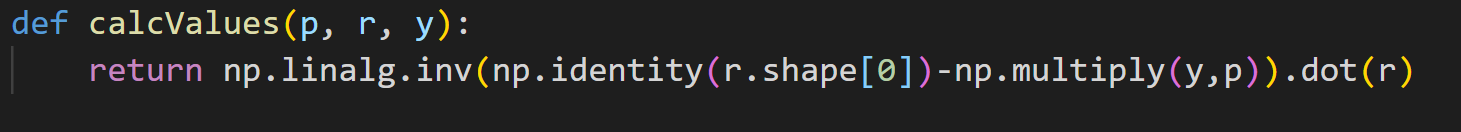
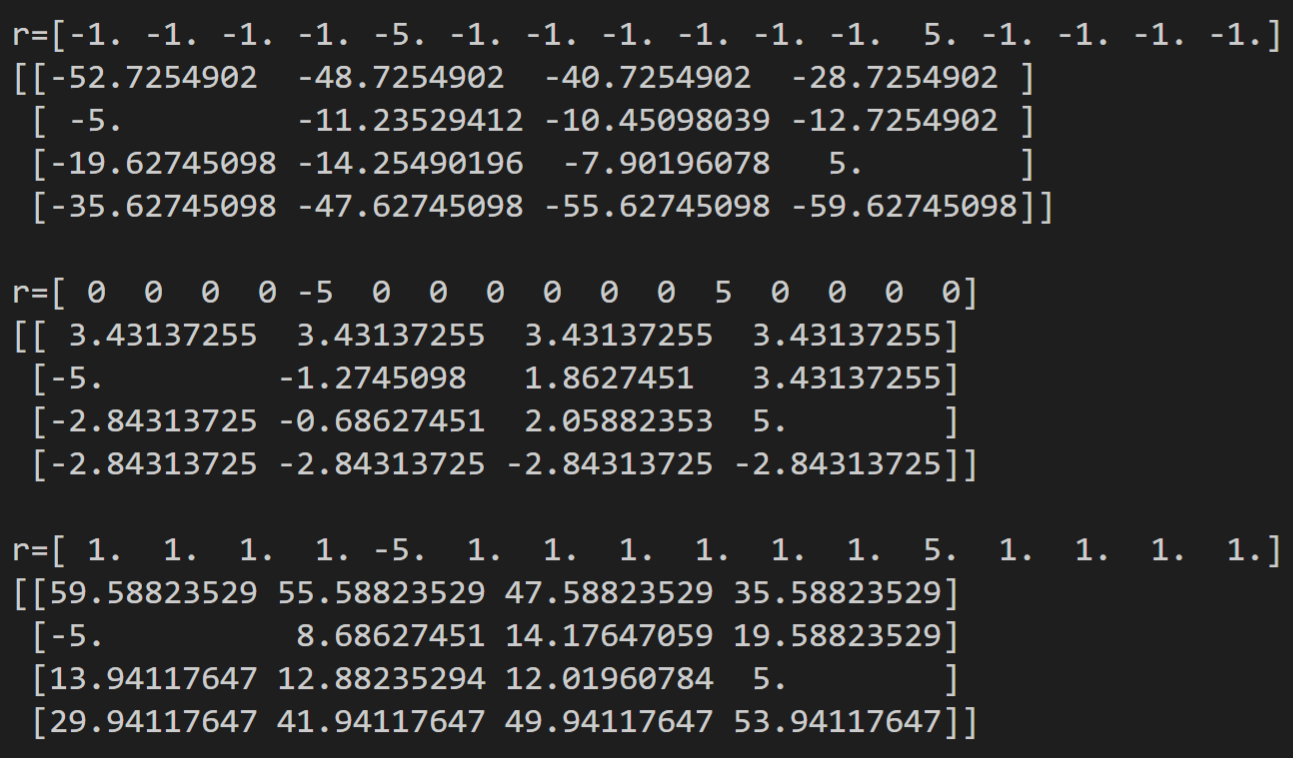
1. A.



Where ‘p’ is the probability matrix, ‘r’ is the reward list, ‘y’ is gamma.

We get the following results:

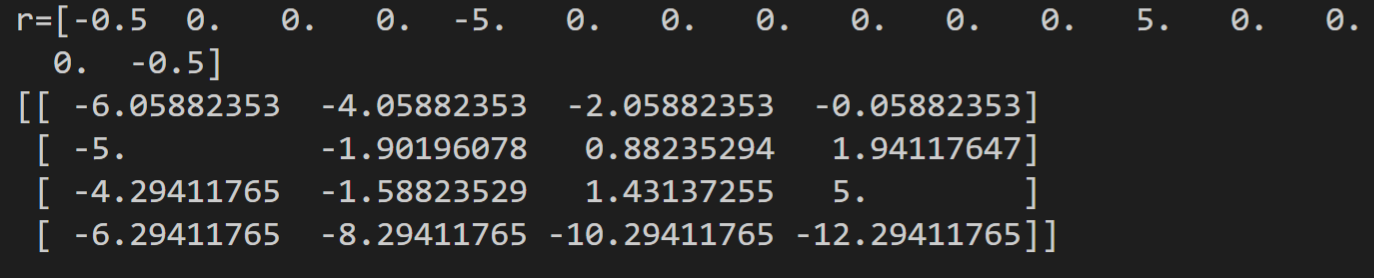


As you can see, will result in a policy that drives the agent to the green square. However, if the agent is in tile #9 it will elect to go to tile #5 which will result in a negative outcome. Therefore, if the game starts on the bottom side of the board it will end negatively.

