Answer the following question?

1) Explain how the ε-greedy approach balances exploration and exploitation.

The random approach is purely explorative and will eventually converge to the optimal policy, however it is very slow to converge because many random samples are wasted and do not contribute to converging to the optimal policy. This is basically the fundamental idea behind probability theory in that “random” outcomes will eventually converge to some joint distribution among all possible outcomes in the long run (many trials).

The Greedy approach seldom converges to the optimal policy, but it does converge much faster than the random approach. This is because (as the name suggests) the sampling approach is greedy by *always* exploiting the short-term, current best policy from the current utility estimates (derived from most updated sample set). This can cause the system to converge to local minima as its greediness and strong current best policy bias causes the transition model estimate to converge fastest on a short-term best policy.

The E-greedy approach balances both the random and greedy approach by allowing sampling to be a combination of the greedy approach (exploitation), and the random approach (exploration). This allows the system to converge faster than random approach and maximize (exploit) utility based on current estimates (local minima, which could turn out to be true minima = optimal policy convergence), while still deferring to the random approach for sampling some percentage of the time so that local minima can be escaped (assuming the local minima is not already the optimal policy) as exploration will still occur. Because exploration (random sampling) is still involved, this sampling approach will still eventually converge to an optimal policy while having a more flexible possibility at not wasting so many samples in order to converge to a useful if not optimal policy.

Also, this sampling approach allows you to tune exactly how exploitative or explorative you want to be. If you need immediate results, you can sample with more bias (greedy) more often; if you need to attempt to have some improvement (see if you are at a local minima so that you can escape it and go to a better if not the absolute (optimal) minima) and need longer term results that can give you confidence you have converged at the optimal policy, then you can use exploration by taking more random samples than greedy samples.

2) Explain how the incremental mean method in the utility mean update reduces the memory usage.

The incremental mean is quite simple, but brilliant. One mathematically basic definition of a “mean” or “average” is to sum all data points, then divide that sum by the number of data points. So, one way to always ensure you can recalculate/update the mean whenever new data points are added is to keep track of all data points and the number of data points. From a computer science perspective, that means you have to constantly store all data points in memory. We can avoid this memory usage by using a mathematically equivalent way for updating an already calculated mean with new data. This is where the **incremental mean** comes in: if we already know the current mean, and we keep track of the total number of data points that was used to calculate the mean (the “total weight” or “divisor”), then we can compute the previous sum that was used to compute the mean

🡪 *current mean \* total weight (number of data points) = SUM of the values of all data points.*

Now that we have the sum, we can simply add any new data points to that SUM, and also increase the total weight (population) by the number of data points added to SUM. Now we are back at the definition of a mean, and we can now recalculate so that the new mean reflects the newly added data:

🡪 SUM of all data points / number of data points.

So, we tremendously save memory space by simply keep track of number of data points (population) and current mean. Alternatively, you could simply keep track of the SUM of all data points and the population, and add any new data points to that sum as mentioned before and update the population to recalculate the mean.

3) Explain the difference between the Monte-Carlo (MC) method and the Temporal-Difference (TD) method.

The MC method waits for an entire sample's trajectory to be collected before updating u (the final utility value of the sample). It uses the reward collected at each state along the sequence, including the discount factor y as a power of t (time = each data point for the trajectory collected). For example, a sensor might capture data in a sequence for each state along the trajectory and push each captured data point (reward) into a stack. Then, once it has finished collecting all of the data (the whole trajectory), it will go backward along the trajectory to update u, meaning that it will pop the data from the stack to perform the calculation that each data point from the u trajectory has on the final utility calculation for the trajectory u 🡪 once we have the whole trajectory, then we know what y^t (discount factor at time (datapoint) t) for each data point as *t* will be decremented as we move along the trajectory backwards to each data point. So, you would need to store a 1-dimensional array in memory for the entire trajectory using the MC method to calculate u (the sample utility) before you could finally use the incremental mean to calculate the new average (average utility of all u samples).



