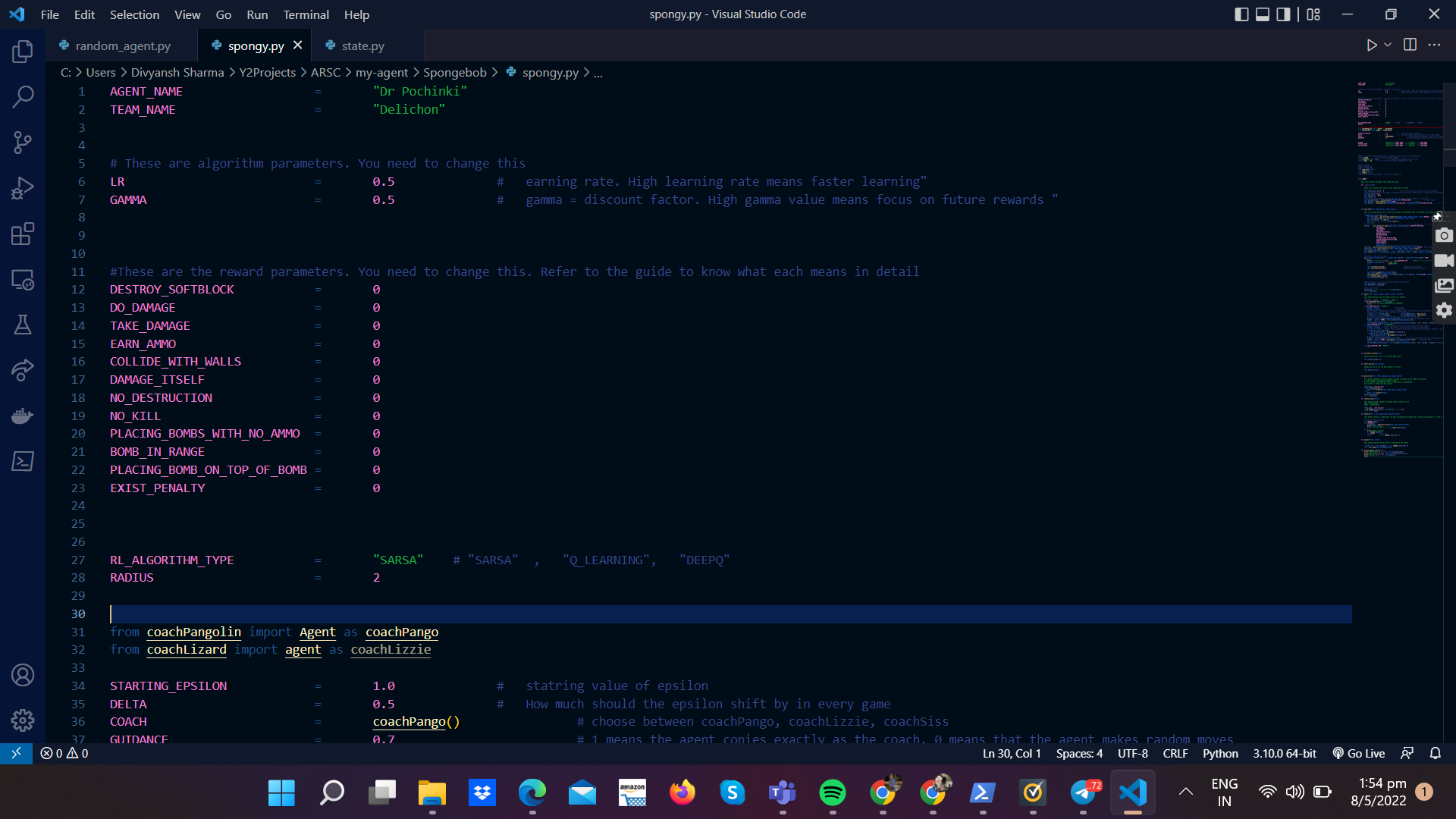


The above values are used in defining your Q-Function. They have already been defined in the tutorial. But just as a recap, we will redefine them again. Learning Rate means how an agent should treat a new experience. Should it learn immediately or should it learn slowly over a course of many actions. This value can be anywhere between 0 and 1.

Note: a high learning rate may seem attractive, but is it really good? Research about what should a learning rate be like

Gamma is a discount factor. It is the focus on future rewards and immediate rewards. Watching TV and Netflix is a quick and easy way of pleasure(immediate rewards). However, sacrificing these pleasures and replacing them with studying and hard work can earn you a high GPA(distant rewards). What should your agent be like? Research on discount factor to know more

## Reward Parameters

The above are reward parameters. They have been set to 0. This is one of the most important parts of your agent as it gives the sense of task in hand to your agent. For each value you can assign an integer or floating point number. This number can be negative, positive or 0. Negative would mean to punish the agent, while a positive reward would mean to encourage the agent. 0 means don’t care. The following is the description of each of the reward parameters.

