Q1: I used the PPO algorithm (using the stable-baselines3 implementation) with a small CNN as the model for processing grid “visual” information where the observation was the environment standard 7x7 field of view where each point on the grid is given by the tuple (object, color, state). The code used was largely out-of-the-box from the minigrid API.

Performance:

```
Best score: 0.984
Average score: (0.966, 0.011)
array([0.97, 0.95, 0.96, 0.95, 0.97, 0.98, 0.97, 0.96, 0.97, 0.97])
```

Q2: I used the same algorithm from Q1, and in fact even initialized the agent of Q2 using the agent of Q1. Reasoning was that the agent in Q1 already learned how to reach the door and open it, so why not load its weights into agent #2 to only have to learn to pick up the box in room #2. I experimented with different intermediate rewards and exploration rewards, but ultimately experienced a lot of noisy exploration from the exploration rewards, which delayed finding the target box. I also encountered the agent running back and forth between an intermediate reward point (the unlocked door) or taking too long to get to the target.

In the end I decided to do away with exploration rewards and only used a one-time time-discounted reward for unlocking the door to room #2 (implemented the same formula as the reward for finishing the task) and implemented it at half the weight of the task reward. I used a custom wrapper to allow for this timekeeping and custom reward. The idea was originally to create a goal for the agent to aim towards when the agent wasn’t initialized with Q1’s agent and when exploration rewards were being used, but I kept it in to keep the agent focused during the re-training for Q2. Looking at the numeric rewards and episode length also allowed me to estimate the agent’s approach and performance by inferring how quickly it achieves each of the 2 objectives.

Performance:

```
Best score: 0.984
Average score: (0.966, 0.011)
array([0.97, 0.95, 0.96, 0.95, 0.97, 0.98, 0.97, 0.96, 0.97, 0.97])
```