Although it costs storing the entire trajectory, the MC method approach is a more accurate estimation of u compared to the TD method since no state reward estimations are made that are not based on the entire trajectory (which would cause accuracy loss) when calculating the utility for the trajectory and thus does not require as many samples to converge to the true average utility for a state sequence U(s).

On the other hand, The TD method is a less accurate estimation of u and requires more samples for average utility U(s) to finally converge to a more accurate estimation. This is because *bootstrapping* is used, meaning that the **predictions** of the utility value of a trajectory *u* is used as the target value for the final utility value of the trajectory *u.* Therefore, you do not need to store the entire trajectory sample *u* in order to update the average expected utility U(s) for a state, s; instead, the Temporal Difference takes the reward at its start state and 1 or more average **utility** values from states along the trajectory and uses only those values (not the rewards at all states of the entire trajectory) as the estimated utility of the sample trajectory u. The temporal difference can be set such that only one utility value from a state data point of the trajectory is used, or if desired the first x data points (states) of the trajectory in order to estimate the utility of the sample trajectory *u*. The TD method then is just an estimate utility of the trajectory u as a sum of reward at the start state and the current average utilities at x data points (states) locations along the trajectory. The more data points used to estimate the trajectory utility, the more accurate the estimation, but the more memory it will require for storing that part of the (estimated) trajectory.



Essentially, you can load balance between the MC and the TD method: if you have minimal sample trajectories and more available memory, you will want to use more of the MC method and store most if not all of the trajectory in order to get more accurate estimations of the average utility U(s). If you have a lot of sample trajectories available to collect or your memory space is limited, then you can trade space for time and simply perform more calculations by only taking parts of trajectories to get less accurate sample estimations which will propagate into less accurate U(s) estimations, and then make up for the loss of accuracy by taking more trajectory samples so that you can eventually converge to a more accurate average expected utility U(s) for state s.

4) Explain how the temporal-difference (TD) method in the utility update reduces the memory usage.

The TD method in the utility update allows for reducing memory from collected trajectories (column-wise). Incremental mean reduces the need to store all trajectories (reduce row wise), and TD method allows for not having to store the entire trajectory by using only the reward for the starting state of the state sequence of the trajectory and the current estimated utility for only 1 or more subsequent states along the trajectory and their respective expected utilities. So, the TD method relies on using current average utilities of only some of the states along the trajectory of a sample in order to estimate the utility for that sample trajectory, which will then be used to update the average expected utility of U(s). This estimation of course is less accurate than using MC method as not only does it not collect an entire trajectory but also the states that it does collect as part of its trajectory it uses only the average utility of those states (which is better than reward, since at least average utility at the state will be more reasonable since utility for any state is a state sequence sum until a final state); however it trades space (memory) for time (more calculations) since you would need to collect more of these estimated sample trajectories in order to converge to a more accurate average expected utility U(s) for state s.

5) Explain in what occasion the TD Q-value update might result different results under SARSA learning and Q-Learning, respectively.

The difference between SARSA and Q-Learning is rather simple: Q-Learning is greedy(exploitative) in hopes of converging faster because it always takes the current best action at a given state along its trajectory sequence when updating Q-values. SARSA learning is more explorative since it uses not only the state along its sample trajectory but also the action as well. For example, Q-Learning and SARSA learning might both have the same state-action sample generated, but Q-Learning will only use the state and use the best action at the state regardless of the sample action at the state, while SARSA will use both the state and the action from the sample in order to update its Q values. Thus, SARSA may be more accurate in the long run and can better guarantee converging on the true expected utility, while Q-Learning risks being less accurate and it will converge faster at the prospect of more immediate (local) benefits from utility updates and possibly its convergence being at the true (optimal) expected utility values. It is very similar to the random (explorative) versus greedy (exploitative) sampling approaches situation (where E-greedy is the combination of both).