DESTROY\_SOFTBLOCK : This reward is earned when your agent’s bomb explodes in the map, and destroys a softblock. For every softblock destroyed, this reward is earned.

DO\_DAMAGE : When your agent’s bomb explosion does damage to the enemy.

TAKE\_DAMAGE : When enemy’s agent’s bomb explosion does damage to your agent

EARN\_AMMO: When your agent earns ammo from the map

COLLIDE\_WITH\_WALLS : When your agent collides with walls. This is a waste of a move.

DAMAGE\_ITSELF : When your agent’s bomb explosion hurts your agent itself. This is a suicidal move.

NO\_DESTRUCTION : When your agent’s bomb explosion does not destroy any blocks.

NO\_KILL : When your agent’s bomb explosion does not no damage to the opponent.

PLACING\_BOMBS\_WITH\_NO\_AMMO : When your agent tries to place a bomb, but has no ammo. This is a waste of a move.

BOMB\_IN\_RANGE : If your agent finds itself in a region of bomb

PLACING\_BOMB\_ON\_TOP\_OF\_BOMB : When your agent tries to place a bomb on top of another bomb

EXIST\_PENALTY : This value needs to be negative. If it’s positive then your agent would be incentivized to stay on the map and not do anything

Note on rewards:

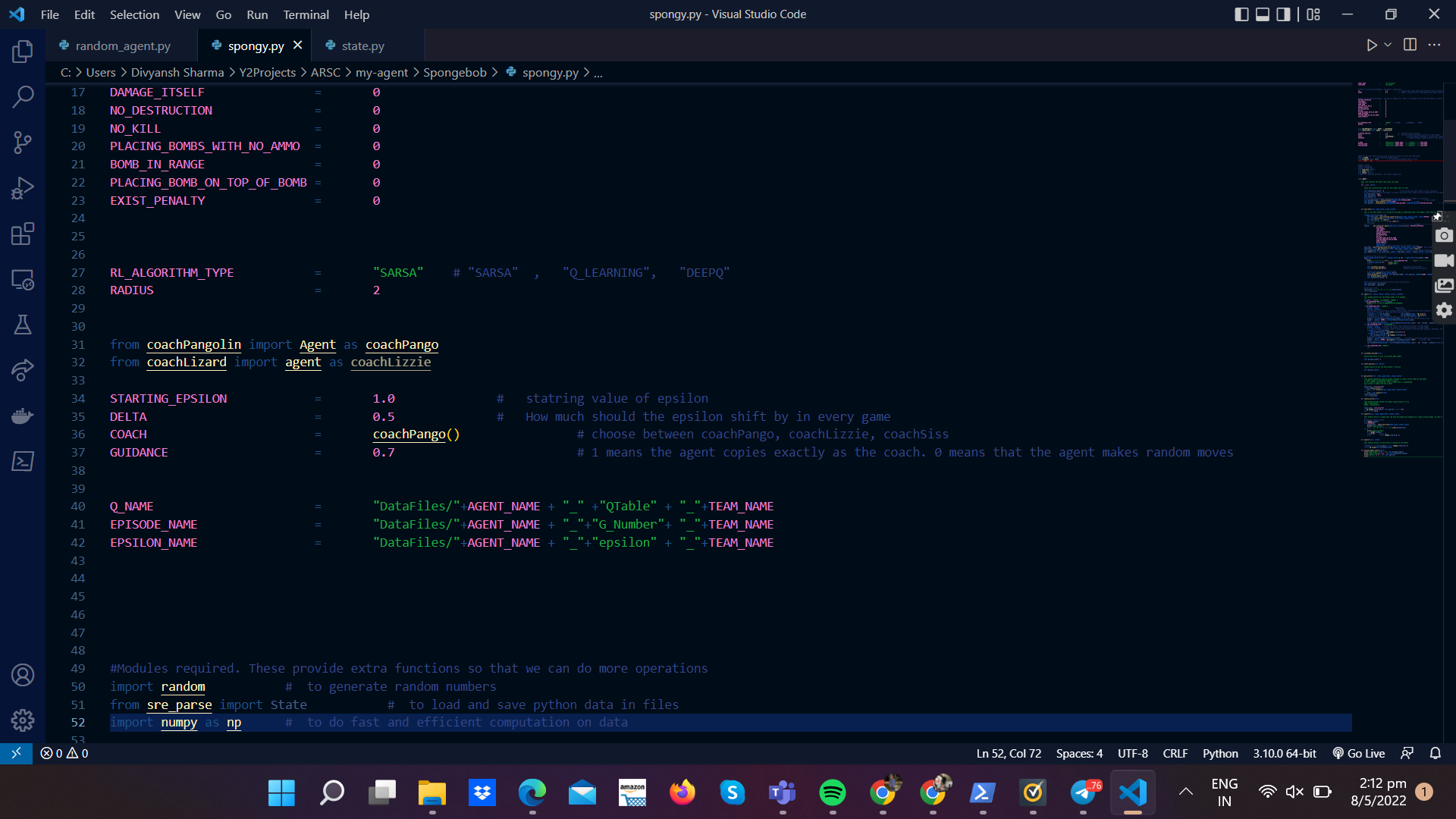
Setting up reward parameters is highly important. Make sure you consider the following :