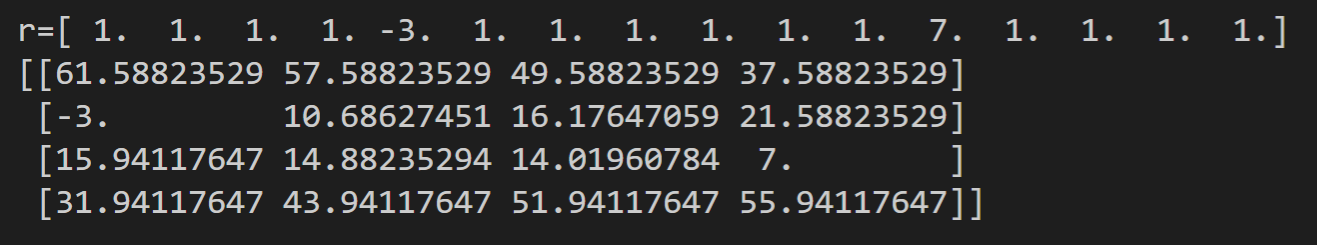
When , because our gamma is set to 1, our agent will become “long-sighted” and recognize that there is actually less reward for finishing the game. Therefore, it will avoid the goal state.

When there is no value gradient that would drive optimal policy improvement from either tiles (1,2,3,4) or (16,15,14,13). However, if the game horizon is long enough it will eventually get toward the final goal state. This is probably the best option.

If we assign individual reward values to the tiles we can have greater control over the policy creation. Setting results in a nice value function that drives agent action to the goal state regardless of starting position.

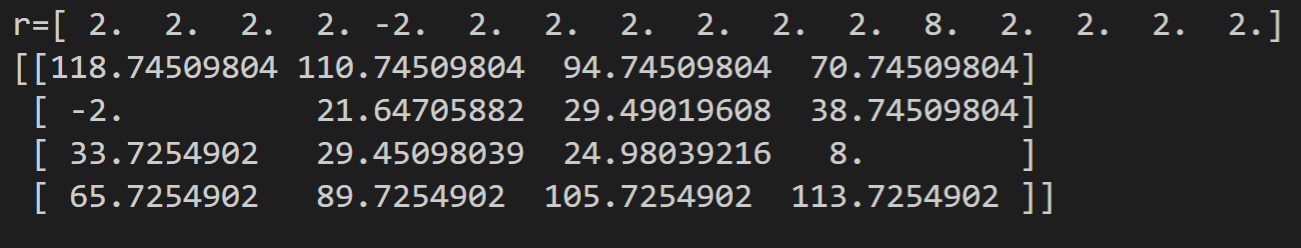


B.



Adding +2 to all rewards changes the state value function landscape drastically. The gamma value of 1 makes the actor longsighted, and exaggerates values of states that don’t terminate. As such the actor will not enter the terminal goal state. In fact, the actor will move outward from the center of the grid. It will move toward states 1 and 16 and remain there.

D.

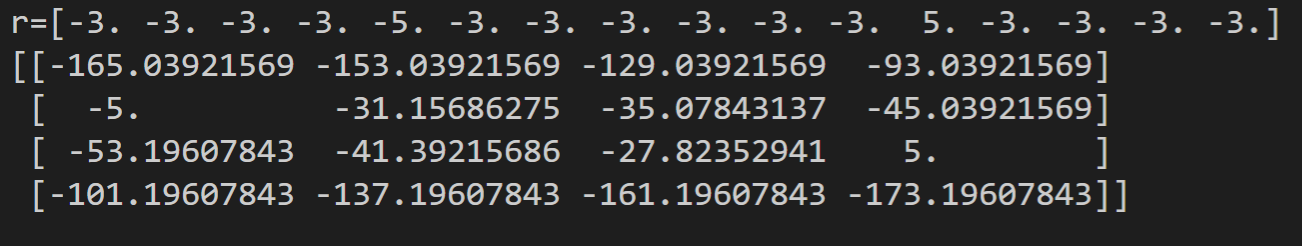


Similar to the policy derived in B, the red tile will be avoided and so will the green one. Because they terminate, this longsighted actor will move toward the highest value tiles which in this case are 1 and 16.

E.

Implementing a gamma that is changes the actor, and the policy, to one that is more short-sighted. As a result, the value of terminal states will be higher relative to the non-terminating ones. In our case, the actor will move to the goal (green) tile, while avoiding the red tile.

F.



Setting and beginning in unshaded squares #(16,15,14,13,9) will result in the actor terminating in the red shaded tile

1. A. See vi\_and\_pi.py

B. See vi\_and\_pi.py

C. Stochasticity increases the number of iterations required in order to return an ‘optimal’ policy. Moreover, the policy that is generated will not have a 100% success rate. In my implementation I also created a small experiment function that runs both value and policy iteration against each environment 200 times and gathered information about time to complete, success rate, and number of iterations. Running the experiment showed:

-----------------Deterministic-----------------

Policy Iteration Average Time to Complete: 0.019742501974105833

Policy Iteration Success Rate: 1.0

Policy Iteration Average Iterations: 7.0

Value Iteration Average Time to Complete: 0.018573970794677735

Value Iteration Success Rate: 1.0

Value Iteration Average Iterations: 7.0

------------------Stochastic-------------------

Policy Iteration Average Time to Complete: 0.09560156583786011

Policy Iteration Success Rate: 0.725

Policy Iteration Average Iterations: 7.0

Value Iteration Average Time to Complete: 0.09374362707138062

Value Iteration Success Rate: 0.68

Value Iteration Average Iterations: 27.0