1. Positive and negative rewards. You do not want to encourage a bad act like “kill itself. At the same time you do not want to discourage a good activity.
2. Magnitude of the rewards. These should be set relative to your other rewards. Are some rewards worth more than the others? For example, is taking ammo more worth than killing an opponent?

## IV. RL Algorithm Type

Enter “SARSA” or “Q\_LEARNING” based on which algorithm you want to use. These decide how your Q function is created.

## V. EXPLORATION vs EXPLOITATION STRATEGY



For our agent we will be using the epsilon greedy strategy. We want our agent to be a fully exploring agent at the start and then gradually and linearly transition into a fully exploiting agent. Research about the epsilon greedy strategy. The starting epsilon value divided by the delta value determines the number of matches for your agent to transition from a fully exploring agent to an exploiting agent.

We have allowed you to use a coach to learn your agent from. In our battle, two more animals have volunteered to help. Lizzie the Lizard and Pango the Pangolin are two coaches. So when your agent is exploring, it doesn’t just have an option to make random moves, it can also ask for guidance from teachers. Think about how much effect you want these coaches to have on your agent. Should the agent fully copy its coaches or should it also come up with new strategies. Do remember to check out how the coaches are working as well? What strategies have they used? GUIDANCE is the measure of how much effect do the coaches have on training. High guidance means high effect. Note that this value should be between 0 and 1.

And that’s it!

You can explore the code below to find out how we have implemented the algorithm. You can also see the files state.py and reward.py to see how the rewards and states have been calculated.

Late when you run your algorithm. Your agent will store some files. These include your Q Tables, episode number, and epsilon. Do not delete these files. They will help you to store your agent progress.

Running and training the agent

To run the agent, open your windows powershell in the same directory, and activate the windows script, just like it’s been done in the setup guide.

You can execute the following commands to run different modes of the game.

**Important!** Remember to **activate the virtual environment** *before* running the below commands. (Refer to the setup guide, step B3 (Mac))

This is the basic format of running an agent.

*\*Note that – is a double dash. Do not copy paste these commands. You will have to type them manually.*

| **Coderone-dungeon <<agent\_name/agent\_name>> <<agent\_name/agent\_name>> –watch**  For infinite matches between 2 agents in watch mode, |
| --- |
| **Coderone-dungeon <<agent\_name/agent\_name>> <<agent\_name/agent\_name>> –watch –endless**    Infinite matches between 2 agents in headless mode. In this mode, you won’t be able to watch the match, but you will get the tick progress of the agent for every game played. |
| **Coderone-dungeon <<agent\_name/agent\_name>> <<agent\_name/agent\_name>> –headless –endless**  For playing a match one step at a time |
| **Coderone-dungeon <<agent1>> <<agent2>> –single\_step –watch** |

*Note <<agent1>> and <<agent2>> are 2 names of the agents (the name of the .py file containing the agent). Remember to put the agent .py file in the same folder as the virtual environment folder.*

**How to Stop ?**

Open the running powershell and execute control c

(*On Mac:* Go to the running terminal tab and press Control + C (^ + C))

**How to Pause?**

Press enter (Mac: return) to pause and play

**How do you check if the agent is running or not?**

Remember to call the agent name properly

**How do I train my agent?**

This is something you will have to think about yourself. When you first start learning a game, do you directly take on the boss? Playing the boss on the first level can get you demotivated. You will think that the game is impossible. Same is the case for the agent. Therefore, it’s necessary to start off with something easy first and then gradually build levels. We suggest that maybe you should play games against an easy agent. For example, wanderer.py, stand-still.py. After mastering them, you can train your agent to go against the legends – coachLizzie and coachPango. You can also ask your agent to play itself.